Introduction to Multimedia

Dave Marshall
10/4/2001

About This Course

Dave Marshall
10/4/2001

Aims of Module

To give students a broad grounding in issue surrounding multimedia, including the role of and design of multimedia Systems which incorporate digital audio, graphics and video, underlying concepts and representations of sound, pictures and video, data compression and transmission, integration of media, multimedia authoring, and delivery of multimedia.

Dave Marshall
10/4/2001

Objectives of Module

Students should be able to:

- Understand the relevance and underlying infrastructure of the multimedia systems.
- Understand core multimedia technologies and standards (Digital Audio, Graphics, Video, VR, data transmission/compression)
- Be aware of factors involved in multimedia systems performance, integration and evaluation

---

Dave Marshall
10/4/2001

Syllabus Outline

Topics in the module include the following:

1. Introduction: Multimedia applications and requirements (e.g., overview of multimedia systems, video-on-demand, interactive television, video conferencing, hypermedia courseware, groupware, World Wide Web, and digital libraries).
2. Audio/Video fundamentals including analog and digital representations, human perception, and audio/video equipment, applications.
3. Audio and video compression including perceptual transform coders for images/video (e.g., JPEG, MPEG, H.263, etc.), scalable coders (e.g., pyramid coders), and perceptual audio encoders. Application and performance comparison of various coding algorithms including hardware/software trade-offs. Image and video processing applications and algorithms.
4. Multimedia Programming Frameworks: Java for Quicktime, Java Media Framework

---

Dave Marshall
10/4/2001

Recommended Course Books

The following book is the core text for this module:

- Multimedia Communications: Applications, Networks, Protocols and Standards, Fred Halsall, Addison Wesley, 2000 (ISBN 0-201-39818-4)

  OR
- Networked Multimedia Systems, Raghavan and Tripathi, Prentice Hall, (ISBN 0-13-210642)

The following books are highly recommended reading:

- Hypermedia and the Web: An Engineering Approach, D. Lowe and W. Hall, J. Wiley and Sons, 1999 (ISBN 0-471-98312-8).
- Multimedia Systems, J.F.K, Buford, ACM Press, 1994 (ISBN 0-201-53258-1).
- Understanding Networked Multimedia, Fluckiger, Prentice Hall, (ISBN 0-13-190992-4)
- Design for Multimedia Learning, Boyle, Prentice Hall, (ISBN 0-13-242215-8)
- Distributed Multimedia:Technologies, Applications, and Opportunities in the Digital Information Industry (1st Edition) P.W. Agnew and A.S. Kellerman , Addison Wesley, 1996 (ISBN 0-201-76536-5)
- Multimedia Communication, Sloane, McGraw Hill, (ISBN 0-077092228)
- Virtual Reality Systems, J. Vince, Addison Wesley, 1995 (ISBN 0-201-87687-6)
- Encyclopedia of Graphics File Formats, Second Edition by James D. Murray and William vanRyper, O'Reilly & Associates, 1996 (ISBN: 1-56592-161-5)

The following provide good reference material for parts of the module:

Multimedia Systems

- Hyperwave:The Next Generation Web Solution, H. Maurer, Addison Wesley, 1996 (ISBn 0-201-40346).

Digital Audio

- A programmer's Guide to Sound, T. Kientzle, Addison Wesley, 1997 (ISBN 0-201-41972-6)
- Audio on the Web -- The official IUMA Guide, Patterson and Melcher, Peachpit

Press.
- The Art of Digital Audio, Watkinson, Focal/Butterworth-Heinmann.
- Synthesiser Basics, GPI Publications.
- Signal Processing: Principles and Applications, Brook and Wynne, Hodder and Stoughton.
- Digital Signal Processing, Oppenheim and Schafer, Prentice Hall.

Digital Imaging/Graphics/Video

- ***Digital video processing***, A.M. Tekalp, Prentice Hall PTR, 1995.
- ***Intro. to Computer Pictures, http://ac.dal.ca:80/ dong/image.htm*** from Allison Zhang at the School of Library and Information Studies, Dalhousie University, Halifax, N.S., Canada
- ***http://www.cica.indiana.edu/graphics/image.formats.html*** contains a comprehensive list of various graphics/image file formats.
- ***Encyclopedia of Graphics File Formats***, Second Edition by James D. Murray and William vanRyper, 1996, O'Reilly & Associates.

Data Compression

- ***The Data Compression Book***, Mark Nelson,M&T Books, 1995.
- ***Introduction to Data Compression***, Khalid Sayood, Morgan Kaufmann, 1996.
- G.K. Wallace, *The JPEG Still Picture Compression Standard*
- CCITT, *Recommendation H.261*
- D. Le Gall, *MPEG: A Video Compression Standard for Multimedia Applications*
- K. Patel, et. al., *Performance of a Software MPEG Video Decoder*
- P. Cosman, et. al., *Using Vector Quantization for Image Processing*

Animation

- Animation on the Web, S. Wagstaff, Peachpit Press, 1999 (ISBN 0-201-69687)

Multimedia Authoring

- Creating and Designing Multimedia with Director, P. Petrik and B. Dubrovsky, Prentice Hall, 1997 (ISBN 0-13-528985-8)

User Interface Design Issues

- Human Computer Interaction, A. Dix et al, Printice Hall, 1998 (ISBN 0-13-

239864)

- Designing the User Interface , B. Schneiderman, Addison Wesley, 1998 (ISBN 0-201-694497)
- Human Computer Interaction, Preece et al, Addison Wesley, 1994, 0-201-62769-8)

Multimedia Databases

- Multimedia Database Management Systems, B Prbhakaran, Kluwer, 1997, (ISBN 0-7923-9784-3).

Internet/WWW related Books

Teach Yourself Web Publishing with HTML 3.2 in 14 Days, ***Laura Lemay***, Sams.Net Publishing.

The Internet Unleashed*, J. Ellsworth, B. Baron, et al.*, Sams.Net Publishing

WWW, Internet in General

- **The Internet Complete Reference(2nd Ed.)*, H. Hahn***, McGraw-Hill
- **Webmaster in a Nutshell**, ***S. Spainhour and V. Quercia***, O'Reilly and Associates Inc.
- **Every Student's Guide to the World Wide Web**, ***K. Pitter and R. Minato***, McGraw Hill
- **Every Student's Guide to the World Wide Web (Macintosh Version)**, ***K. Pitter and R. Minato***, McGraw Hill

Internet -- Theory and Practice

- **Demystifying TCP/IP**, ***E. Taylor***, Wordware Publishing Inc.

HTML BOOKS:

- **HTML Sourcebook**, ***I.S. Graham***, Wiley and Sons
- **HTML: The Definitive Guide**, ***C. Musciano and B. Kennedy***, O'Reilly and Associates Inc.

CGI Scripts / Perl Programming BOOKS:

- **Teach Yourself CGI Programming with Perl 5 in a Week**, ***E. Herrmann***, Sams.Net

- **CGI Developer's Guide**, ***E.E. Kim***, Sams.Net
- **CGI Programming on the World Wide Web**, ***S. Gundavaram***, O'Reilly and Associates Inc.
- **Learning Perl**, ***R.L. Schwartz***, O'Reilly and Associates Inc.
- **Programming Perl**, ***L. Wall, T. Christiansen and R.L. Schwartz***, O'Reilly and Associates Inc.
- **Perl 5 Desktop Reference**, ***J. Vromans***, O'Reilly and Associates Inc.

WWW Design Issues BOOKS:

- **Designing with Style Sheets, Tables, and Frames**, ***M.E. Holzschlag***, Sams.Net
- **HTML Style Sheets Quick Reference**, ***R. Falla***, Que
- **10 Minute Guide to HTML Style Sheets**, ***C. Zacker***, Que
- **Teach Yourself Great Web Design in a Week**, ***A. Vasquez-Peterson and P. Chow***, Sams.Net
- **Designing for the Web**, ***Jennifer Niederst***, O'Reilly and Associates
- **GIF Animation Studio: Animating Your Web Site**, R. Koman, O'Reilly and Associates

A comprehensive guide to all Internet related books is available at

the ***Unofficial Internet Book List*** WWW site (URL: ***http://www.savetz.com/booklist/***)

Unofficial Internet Book List

Next Up Previous
Next: Introduction **Up:** About This Course **Previous:** Syllabus Outline
Dave Marshall
10/4/2001

Introduction

Dave Marshall
10/4/2001

History of Multimedia Systems

Newspaper were perhaps the first mass communication medium to employ Multimedia -- they used mostly text, graphics, and images.

In 1895, Gugliemo Marconi sent his first wireless radio transmission at Pontecchio, Italy. A few years later (in 1901) he detected radio waves beamed across the Atlantic. Initially invented for telegraph, radio is now a major medium for audio broadcasting.

Television was the new media for the 20th century. It brings the video and has since changed the world of mass communications.

Some of the important events in relation to Multimedia in Computing include:

- 1945 - Bush wrote about Memex
- 1967 - Negroponte formed the Architecture Machine Group at MIT
- 1969 - Nelson & Van Dam hypertext editor at Brown
- Birth of The Internet
- 1971 - Email
- 1976 - Architecture Machine Group proposal to DARPA: Multiple Media
- 1980 - Lippman & Mohl: Aspen Movie Map
- 1983 - Backer: Electronic Book
- 1985 - Negroponte, Wiesner: opened MIT Media Lab
- 1989 - Tim Berners-Lee proposed the World Wide Web to CERN (European Council for Nuclear Research)
- 1990 - K. Hooper Woolsey, Apple Multimedia Lab, 100 people, educ.
- 1991 - Apple Multimedia Lab: Visual Almanac, Classroom MM Kiosk
- 1992 - the first M-bone audio multicast on the Net
- 1993 - U. Illinois National Center for Supercomputing Applications: NCSA Mosaic
- 1994 - Jim Clark and Marc Andreesen: Netscape
- 1995 - JAVA for platform-independent application development. Duke is the first applet.
- 1996 - Microsoft, Internet Explorer.

Dave Marshall

Multimedia/Hypermedia

Dave Marshall
10/4/2001

What is Multimedia?

Multimedia can have a many definitions these include:

Multimedia means that computer information can be represented through audio, video, and animation in addition to traditional media (i.e., text, graphics drawings, images).

A good general definition is:

Multimedia is the field concerned with the computer-controlled integration of text, graphics, drawings, still and moving images (Video), animation, audio, and any other media where every type of information can be represented, stored, transmitted and processed digitally.

A ***Multimedia Application*** is an Application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.

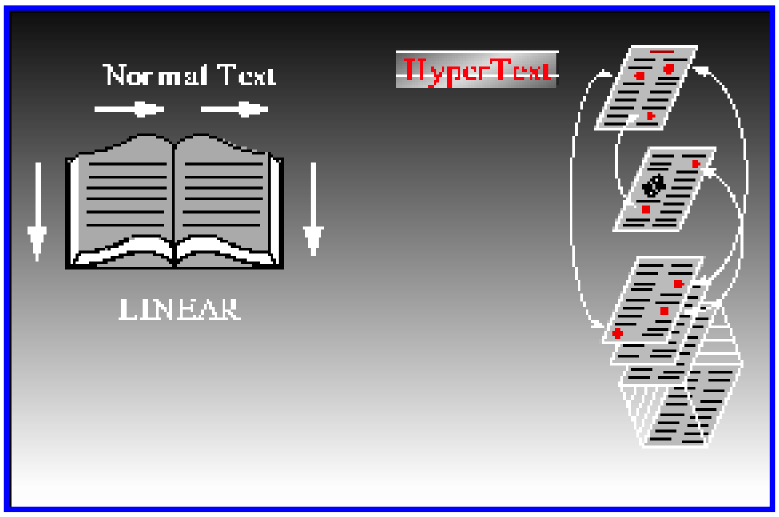
Hypermedia can be considered as one of the multimedia applications.

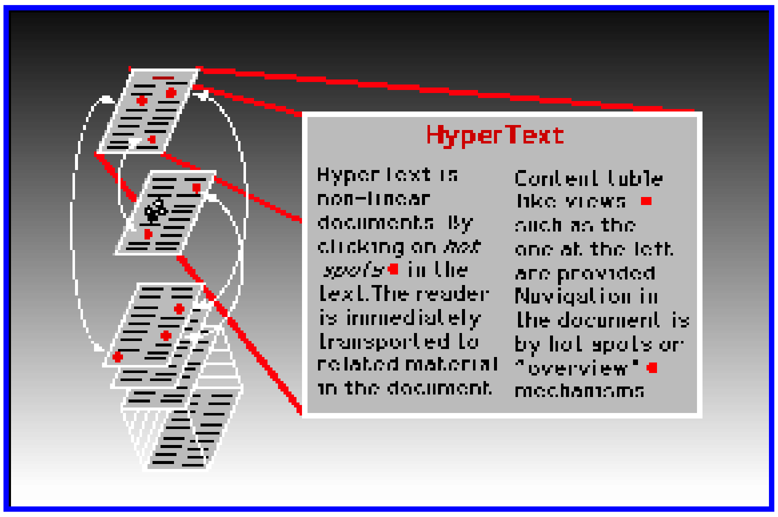
Dave Marshall
10/4/2001

What is HyperText and HyperMedia?

Hypertext is a text which contains links to other texts. The term was invented by Ted Nelson around 1965.

Hypertext is therefore usually non-linear (as indicated below).



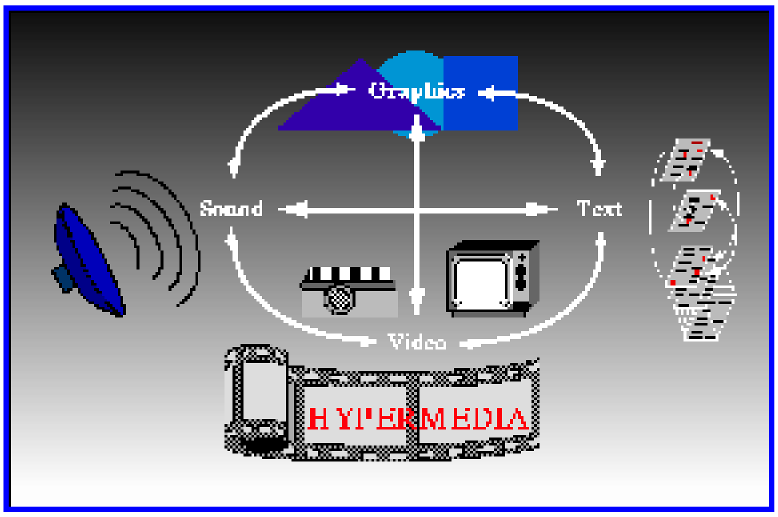

HyperText

HyperText is non-linear documents. By clicking on *hot spots* ■ in the text. The reader is immediately transported to related material in the document.

Content table like views ■ such as the one at the left are provided. Navigation in the document is by hot spots or "overview" ■ mechanisms

Definition of Hypertext

HyperMedia is not constrained to be text-based. It can include other media, e.g., graphics, images, and especially the continuous media - sound and video. Apparently, Ted Nelson was also the first to use this term.



Definition of HyperMedia

The World Wide Web (WWW) is the best example of hypermedia applications.

Dave Marshall
10/4/2001

Multimedia Systems

A ***Multimedia System*** is a system capable of processing multimedia data and applications.

A ***Multimedia System*** is characterised by the processing, storage, generation, manipulation and rendition of Multimedia information.

Dave Marshall
10/4/2001

Characteristics of a Multimedia System

A Multimedia system has four basic characteristics:

- Multimedia systems must be ***computer controlled***.
- Multimedia systems are ***integrated***.
- The information they handle must be represented ***digitally***.
- The interface to the final presentation of media is usually ***interactive***.

Dave Marshall
10/4/2001

Challenges for Multimedia Systems

Supporting multimedia applications over a computer network renders the application ***distributed***. This will involve many special computing techniques -- discussed later.

Multimedia systems may have to render a variety of media at the same instant -- a distinction from normal applications. There is a temporal relationship between many forms of media (***e.g.*** Video and Audio. There 2 are forms of problems here

- Sequencing within the media -- ***playing frames in correct order/time frame in video***
- ***Synchronisation*** -- inter-media scheduling (***e.g.*** Video and Audio). Lip synchronisation is clearly important for humans to watch playback of video and audio and even animation and audio. Ever tried watching an out of (lip) sync film for a long time?

The key issues multimedia systems need to deal with here are:

- How to represent and store temporal information.
- How to strictly maintain the temporal relationships on play back/retrieval
- What process are involved in the above.

Data has to represented ***digitally*** so many initial source of data needs to be ***digitise*** -- translated from analog source to digital representation. The will involve scanning (graphics, still images), sampling (audio/video) although digital cameras now exist for direct scene to digital capture of images and video.

The data is ***large*** several Mb easily for audio and video -- therefore storage, transfer (bandwidth) and processing overheads are high. Data compression techniques very common.

Desirable Features for a Multimedia System

Given the above challenges the following feature a desirable (if not a prerequisite) for a Multimedia System:

Very High Processing Power
-- needed to deal with large data processing and real time delivery of media. Special hardware commonplace.

Multimedia Capable File System
-- needed to deliver real-time media -- ***e.g.*** Video/Audio Streaming. Special Hardware/Software needed ***e.g*** RAID technology.

Data Representations/File Formats that support multimedia
-- Data representations/file formats should be easy to handle yet allow for compression/decompression in real-time.

Efficient and High I/O
-- input and output to the file subsystem needs to be efficient and fast. Needs to allow for real-time recording as well as playback of data. ***e.g.*** Direct to Disk recording systems.

Special Operating System
-- to allow access to file system and process data efficiently and quickly. Needs to support direct transfers to disk, real-time scheduling, fast interrupt processing, I/O streaming ***etc.***

Storage and Memory
-- large storage units (of the order of 50 -100 Gb or more) and large memory (50 -100 Mb or more). Large Caches also required and frequently of Level 2 and 3 hierarchy for efficient management.

Network Support
-- Client-server systems common as distributed systems common.

Software Tools
-- user friendly tools needed to handle media, design and develop applications, deliver media.

Dave Marshall
10/4/2001

Components of a Multimedia System

Now let us consider the Components (Hardware and Software) required for a multimedia system:

Capture devices
-- Video Camera, Video Recorder, Audio Microphone, Keyboards, mice, graphics tablets, 3D input devices, tactile sensors, VR devices. Digitising/Sampling Hardware

Storage Devices
-- Hard disks, CD-ROMs, Jaz/Zip drives, DVD, ***etc***

Communication Networks
-- Ethernet, Token Ring, FDDI, ATM, Intranets, Internets.

Computer Systems
-- Multimedia Desktop machines, Workstations, MPEG/VIDEO/DSP Hardware

Display Devices
-- CD-quality speakers, HDTV,SVGA, Hi-Res monitors, Colour printers ***etc.***

Dave Marshall
10/4/2001

Applications

Examples of Multimedia Applications include:

- World Wide Web
- Hypermedia courseware
- Video conferencing
- Video-on-demand
- Interactive TV
- Groupware
- Home shopping
- Games
- Virtual reality
- Digital video editing and production systems
- Multimedia Database systems

Dave Marshall
10/4/2001

Trends in Multimedia

Current big applications areas in Multimedia include:

World Wide Web
-- Hypermedia systems -- embrace nearly all multimedia technologies and application areas. Ever increasing popularity.

MBone
-- Multicast Backbone: Equivalent of conventional TV and Radio on the Internet.

Enabling Technologies
-- developing at a rapid rate to support ever increasing need for Multimedia. Carrier, Switching, Protocol, Application, Coding/Compression, Database, Processing, and System Integration Technologies at the forefront of this.

Dave Marshall
10/4/2001

Further Reading/Exploration

Try some good sources for locating internet multimedia examples on the Internet

For example:

- WebMuseum, Paris
- Audio Net
- BBC Web Site
- Index of Multimedia Information Sources

Dave Marshall
10/4/2001

Multimedia Authoring

Dave Marshall
10/4/2001

Multimedia Authoring:Systems and Applications

What is an Authoring System?

An Authoring System is a program which has pre-programmed elements for the development of interactive multimedia software titles. Authoring systems vary widely in orientation, capabilities, and learning curve. There is no such thing (at this time) as a completely point-and-click automated authoring system; some knowledge of heuristic thinking and algorithm design is necessary. Whether you realize it or not, authoring is actually just a speeded-up form of programming; you don't need to know the intricacies of a programming language, or worse, an API, but you do need to understand how programs work.

Dave Marshall
10/4/2001

Why should you use an authoring system?

It generally takes about 1/8th the time to develop an interactive multimedia project, such as a CBT (Computer Based Training) program, in an authoring system as opposed to programming it in compiled code. This means 1/8 the cost of programmer time and likely increased re-use of code (assuming that you pass this project's code to the next CBT project, and they use a similar or identical authoring system). However, the content creation (graphics, text, video, audio, animation, etc.) is not generally affected by the choice of an authoring system; any production time gains here result from accelerated prototyping, not from the choice of an authoring system over a compiled language.

Dave Marshall
10/4/2001

Multimedia Authoring Paradigms

The ***authoring paradigm***, or ***authoring metaphor***, is the methodology by which the authoring system accomplishes its task.

There are various paradigms, including:

Scripting Language

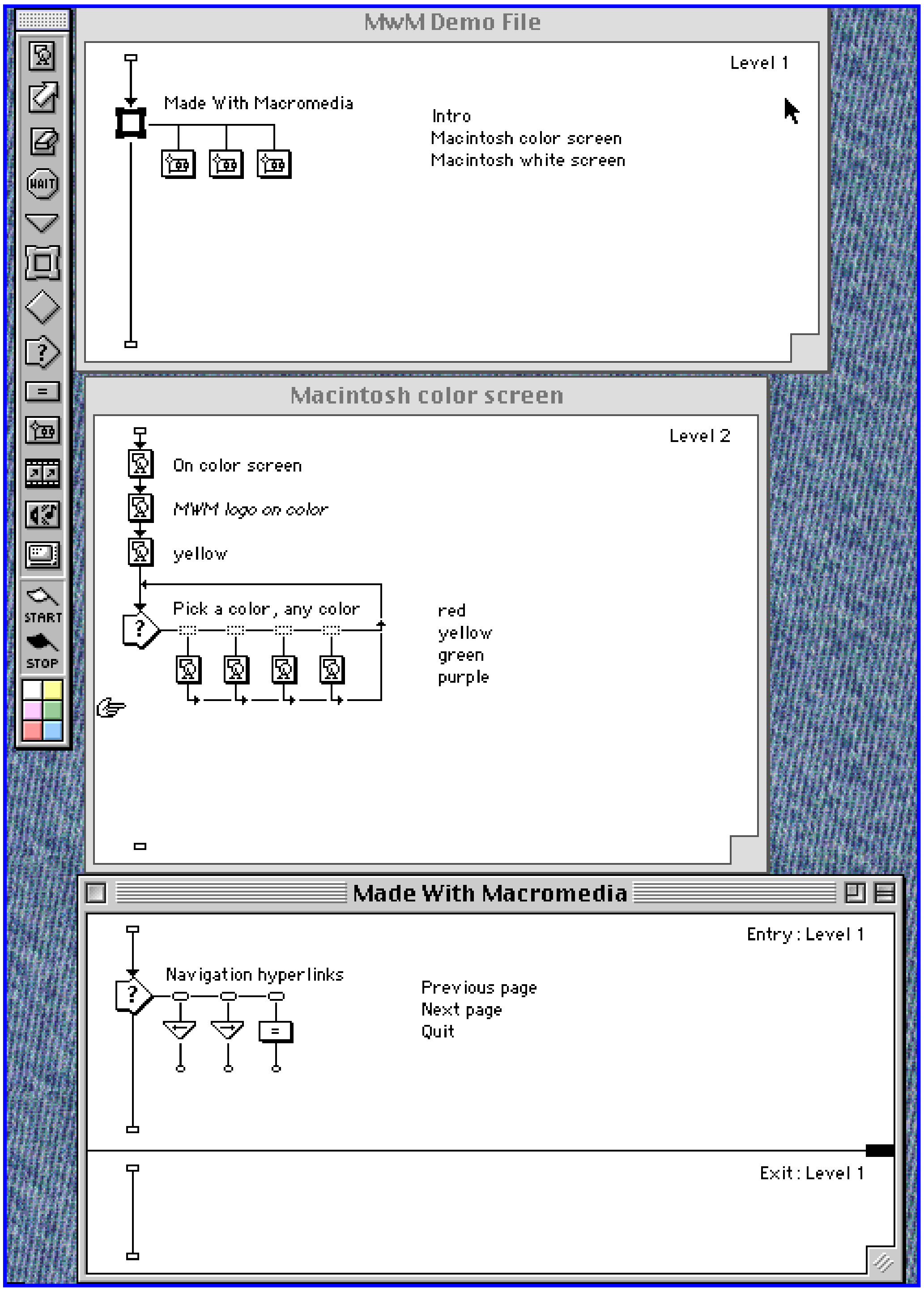
-- the Scripting paradigm is the authoring method closest in form to traditional programming. The paradigm is that of a programming language, which specifies (by filename) multimedia elements, sequencing, hotspots, synchronization, etc. A powerful, object-oriented scripting language is usually the centerpiece of such a system; in-program editing of elements (still graphics, video, audio, etc.) tends to be minimal or non-existent. Scripting languages do vary; check out how much the language is object-based or object-oriented. The scripting paradigm tends to be longer in development time (it takes longer to code an individual interaction), but generally more powerful interactivity is possible. Since most Scripting languages are interpreted, instead of compiled, the runtime speed gains over other authoring methods are minimal. The media handling can vary widely; check out your system with your contributing package formats carefully. The Apple's HyperTalk for HyperCard, Assymetrix's OpenScript for ToolBook and Lingo scripting language of Macromedia Director are examples of a Multimedia scripting language.

Here is an example lingo script to jump to a frame

```
global gNavSprite

on exitFrame
  go the frame
  play sprite gNavSprite
end
```

Iconic/Flow Control

-- This tends to be the speediest (in development time) authoring style; it is best suited for rapid prototyping and short-development time projects. Many of these tools are also optimized for developing Computer-Based Training (CBT). The core of the paradigm is the Icon Palette, containing the possible functions/interactions of a program, and the Flow Line, which shows the actual links between the icons. These programs tend to be the slowest runtimes, because each interaction carries with it all of its possible permutations; the higher end packages, such as Authorware (Fig. 2.1)or IconAuthor, are extremely powerful and suffer least from runtime speed problems.



Macromedia Authorware Iconic/Flow Control Examples

Frame

-- The Frame paradigm is similar to the Iconic/Flow Control paradigm in that it usually incorporates an icon palette; however, the links drawn between icons are conceptual and do not always represent the actual flow of the program. This is a very fast development system, but requires a good auto-debugging function, as it is visually un-debuggable. The best of these have bundled compiled-language scripting, such as Quest (whose scripting language is C) or Apple Media Kit.

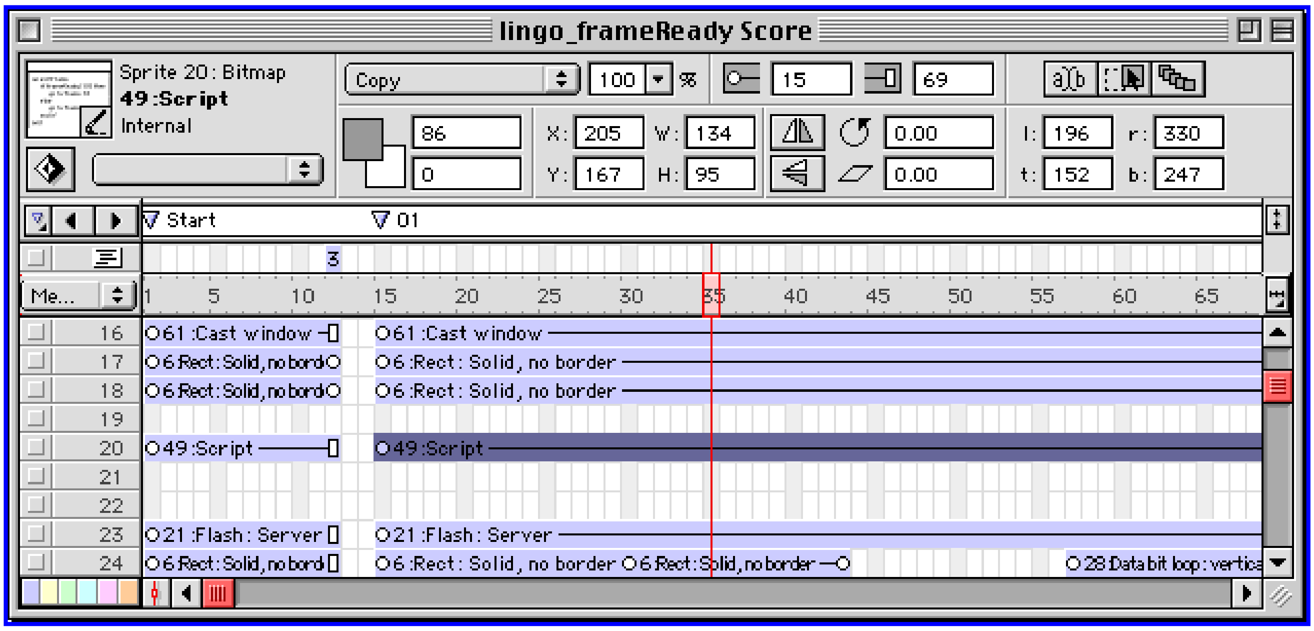
Card/Scripting

-- The Card/Scripting paradigm provides a great deal of power (via the incorporated scripting language) but suffers from the index-card structure. It is excellently suited for Hypertext applications, and supremely suited for navigation intensive (a la Cyan's "MYST" game) applications. Such programs are easily extensible via XCMDs and DLLs; they are widely used for shareware applications. The best applications allow all objects (including individual graphic elements) to be scripted; many entertainment applications are prototyped in a card/scripting system prior to compiled-language coding.

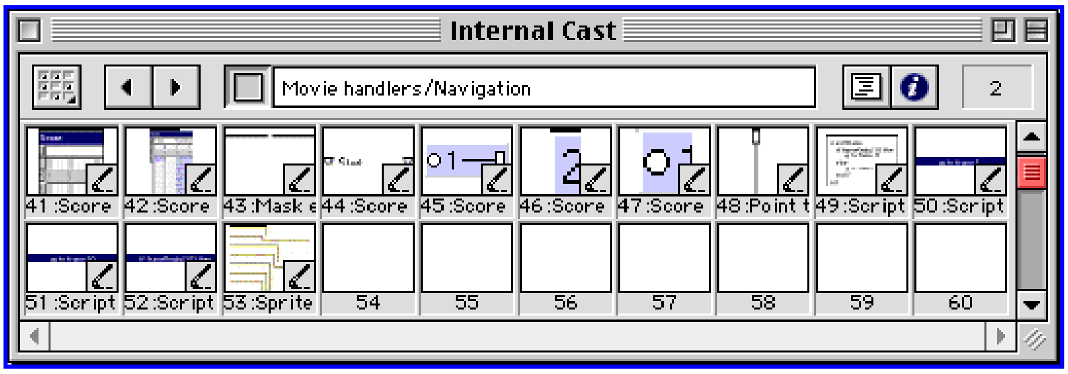
Cast/Score/Scripting

-- The Cast/Score/Scripting paradigm uses a music score as its primary authoring metaphor; the synchronous elements are shown in various horizontal ***tracks*** with simultaneity shown via the vertical columns. The true power of this metaphor lies in the ability to script the behavior of each of the cast members. The most popular member of this paradigm is Director, which is used in the creation of many commercial applications. These programs are best suited for animation-intensive or synchronized media applications; they are easily extensible to handle other functions (such as hypertext) via XOBJs, XCMDs, and DLLs.

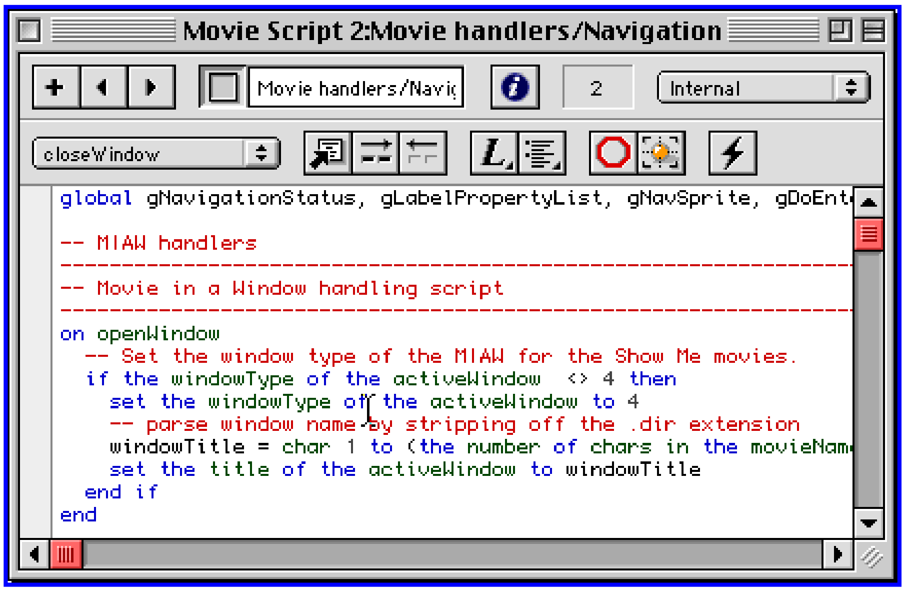
Macromedia Director uses this method and examples can be found in Figs 2.2-- 2.4


lingo_frameReady Score
Sprite 20: Bitmap
49:Script
Internal
Copy
100 %
15
69
86
0
X: 205 W: 134
Y: 167 H: 95
0.00
0.00
l: 196 r: 330
t: 152 b: 247
Start
01
16 ○61 :Cast window ○61 :Cast window
17 ○6 Rect: Solid, no border ○6 :Rect: Solid, no border
18 ○6 Rect: Solid, no border ○6 :Rect: Solid, no border
19
20 ○49 :Script ○49 :Script
21
22
23 ○21 :Flash: Server ○21 :Flash: Server
24 ○6 Rect: Solid, no border ○6 :Rect: Solid, no border ○6 Rect: Solid, no border ○28 Databit loop: vertica

Macromedia Director Score Window



Macromedia Director Cast Window



```
global gNavigationStatus, gLabelPropertyList, gNavSprite, gDoEnt

-- MIAW handlers
----------------------------------------------------------------
-- Movie in a Window handling script
----------------------------------------------------------------
on openWindow
  -- Set the window type of the MIAW for the Show Me movies.
  if the windowType of the activeWindow  <> 4 then
    set the windowType of the activeWindow to 4
    -- parse window name by stripping off the .dir extension
    windowTitle = char 1 to (the number of chars in the movieNam
    set the title of the activeWindow to windowTitle
  end if
end
```

Macromedia Director Script Window

Hierarchical Object

-- The Hierarchical Object paradigm uses a object metaphor (like OOP) which is visually represented by embedded objects and iconic properties. Although the learning curve is non-trivial, the visual representation of objects can make very complicated constructions possible.

Hypermedia Linkage

-- The Hypermedia Linkage paradigm is similar to the Frame paradigm in that it shows conceptual links between elements; however, it lacks the Frame paradigm's visual linkage metaphor.

Tagging

-- The Tagging paradigm uses tags in text files (for instance, SGML/HTML, SMIL (Synchronised Media Integration Language), VRML, 3DML and WinHelp) to link pages, provide interactivity and integrate multimedia elements.

Next Up Previous

Next: Multimedia Programming vs Multimedia **Up:** Multimedia Authoring:Systems and Applications **Previous:** Why should you use

Dave Marshall
10/4/2001

Multimedia Programming vs Multimedia Authoring

It should be noted that a distinction should be made between Programming and Authoring.

Authoring involves the assembly and bringing togther of Multimedia with possiby high level graphical interface design and some high level scripting.

Programming involves low level assembly and construction and control of Multimedia and involves real languages like C and Java.

Later in this course will will study Java programming in Quicktime and the Java Media Framework.

Quicktime may also be programmed in C.

Issues in Multimedia Applications Design

There are various issues in Multimedia authoring below we summarise issues involved in Multimedia content and technical design.

Dave Marshall
10/4/2001

Content Design

Content design deals with:

- What to say, what vehicle to use.

"In multimedia, there are five ways to format and deliver your message. You can *write* it, *illustrate* it, *wiggle* it, *hear* it, and *interact* with it."

Dave Marshall
10/4/2001

2.1.1 Scripting (*writing*)

Dave Marshall
10/4/2001

Rules for good writing:

1. Understand your audience and correctly address them.

2. Keep your writing as simple as possible. (e.g., write out the full message(s) first, then shorten it.)

3. Make sure technologies used complement each other.

Dave Marshall
10/4/2001

2.1.2 Graphics (*illustrating*)

- Make use of pictures to effectively deliver your messages.
- Create your own (draw, (color) scanner, PhotoCD, ...), or keep "copy files" of art works. - "Cavemen did it first."

Graphics Styles

- fonts
- colors
  - pastels
  - earth-colors
  - metallic
  - primary color
  - neon color

Dave Marshall
10/4/2001

2.1.3 Animation (*wiggling*)

1. **Types of Animation**

   - Character Animation - humanise an object

     e.g., a toothbrush, a car, a coke bottle, etc.

     Factors in choosing a character

     - Emotion - Is it happy, sad, funny, sloppy, ...?
     - Movement - Is it fast, slow, bumpy, ...?
     - Visual style - Is its color/texture consistent with the rest?
     - Copyright - "Don't use Mickey before checking with Walt."
     - Adequacy - e.g., Does it provide various poses (can't make a broomstick sit!)

   - Highlights and Sparkles

     e.g., to pop a word in/out of the screen, to sparkle a logo -> to draw attention
   - Moving Text

     e.g., put up one character at a time like a typewriter e.g., "pulsing" - the word grows/shrinks (or changes color) a few times

     Note: Do not slowly move entire line of text, they are not readable. Instead, for example, slide the bullets in and out.

   - Video - live video or digitized video

     +: more powerful than still images

     +: often easier to obtain than graphics animation

     -: takes a lot of disk space

     -: sometimes needs special hardware

2.

When to Animate

"A leaf doesn't flutter if the wind doesn't blow."**Only animate when it has a specific purpose**

- Enhance emotional impact

  e.g., dove softly flapping its wings -> peace

  e.g., air bag explosion + dummy movements -> car crash.

- Make a point

  e.g., show insertion of a memory chip onto the motherboard (much better than a diagram) e.g., Microsoft Golf (instructional)

- Improve information delivery

  e.g., "pulsing" words (in and out of screen) adds emphasis

- Indicate passage of time

  e.g., clock/hourglass -> program still running e.g., animated text -> to prompt for interaction/response

- Provide a transition to next subsection

  - Wipes - e.g., L-to-R, T-D, B-U, diagonal, iris round, center to edge, etc.

  - Dissolve - the current image distorts into an unrecognizable form before the next clear image appears, e.g., boxy dissolve, cross dissolve, etc.

  - Fade - a metaphor for a complete change of scene

  - Cut - immediate change to next image, e.g., for making story points using close-up ** Cuts are easy to detect in "video segmentation"

Next: 2.1.4 Audio (hearing) **Up:** Content Design **Previous:** Graphics Styles
Dave Marshall
10/4/2001

2.1.4 Audio (*hearing*)

Dave Marshall
10/4/2001

Types of Audio in Multimedia Applications:

1. Music - set the mood of the presentation, enhance the emotion, illustrate points
2. Sound effects - to make specific points, e.g., squeaky doors, explosions, wind, ...
3. Narration - most direct message, often effective

Dave Marshall
10/4/2001

Next Up Previous
Next: Types of Interactive Multimedia **Up:** Content Design **Previous:** Types of Audio in

2.1.5 Interactivity (*interacting*)

- interactive multimedia systems!
- people remember 70% of what they interact with (according to late 1980s study)

Dave Marshall
10/4/2001

Types of Interactive Multimedia Applications:

1.
Menu driven programs/presentations

- often a hierarchical structure (main menu, sub-menus, ...)

2.
Hypermedia

+: less structured, cross-links between subsections of the same subject -> non-linear, quick access to information +: easier for introducing more multimedia features, e.g., more interesting "buttons"

-: could sometimes get lost in navigating the hypermedia

3.
Simulations / Performance-dependent Simulations

- e.g., Games - SimCity, Flight Simulators

Dave Marshall
10/4/2001

Technical Design

Technologicical factors may limit the ambition of your mutlimedia presentation:

- Technical parameters that affect the design and delivery of multimedia applications

Video Mode	Resolution	Colors
CGA	320 x 200	4
MCGA	320 x 200	256
EGA	640 x 350	16
VGA	640 x 480	256
S-VGA	1,024 x 768	$>$= 256
S-VGA	1,280 x 1,024	$>$= 256

```
                              .
                              .
                              .
          16-bit color --$>$ 65536 colors
          24-bit color --$>$ 16.7 million colors
```

1. **Video Mode and Computer Platform**

   PC <-> Macintosh

   There are many "portable", "cross-platform" software and "run-time modules", but many of them lose quality/performance during the translation.

2. **Memory and Disk Space Requirement**

   Rapid progress in hardware alleviates the problem, but software is too "greedy", especially the multimedia ones.

3. **Delivery**

- Live Presentation

  Short checking list for hardware/software requirements:

  - type of graphics card
  - video memory (1 MB, 2 MB, 4 MB, etc.)
  - access time of hard disk (important for real-time video)
  - type of sound card (support for General MIDI)
  - audio-video software

- Delivery by diskette

  -: Small in size, slow to install

- Delivery by CD-ROM

  +: Large capacity -: Access time of CD-ROM drives is longer than hard-disk drives

- Electronic Delivery (ftp, www, etc.)

  - depends on baud rate, network connection, and monthly bill

---

Dave Marshall
10/4/2001

Visual Design

Here we summarise factors that should be considers in the visual design of a multimedia presentation:

1. **Themes and Styles**

   - A multimedia presentation should have a consistent theme/style, it should not be disjointed and cluttered with multiple themes.

   - The choice of theme/style depends on the styles and emotions of your audience.

   Some Possible Themes:

   - Cartoon theme

     +: interesting / entertaining -: must be consistent with the character's personality

   - Traditional theme - straightforward

     +: simple, often informative

     -: not as interesting

   - High tech theme - contemporary computer art work (morphing, texture mapping, metal texture, explosions, ...)

     +: attractive, easy to animate

   - Technical theme - include blueprints, 3D models of the product, ... e.g., start with a drawing, then transformed into a rendered image.

     +: shows adequate technical information

     +: gives impression of solid design and construction

Color Schemes and Art Styles

- Natural and floral

  (outdoor scenes, e.g., mountains, lakes, ...) -> getting back to nature

- Oil paints, watercolours, colored pencils, pastels.

  - these art styles can be combined with e.g., cartoon or high tech themes

2.

Pace and Running length

A few guidelines:

- Allow a block of text to be slowly read twice.
- Transition time should be an indication of real-time.
  - dissolve - time delay, scene change
  - cut - two views of same scene at same time, or abrupt scene change
- Running length
  - self running presentation: 2-3 minutes
  - limited interaction: 5-6 minutes
  - complete analytical, hands-on demo: < 15 minutes
  - with questions, discussions: > 30 minutes

  ** build in breaks for long presentations

3.

Basic Layout

(a) Title (b) Action area (c) Narration (d) Dialog (e) Interactive controls

- make sure that the information delivery path in the layout is smooth, not irregular/jumpy
- use headlines/subtitles, additional shapes, buttons, fonts, backgrounds and textures to enhance the visual appearance.

Dave Marshall
10/4/2001

Storyboarding

The concept of storyboarding has been by animators and their like for many years.

Storyboarding is used to help plan the general organisation or content of a presentation by recording and organizing ideas on index cards, or placed on board/wall. The storyboard evolves as the media are collected and organised: new ideas and refinements to the presentation are

---

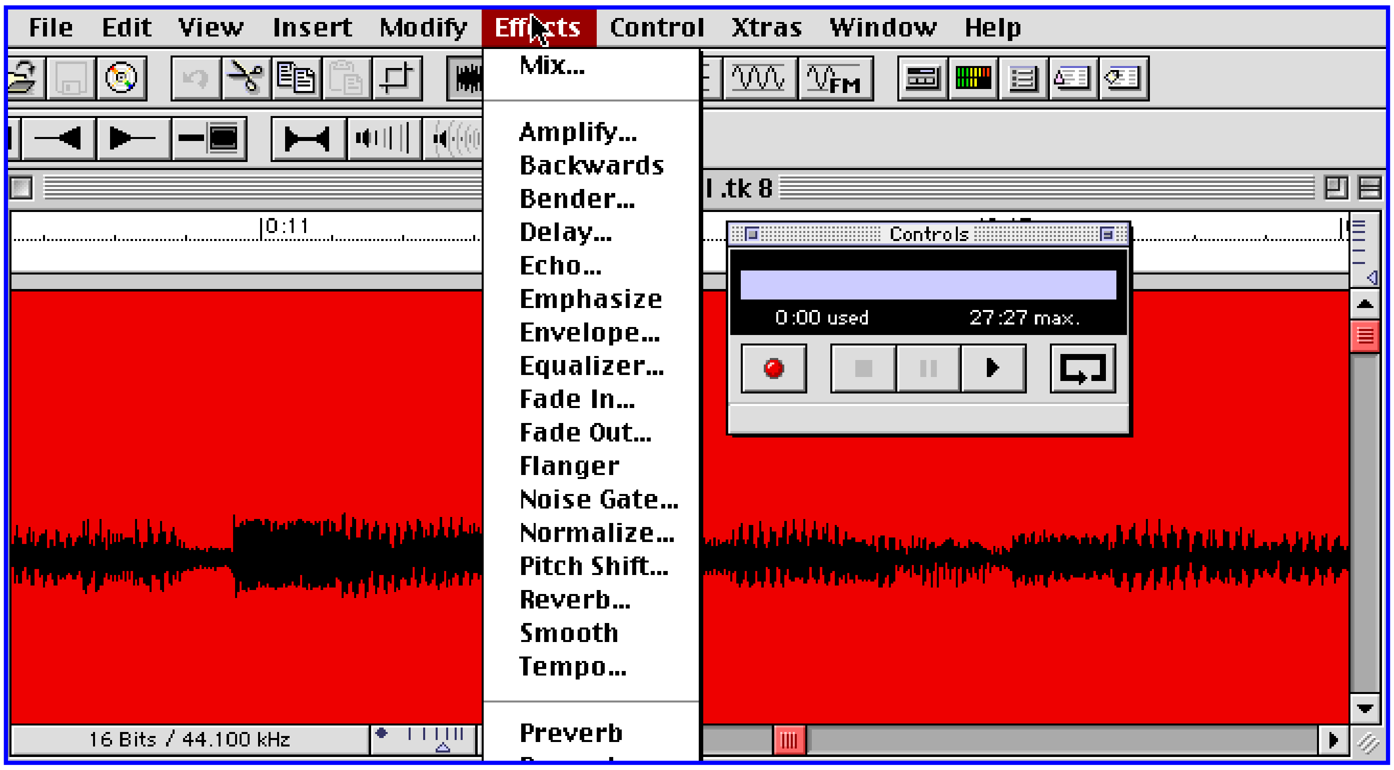
Dave Marshall
10/4/2001

Overview of Multimedia Software Tools

Dave Marshall
10/4/2001

Digital Audio

Macromedia Soundedit -- Edits a variety of different format audio files, apply a variety of effects (Fig 2.5)



Macromedia Soundedit Main and Control Windows and Effects Menu

CoolEdit -- Edits a variety of different format audio files

Many Public domain tools on the Web.

Dave Marshall
10/4/2001

Next Up Previous
Next: Image/Graphics Editing **Up:** Overview of Multimedia Software **Previous:** Digital Audio

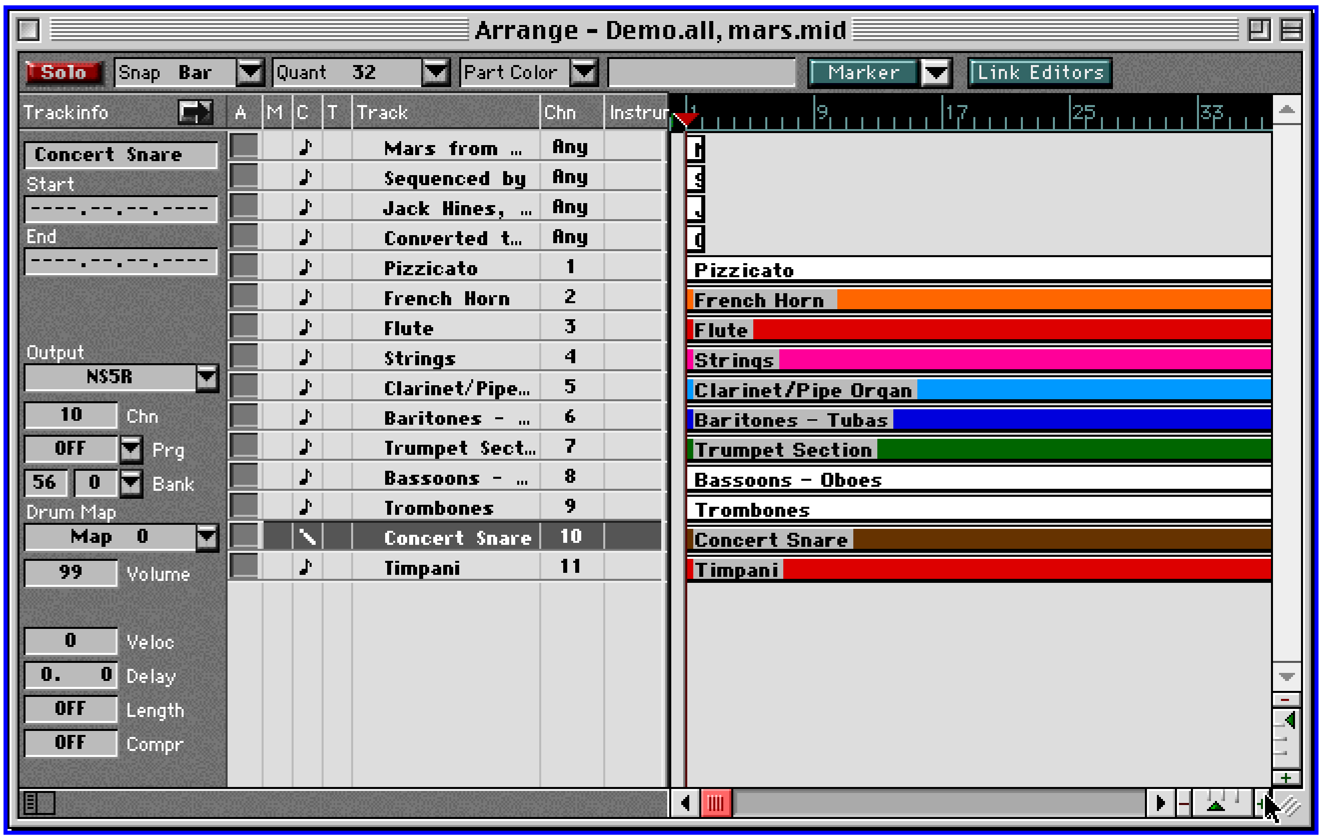
Music Sequencing and Notation

Cakewalk

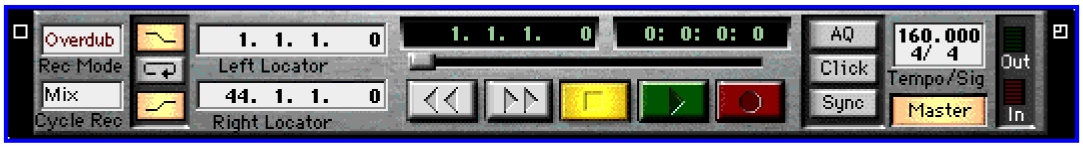
- Supports General MIDI
- Provides several editing views (staff, piano roll, event list) and Virtual Piano
- Can insert WAV files and Windows MCI commands (animation and video) into tracks

Cubase

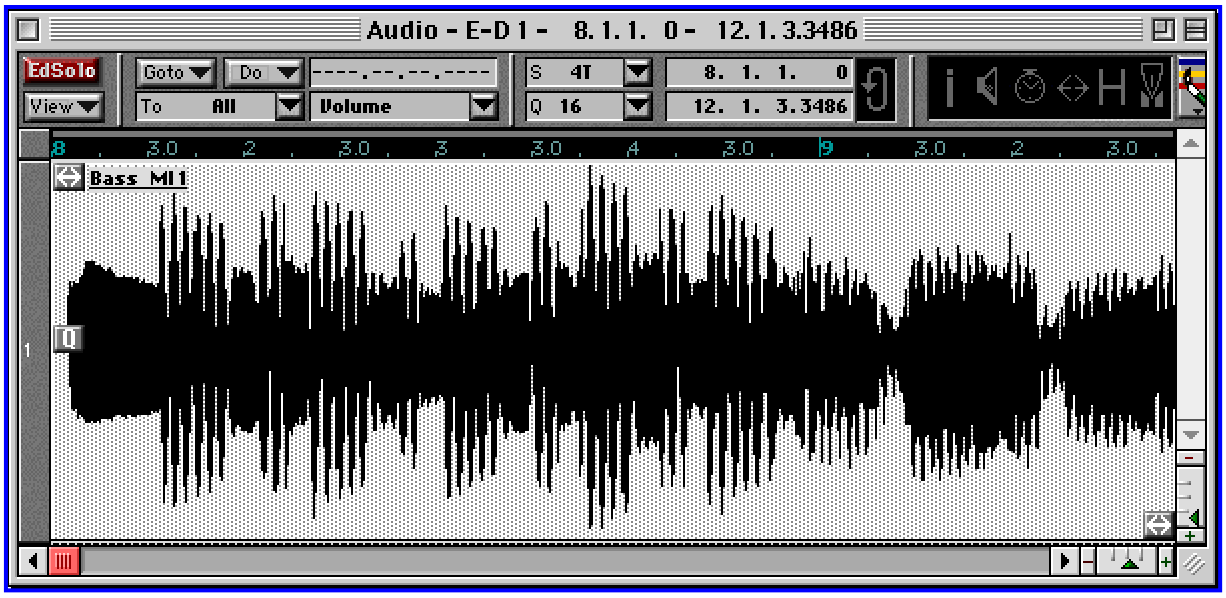
- A better software than Cakewalk Express
- Intuitive Interface to arrange and play Music (Figs 2.6 and 2.7)
- Wide Variety of editing tools including Audio (Figs 2.8 and 2.9)



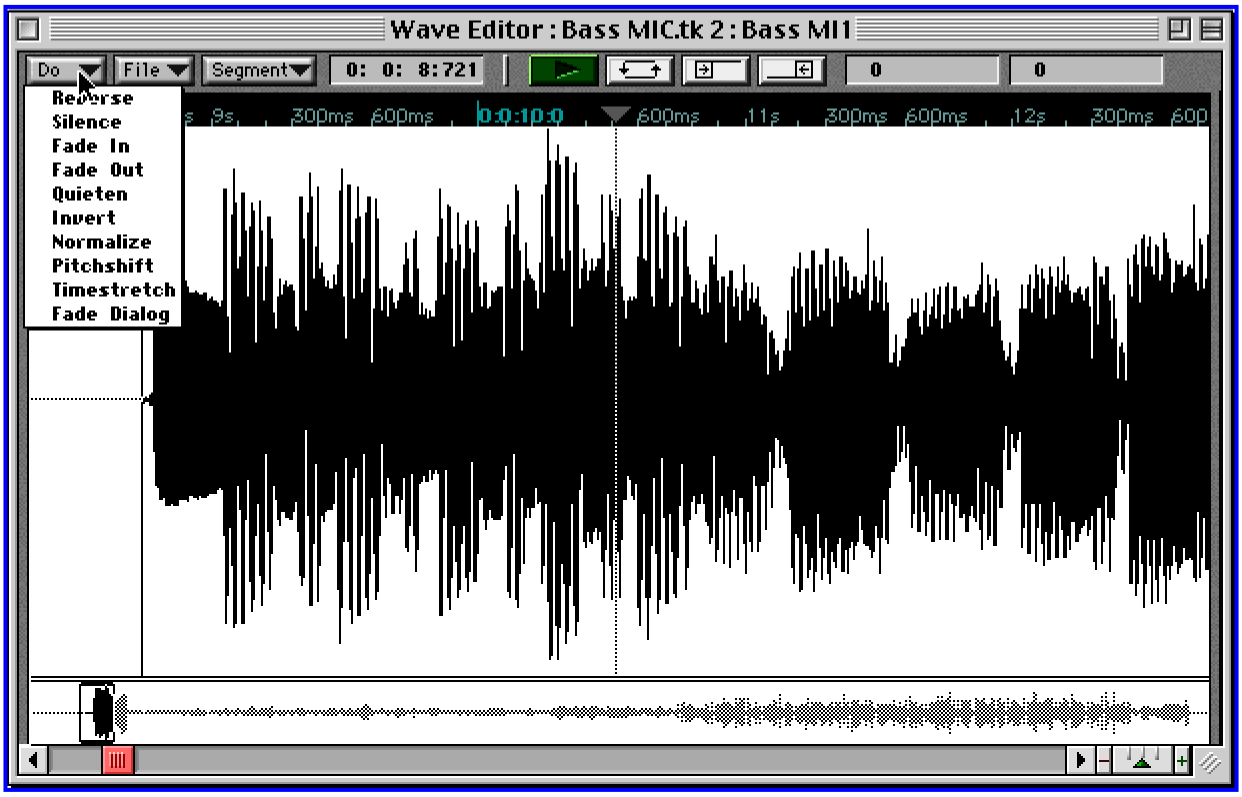
Cubase Arrange Window (Main)



Cubase Transport Bar Window --- Emulates a Tape Recorder Interface

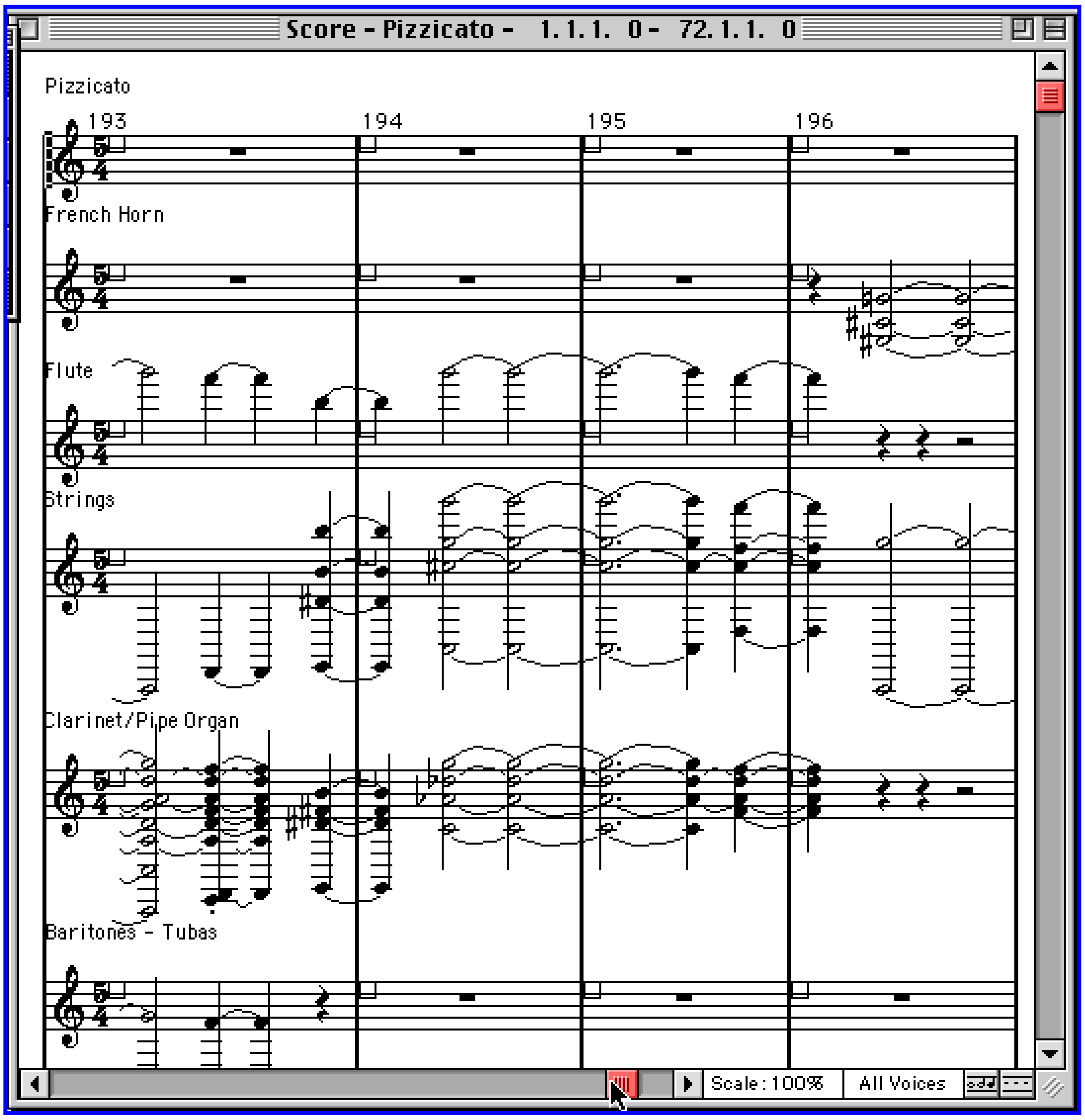


Cubase Audio Window



Cubase Audio Editing Window with Editing Functions

- Allows printing of notation sheets



Cubase Score Editing Window

Logic Audio

- Cubase Competitor, similar functionality

Marc of the Unicorn Performer

- Cubase/Logic Audio Competitor, similar functionality

Dave Marshall
10/4/2001

Image/Graphics Editing

Adobe Photoshop

- Allows layers of images, graphics and text
- Includes many graphics drawing and painting tools
- Sophisticate lighting effects filter
- A good graphics, image processing and manipulation tool

Adobe Premiere

- Provides large number (up to 99) of video and audio tracks, superimpositions and virtual clips
- Supports various transitions, filters and motions for clips
- A reasonable desktop video editing tool

Macromedia Freehand

- Graphics drawing editing package

Many other editors in public domain and commercially

Dave Marshall
10/4/2001

Image/Graphics Editing

Many commercial packages available

- Adobe Premier
- Videoshop
- Avid Cinema
- SGI MovieMaker

Dave Marshall
10/4/2001

Animation

Many packages available including:

- Avid SoftImage
- Animated Gif building packages ***e.g. GifBuilder***

Dave Marshall
10/4/2001

Multimedia Authoring

- Tools for making a complete multimedia presentation where users usually have a lot of interactive controls.

Macromedia Director

- Movie metaphor (the cast includes bitmapped sprites, scripts, music, sounds, and palettes, etc.)
- Can accept almost any bitmapped file formats
- Lingo script language with own debugger allows more control including external devices, e.g., VCRs and video disk players
- Ready for building more interactivities (buttons, etc.)
- Currently in version 7.0, this popular general market product follows the cast/score/scripting paradigm, which makes it the tool of choice for animation content. Its roots as a cel- and sprite-animation program are unmistakable; and its inclusion of Lingo, its object-based scripting language, has made it the animation-capable program to beat. The AfterBurner compression Xtra creates Shockwave files, allowing Web playback.

Authorware

- Professional multimedia authoring tool
- Supports interactive applications with hyperlinks, drag-and-drop controls, and integrated animation
- Compatibility between files produced from PC version and MAC version

Other authoring tools include:

- Microcosm : Multicosm, Ltd. ; DOS, Windows Microcosm is a Hypermedia Linkage authoring system.
- Question Mark : Question Mark Computing Ltd ; DOS, Mac, Windows; WWW (via Perception) Question Mark is optimized for Electronic Assessment production.
- Emblaze Creator : Geo International ; JavaScript, Mac, Windows95, WWW.

  Emblaze Creator 2.5 is a cast/score/scripting tool which is designed for Web-based playback of interactive multimedia.
- Flash : Macromedia ; Mac, Windows95, NT, WWW (via Flash Player).

Flash 3.0 is a cast/score/scripting tool, which primarily uses vector graphics (and can create vector graphics from imported bitmaps). It is optimized for Web delivery, and is especially common for banner adds and small interactive web deliverables.

- HyperCard : Apple Computer ; Mac, WWW (via LiveCard!).

  HyperCard is a card/scripting authoring system currently in version 2.4.1. It runs natively on both 68K and PowerMacintosh machines, and is widely used because of its easy availability at a low price. Its largest drawback is the lack of integrated color; current color implementation is via the ColorTools XCMD set (included) or via third-party XCMDs.
- HyperGASP : Caliban Mindwear.

  HyperGASP is a card/scripting authoring system currently in version 3.0; the newest version no longer requires HyperCard. Supports export to HTML for Web authoring.
- HyperStudio ; Roger Wagner Publishing ; Mac, Windows, WWW (via HyperStudio plug-in).

  HyperStudio is a card/scripting paradigm authoring system, optimized for and focussed on the educational market.
- IconAuthor : Asymetrix ; Windows, NT, Solaris, UNIX, WWW (via Windows).

  IconAuthor follows the iconic/flow control paradigm. It is notable for its SmartObject editor, which tags content files (still graphics, RTF text, etc.) for interactivity. It has the option to either embed content files or leave them external in specified directories. The biggest strength of this program is its included data handling, which makes it unparalleled for CBT data tracking. The latest version should also provide WWW porting of existing content. Avoid its internal "Move Object" path animation feature due to jerky response - use a .FLC or .AVI instead

---

Dave Marshall
10/4/2001

Further Information

See Chapter 12 ***Multimedia Presentation and Authoring*** pages 285-303, Multimedia Systems, J.F.K, Buford, ACM Press, 1994 (ISBN 0-201-53258-1).

See also:

- Mutlimedia Frequently Asked Question
- StoryBoarding Links
- Director Web

Dave Marshall
10/4/2001

Multimedia Programming:Scripting (Lingo)

In the last chapter we covered many Multimedia Authoring paradigms. Some of these basically involve only graphical programming. However, there is always a limit to how much can be achieved this ways, and so we must resort to scripting or hard programming.

We will examine two of these programming paradigms a little further in the coming 2 chapters:

- Scripting -- we will overview the Director Authoring systems and in particular it's ***LINGO*** scripting language
- Tagging -- we will overview ***SMIL*** an extension of XML for synchronised media integration.

Dave Marshall
10/4/2001

Director programming/Lingo Scripting

This section provides a very brief overview of Basic ideas of Director programming based on the ***Cast/Score/Scripting*** paradigm.

This section is essentially a precise of Director's own manual pages. You should consult:

- ***Macromedia Director 7: Using Director*** Manual -- ***In Library***
- ***Macromedia Director 7: Lingo Dictionary*** Manual -- ***In Library***
- ***Macromedia Director: Application Help*** -- Select ***Help*** from within the Director application. This is very thorough resource of information.
- ***Macromedia Director Guided tours*** -- see Help menu option.
- A variety of web sites contain director tutorials, hints and information:
  - Director Web: ***http://www.mcli.dist.maricopa.edu/director/***
  - Macromedia Director Support Center:

    http://www.macromedia.com/support/director/
  - An excellent set of Basic Director tutorial may be found at:

    http://www.fbe.unsw.edu.au/subjects/BENV/1043/tutorials.htm
  - The book ***Creating and Designing Multimedia with Director***, P. Petrik and B. Dubrovsky, Prentice Hall, 1997 (ISBN 0-13-528985-8) is also worth investigating.

Dave Marshall
10/4/2001

Director Basics

Dave Marshall
10/4/2001

Overview and Definitions

The Basic commodity in Director is the ***Director Movie***:

Director movies are interactive multimedia pieces that can include animation, sound, text, digital video, and many other types of media. A movie can be as small and simple as an animated logo or as complex as an online chat room or game.

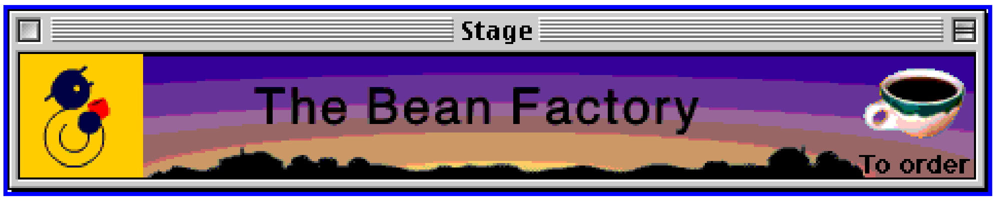
You're probably familiar with Director movies in the Shockwave movie format, which play in web browsers.

A movie may link to external media or be one of a series of movies that refer to each other. Director's interactivity lets the movie respond to events and change in specified ways.

Director divides lengths of time into a series of ***frames***, similar to the frames in a celluloid movie.

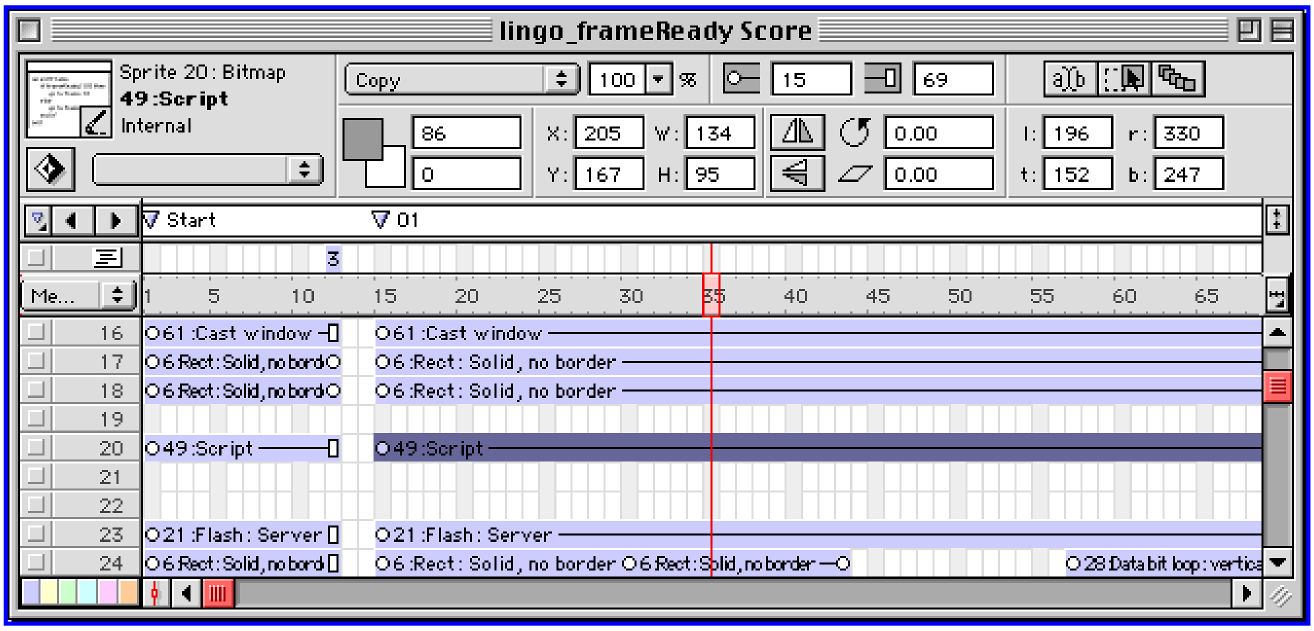
When creating and editing movies, you typically work in the four key windows that make up Director's ***work area***:

the Stage
, the rectangular area where the movie plays(Fig. 3.1):



Macromedia Director stage Window

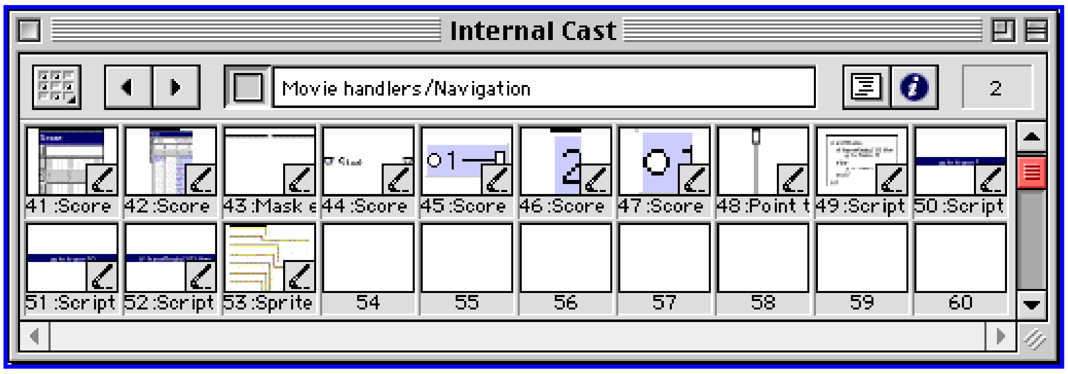
the Score
, where the movie is assembled(Fig. 3.2);



Macromedia Director Score Window

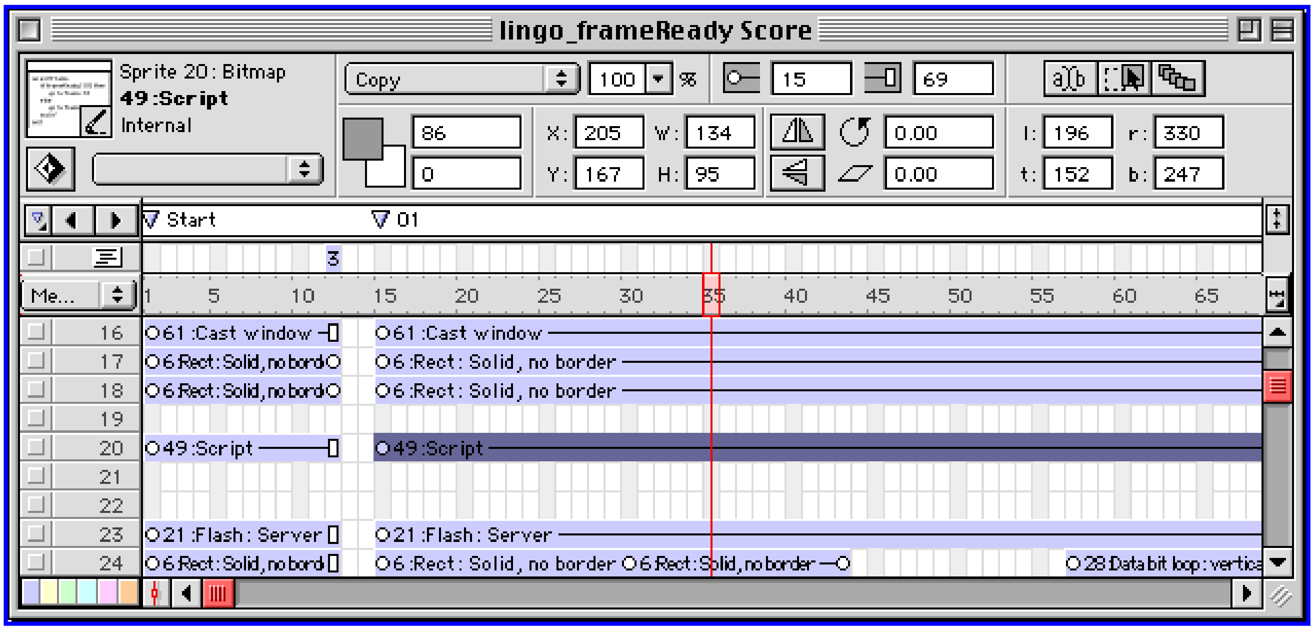
one or more Cast windows

, where the movie's media elements are assembled (Fig. 3.3);



Macromedia Director Cast Window

and
the Control Panel
, which controls how the movie plays back(Fig. 3.4).



Macromedia Director Control Panel

To create a new movie:

- Choose File > New > Movie.

Dave Marshall
10/4/2001

The Score and the Stage

The Score coordinates the movie's media, determining when images appear and sounds play. Special channels control the movie's tempo, sound, and color palettes. The Score also assigns scripts (Lingo instructions) that specify what the movie does when certain events occur in the movie (more on this shortly).

The Stage is the visible portion of a movie. Use the Stage to determine where media appears. Working together, the Score's settings and controls create a dynamic, high-quality interactive piece that plays in a web page or as a stand-alone application directly on the user's local computer.

Movie properties specify properties that affect the entire movie, such as how colors are defined, the size and location of the Stage, the number of channels in the Score, copyright information, and font mapping.

Use the Movie > Properties command to set these settings. These settings apply only to the current movie, whereas the settings you choose from File > Preferences apply to every movie. See manuals for exact properties and parameters ***etc.*****|**

Dave Marshall
10/4/2001

Using The Score

The Score organizes and controls a movie's content over time in channels. The most important components of the Score are ***channels, frames***, and the ***playback head***.

You can control the Score by zooming to reduce or magnify your view and by displaying multiple Score windows. You can also control the Score's appearance using preference settings.

To display the Score:

- Choose Window > Score.

Dave Marshall
10/4/2001

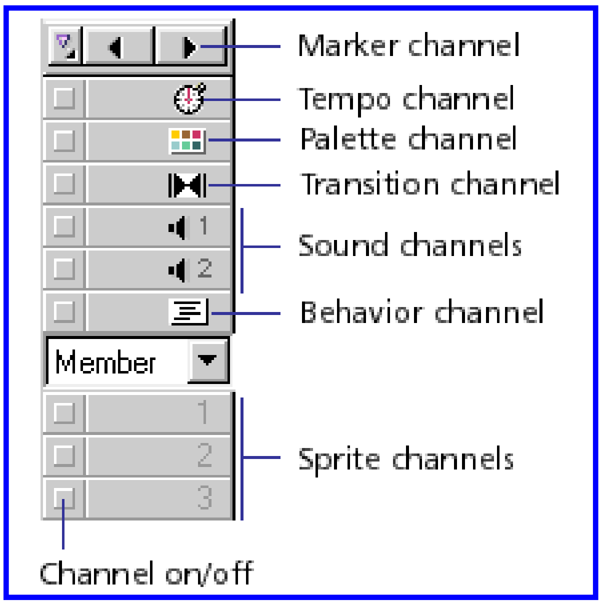
The playback head

The playback head moves through the Score to show what frame is currently displayed on the Stage. The playback head moves to any frame you click in the Score. (See Fig 3.6)

Dave Marshall
10/4/2001

Channels

Channels are the rows in the Score that contain sprites for controlling media. Sprite channels are numbered and contain the sprites that control all the visible media in the movie. Special effects channels at the top of Score contain behaviors as well as controls for the tempo, palettes, transitions, and sounds. The Score displays channels in the order shown in Fig. 3.5.



Macromedia Director Channel Display

The channel at the very top of the Score contains markers that identify places in the Score, such as the beginning of a new scene. Markers are useful for making quick jumps to certain locations in a movie.

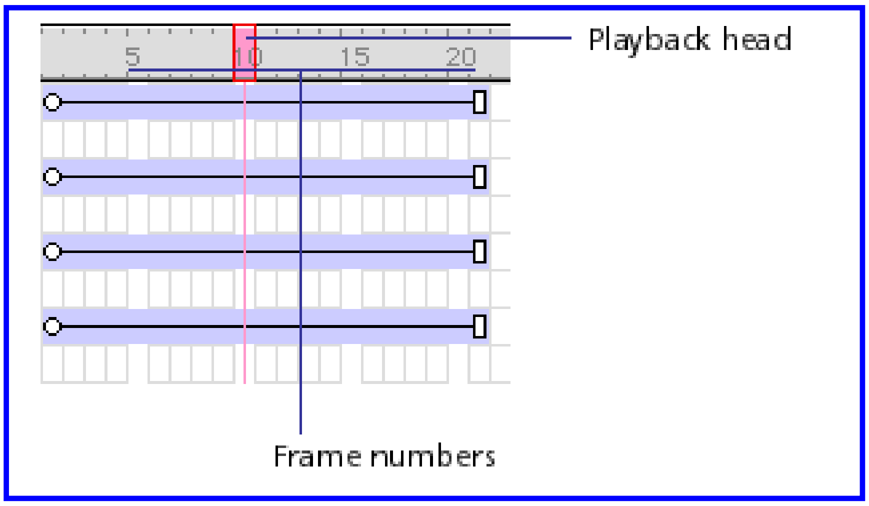
The Score can include up to 1000 channels. Most movies contain a much smaller number. To improve a movie's performance in the authoring environment and during playback, you should not use many more channels than necessary. Use Movie Properties to control the number of channels in the Score for the current movie.

Use the button at the left of any channel to hide its contents on the Stage or to disable the contents if they are not visible sprites. When you turn off a special effects channel, the channel's data has no effect on the movie. Turn Score channels off when testing performance or working on complex overlapping animations.

Dave Marshall
10/4/2001

Frames

Frames are represented by the numbers listed horizontally in the sprite and special effects channels. A frame is a single step in the movie, like the frames in a traditional film. Setting the number of frames displayed per second sets the movie's playback speed;



Macromedia Director Frame Playback

Dave Marshall
10/4/2001

Sprites

Sprites are objects that control when, where, and how media appears in a movie. The media assigned to sprites are ***cast members***.

Director organizes cast members in libraries called casts.

Creating a Director movie ***consists*** largely of defining where sprites appear on the Stage, when they appear in the movie, how they behave, and what their properties are. You work with sprites on the Stage and in the Score to change where and how cast members appear in the movie.

Dave Marshall
10/4/2001

Cast members

Cast members are the media that make up a movie. They can include bitmap images, text, vector shapes, sounds, Flash movies, digital videos, and more.

You can create cast members directly in Director or import existing media.

Dave Marshall
10/4/2001

Lingo

Lingo, Director's scripting language, adds interactivity to a movie. Often Lingo accomplishes the same tasks-such as moving sprites on the Stage or playing sounds-that you can accomplish using Director's interface.

Much of Lingo's usefulness, however, is in the flexibility it brings to a movie. Instead of playing a series of frames exactly as the Score dictates, Lingo can control the movie in response to specific conditions and events.

For example, whether a sprite moves can depend on whether the user clicks a specific button; when a sound plays can depend on how much of the sound has already streamed from the Internet.

Behaviors are pre-existing sets of Lingo instructions. Attaching behaviors to sprites and frames lets you add Lingo's interactivity without writing Lingo scripts yourself.

If you prefer writing scripts to using Director's interface and behaviors, Lingo provides an alternative way to implement common Director features; for example, you can use Lingo to create animation, stream movies from the web, perform navigation, format text, and respond to user actions with the keyboard and mouse.

Writing Lingo also lets you do some things that the Score alone can't do. For example, Lingo's lists let you create and manage data arrays, and Lingo operators let you perform mathematical operations and combine strings of text. You can also write your own behaviors that perform tasks beyond those possible with the behaviors that you already have available.

Dave Marshall
10/4/2001

Markers

Markers identify fixed locations at a particular frame in a movie. Markers are vital to navigation in movies. Using Lingo or draggable behaviors, you can instantly move the playback head to any marker frame.

This is useful when jumping to new scenes from a menu or looping while cast members download from the web. Markers are also useful while authoring to advance quickly to the next scene.

Once you've marked a frame in the Score, you can use the marker name in your behaviors or scripts to refer to exact frames. Marker names remain constant no matter how you edit the Score. They are more reliable to use as navigation references than frame numbers, which can change if you insert or delete frames in the Score.

Use the Markers window to write comments associated with markers you set in the Score and to move the playback head to a particular marker.

To create a marker:

- Click the markers channel to create a marker. A text insertion point appears to the right of the marker.
- Type a short name for the marker.

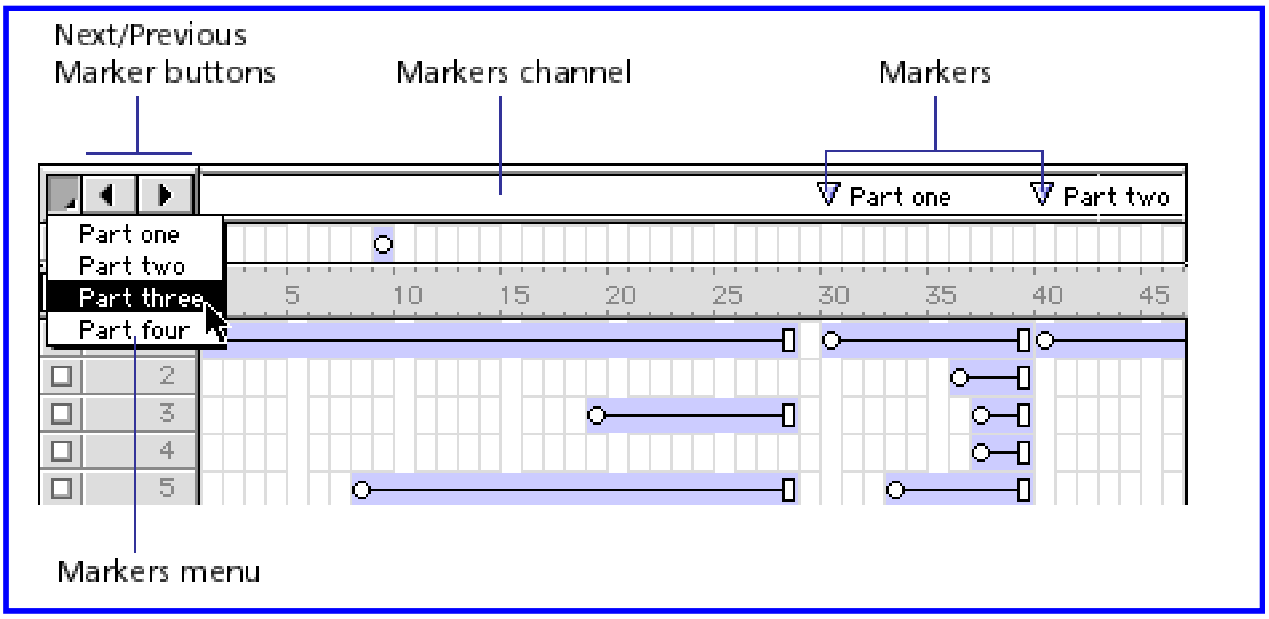
To delete a marker:

- Drag the marker up or down and out of the markers channel.

To jump to markers while authoring:

There are a few ways to do this:

- Click the Next and Previous Marker buttons on the left side of the marker channel.
- Press the 4 and 6 keys on the numeric keypad to cycle backward and forward through markers.
- Choose the name of a marker from the Markers menu.



Macromedia Director Markers

Next Up Previous

Next: Editing Frames **Up:** Multimedia Programming:Scripting (Lingo) **Previous:** Lingo

Dave Marshall
10/4/2001

Editing Frames

You can select a range of frames in the Score and then copy, delete, or paste all the contents of the selected frames. When you select frames, any sprite within the range is selected, even if it extends beyond the range. You can add new frames to a movie at any time.

To move or delete all the contents of a range of frames:

- Double-click and drag in the frame channel to select frames.
- Choose Edit > Cut or Edit > Copy, or press Delete.

  If you cut or delete the selected frames, Director removes the frames and closes up the empty space.
- To paste copied frames, select any frame and choose Edit > Paste.

Note: To delete a single frame, choose Insert > Remove Frame.

To add new frames:

- Select a frame in the Score.
- Choose Insert > Frames.
- Enter the number of frames to insert.

  The new frames appear to the right of the selected frame. If there are sprites in the frames you select, they are tweened or extended.

Dave Marshall
10/4/2001

Identifying Frames with Lingo

If you are writing scripts, use these Lingo terms to refer to frames in a movie:

- The function the frame refers to the current frame.
- The frame number or the frame marker label refers to a specific frame. For example, frame 60 indicates frame 60.
- The keyword `loop` refers to the marker at the beginning of the current segment. If the current frame has a marker, `loop` refers to the current frame; if not, `loop` refers to the first marker before the current frame.
- The word `next` or `previous` refers to the next marker or the marker before the current scene, respectively.
- The term `the frame` followed by a minus or plus sign and the number of frames before or after the current frame refers to a frame that's a specific number of frames before or after the current frame. For example, the frame - 20 refers to the frame 20 frames before the current frame.
- The function `marker()` with the number of markers used as the parameter refers to the marker that's a specific number of markers before or after the current frame. For example, `marker(-1)` gives the previous marker. If the frame is marked, `marker(0)` gives the current frame; if not, `marker(0)` gives the name of the previous marker.
- The word `movie` followed by the movie name refers to the beginning of another movie. For example,

  ```
  movie "Navigation"
  ```

  refers to the beginning of the Navigation movie.
- The word `frame` plus a frame `identifier`, the word of, the word movie, and the movie name refers to a specific frame in another movie; for example,

  ```
  frame 15 of movie "Navigation"
  ```

  refers to frame 15 of the Navigation movie.

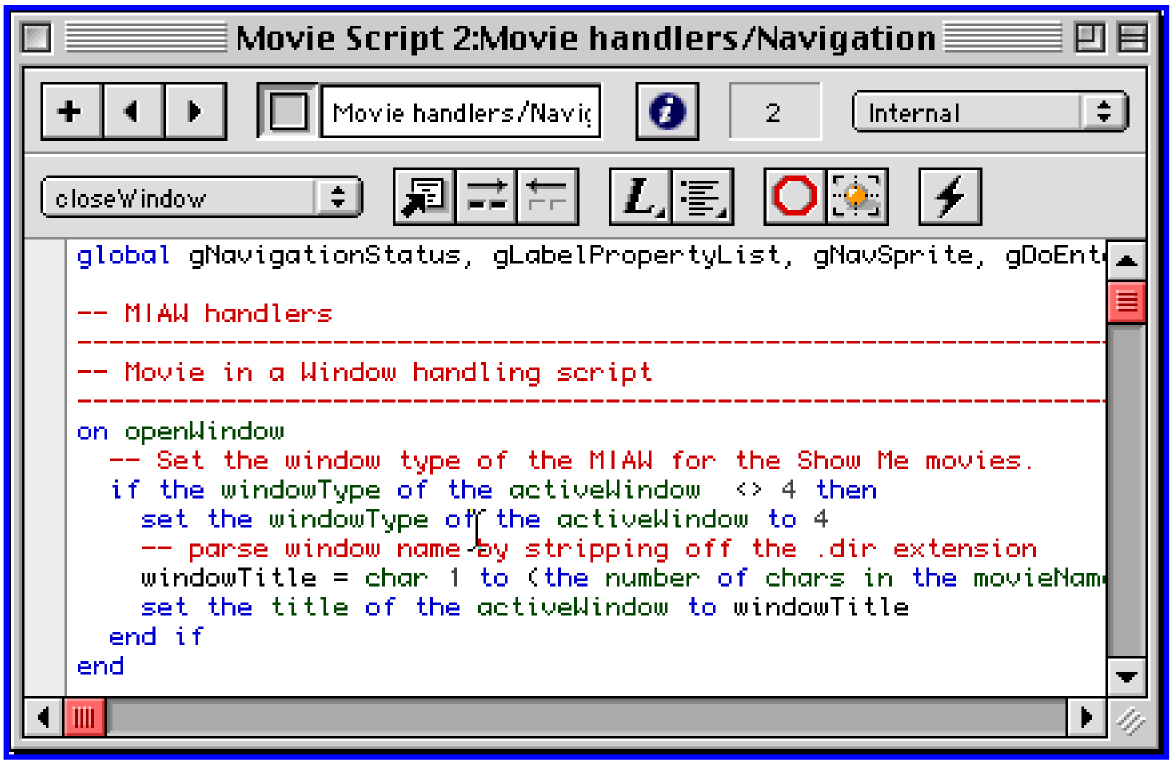
Editing Frames

Dave Marshall
10/4/2001

Lingo Scripting

Lingo, Director's scripting language, adds interactivity to a movie. Use Lingo to control a movie in response to specific conditions and events. For example, Lingo can play a sound after a specified amount of the sound has streamed from the Internet.

Use the Script window (Fig. 3.8) to write and edit scripts. (Window > Script or ***Command-O*** will display this window



```
global gNavigationStatus, gLabelPropertyList, gNavSprite, gDoEnt

-- MIAW handlers
--------------------------------------------------------------------
-- Movie in a Window handling script
--------------------------------------------------------------------
on openWindow
  -- Set the window type of the MIAW for the Show Me movies.
  if the windowType of the activeWindow  <> 4 then
    set the windowType of the activeWindow to 4
    -- parse window name by stripping off the .dir extension
    windowTitle = char 1 to (the number of chars in the movieNam
    set the title of the activeWindow to windowTitle
  end if
end
```

Macromedia Director Script Window

For an a very good introduction to scripting, see the Lingo basics tutorial movie and online help..

Dave Marshall
10/4/2001

When does Lingo run?

When an event occurs, Director generates a message that describes the event. For example, when the user types at the keyboard, a movie stops, a sprite starts, or the playback head enters a frame, these actions are events and generate event messages.

Handlers contain groups of Lingo statements that run when a specific event occurs in a movie. Each handler begins with the word on followed by the message that the handler is set to respond to. The last line of the handler is the word end; you can repeat the handler's name after end, but this is optional.

For example, the `mouseDown` message indicates that the user clicked the mouse button. A handler that started with the line on `mouseDown` contains Lingo statements that run when the handler receives a `mouseDown` message. Whether the handler receives the message depends on which objects the handler is attached to in the movie.

Director contains handlers within scripts. Attach a set of handlers to an object by attaching the handlers' script to the object.

Dave Marshall
10/4/2001

The Lingo language

Just like any scripting language, Lingo has certain elements that you use and rules that you follow.

Lingo terms fall into seven categories:

- commands -- terms that instruct a movie to do something while the movie is playing. For example, go to sends the playback head to a specific frame, marker, or another movie.
- properties -- attributes that define an object. For example`colorDepth`pth is a property of a bitmap cast member,
- functions -- terms that return a value. For example, the date function returns the current date set in the computer. The key function returns the key that was pressed last. Parentheses occur at the end of a function,
- keywords -- reserved words that have a special meaning. For example, end indicates the end of a handler,
- events,
- constants -- elements that don't change. For example, the constants TAB, EMPTY, and RETURN always have the same meaning, and
- operators -- terms that calculate a new value from one or more values. For example, the add operator (+) adds two or more values together to produce a new value.

A Lingo ***statement*** is any valid instruction that Director can execute.

An ***expression*** is any part of a statement, meant to be taken as a whole, that produces a value.

For example, `2 + 2` is an expression but is not a valid statement all by itself.

The line `go to frame 23` is a statement -- `go to` is the command, and `frame 23` is the expression that produces the value that the command requires to execute the instruction.

Lingo supports a variety of data types:

- references to sprites and cast members,

- (Boolean) values: `TRUE` and `FALSE` ,
- strings,
- constants,
- integers, and
- floating-point numbers.

Scripts can use variables to store, update, and retrieve values as the movie plays. Use the equals operator (=) or the set command to assign values to variables or change the values of many properties.

Use `if...then, case`, and `repeat` loop structures to set up statements so that they run when specific conditions exist.

For example, you can create an `if...then` structure that tests whether text has finished downloading from the Internet and then attempts to format the text if it has.

Director always executes Lingo statements in a handler starting with the first statement and continuing in order until it reaches the final statement or a statement that instructs Lingo to go somewhere else.

Some statements that send Lingo to somewhere other than the next statement are repeat loops, `ifŠthenŠelse` structures, the exit command, the return function, and handler names placed within scripts. The order in which statements are executed affects the order in which you should place statements. For example, if you write a statement that requires some calculated value, you need to put the statement that calculates the value first. For instance, in the following example, the first statement adds two numbers, and the second assigns them to a field cast member to be displayed on the Stage:

```
x = 2 + 2
put x into member "The Answer"
```

Next Up Previous

Next: Dot Syntax **Up:** Multimedia Programming:Scripting (Lingo) **Previous:** When does Lingo run?

Dave Marshall
10/4/2001

Dot Syntax

Use dot syntax to express the properties or functions related to an object or to specify a chunk within a text object. An dot syntax expression begins with the name of the object, followed by a period (dot), and then the property, function, or chunk that you want to specify.

For example, the loc sprite property indicates a sprite's horizontal and vertical position on the Stage. The expression `sprite(15).loc` refers to the `loc` property of `sprite 15`. As another example, the number cast member property specifies a cast member's number.

The expression `member("Hot Button").number` refers to the cast member number of the Hot Button cast member. Expressing a function related to an object follows the same pattern. For example, the `pointInHyperLink` text sprite function reports whether a specific point is within a hyperlink in a text sprite.

In addition to the syntax demonstrated in the Lingo Dictionary, you can use the dot syntax `textSpriteObject.pointInHyperlink` to express this function.

For chunks of text, include terms after the dot to refer to more specific items within text. For example, the expression `member("News Items").paragraph(1)` refers to the first paragraph of the `text cast member News Items`.

The expression `member("News Items").paragraph(1).line(1)` refers to the first line in the first paragraph.

Dave Marshall
10/4/2001

Parentheses

Functions that return values require parentheses. When you define functions in handlers, you need to include parentheses in the calling statement.

Use parentheses after the keywords sprite or member to identify the object's identifier:

for example, `member("Jason Jones-Hughes")`

. You can also use parentheses to override Lingo's order of precedence or to make your Lingo statements easier to read.

Dave Marshall
10/4/2001

Character spaces

Words within expressions and statements are separated by spaces. Lingo ignores extra spaces. In strings of characters surrounded by quotation marks, spaces are treated as characters.

If you want spaces in a string, you must insert them explicitly.

Uppercase and lowercase letters Lingo is not case sensitive-you can use uppercase and lowercase letters however you want. For example, the following statements are equivalent:

```
member ("Cat").hilite = TRUE
Set the hiLite of member "cat" to True
set the hilite of member "Cat" to True
SET THE HILITE OF MEMBER "CAT" TO TRUE
Set The Hilite Of Member "Cat" To True
```

However, it's a good habit to follow script writing conventions, such as the ones that are used in this book, to make it is easier to identify names of handlers, variables, and cast members when reading Lingo code.

Dave Marshall
10/4/2001

Comments

Comments in scripts are preceded by double hyphens (-). You can place a comment on its own line or after any statement. Lingo ignores any text following the double hyphen on the same line.

Comments can consist of anything you want, such as notes about a particular script or handler or notes about a statement whose purpose might not be obvious. Comments make it easier for you or someone else to understand a procedure after you've been away from it for a while. Use the Comment and Uncomment buttons in the Script window to enter and remove comments easily.

Dave Marshall
10/4/2001

Optional keywords and abbreviated commands

You can abbreviate some Lingo statements. Abbreviated versions of a command are easier to enter but may be less readable than the longer versions. The go command is a good example. All the following statements are equivalent. The last one takes the fewest number of keystrokes.

```
go to frame "This Marker"
go to "This Marker"
go "This Marker"
```

It is good practice to use the same abbreviations throughout a movie.

Dave Marshall
10/4/2001

Literal Values

A ***literal value*** is any part of a statement or expression that is to be used exactly as it is, rather than as a variable or a Lingo element.

Literal values that you encounter in Lingo are character strings, integers, decimal numbers, cast member names, cast member numbers, symbols, and constants.

Note: The value function can convert a string into a numerical value. The string function can convert a numerical value into a string.

Each type of literal value has its own rules.

Strings are characters that Lingo treats as characters instead of as variables. Strings must be enclosed in double quotation marks. For example, in the statement

```
member ("Greeting").text = "Hello"
```

``Hello" and ``Greeting" are both strings. ``Hello" is the actual string being put into a text cast member; ``Greeting" is the actual name of the cast member.

Similarly, if you test a string, double quotation marks must surround each string, as in the following example:

```
if "Hello Mr. Jones" contains "Hello" then soundHandler
```

Lingo treats spaces at the beginning or end of a string as a literal part of the string. The following expression includes a space after the word to:

```
put "My thoughts amount to "
```

Director works with integers between -2,147,483,648 and +2,147,483,647. (For numbers outside of this range, use floating-point numbers.) Enter integers without using commas. Use a minus (-) sign for negative numbers. You can convert a decimal number to an integer by using the integer() function. For example, the statement

```
set theNumber = integer(3.9)
```

rounds off the decimal number 3.9 and converts it to the integer 4.

Some Lingo commands and functions require integers for their parameters. See the entry for the specific Lingo element for more information.

A ***decimal number***, is what Lingo refers to as a floating-point number. The `floatPrecision` property controls the number of decimal places used to display these numbers. (However, Director always uses the complete number in calculations.) See the `floatPrecision` for information about setting the number of decimal places used for decimal numbers. You can also use exponential notation with decimal numbers: for example, -1.1234e-100 or 123.4e+9. You can convert an integer or string to a decimal number by using the float() function. For example, the statement

```
set theNumber = float(3)
```

stores the value 3.0 in the variable.

Next Up Previous

Next: Identifying cast members and **Up:** Multimedia Programming:Scripting (Lingo)
Previous: Optional keywords and abbreviated

Dave Marshall
10/4/2001

Identifying cast members and casts

Lingo refers to a cast member by using the term member followed by a cast member name or number in parentheses. If more than one cast member has the same name, Director uses the lowest numbered cast member in the lowest numbered cast. (Cast member names are strings and follow the same syntax rules as other strings.) An alternative syntax is the term `member` without parentheses, followed by the cast `member` name or number.

For example, the following all refer to cast `member 50`, which has the name `Jason_Jones-Hughes`:

```
member("Jason_Jones-Hughes")
member(50)
member "Jason_Jones-Hughes"
member 50
```

Use an optional second parameter to specify the cast member's cast. If more than one cast contains a cast member with the same name, you must also specify the cast. If you identify a cast member by its cast member number, you must also specify the cast. To specify a cast when using member without parentheses, include the term of `castLib` followed by the cast's name or number.

When the cast member's name is unique in the movie, the cast's name isn't required, but you can include it for clarity. For example, the following statements refer to cast member 50, which is named `Jason_Jones-Hughes`, in castLib 4, which is named `Wales_Player`:

```
member(50, 4)
member 50 of castLib 4
member("Jason_Jones-Hughes", 4)
member "Jason_Jones-Hughes" of castLib 4
member(50, "Wales_Player")
member 50 of castLib "Wales_Player"
member("Jason_Jones-Hughes", "Wales_Player")
member "Jason_Jones-Hughes" of castLib "Wales_Player"
```

If more than one cast member has the same name and you use the name in a script without specifying the cast or cast member number, Lingo uses the first (lowest numbered) cast member in the lowest numbered cast that has the specified name.

A ***symbol*** is a string or other value that begins with the pound sign (#). Symbols are user-defined constants. Comparisons using symbols can usually be performed very quickly, providing more efficient code. For example, the statement

```
userLevel = #novice
```

runs more quickly than the statement

```
userLevel = "novice"
```

Symbols can't contain spaces or punctuation. Convert a string to a symbol by using the symbol() function. Convert a symbol back to a string by using the string() function.

A ***constant*** is a named value whose content never changes. For example, TRUE, VOID, and EMPTY are constants because their values are always the same. The constants BACKSPACE, ENTER, QUOTE, RETURN, SPACE, and TAB refer to keyboard characters. For example, to test whether the user is pressing the Enter key, use the following statement:

```
if the key = ENTER then beep
```

Next Up Previous

Next: Lingo operators **Up:** Multimedia Programming:Scripting (Lingo) **Previous:** Literal Values

Dave Marshall
10/4/2001

Lingo operators

Operators are elements that tell Lingo how to combine, compare, or modify the values of an expression. They include:

- Arithmetic operators (such as +, -, /, and *)
- Comparison operators (for example, +, >, and >=), which compare two arguments
- Logical operators (not, and, or), which combine simple conditions into compound ones
- String operators (& and &&), which join strings of characters

---

Dave Marshall
10/4/2001

Lingo Lists

In Lingo, Lists provide an efficient way to track and update an array of data such as a series of names or the values assigned to a set of variables.

Lists are basically a set of elements separated by commas. Lingo encloses the set of values in square brackets. A simple example of a list is a list of numbers such as `[1, 4, 2]`.

Lingo can create, retrieve, add to, reorder, sort, or substitute a list's contents. Director offers two types of lists:

Linear lists

-- In which each element is a single value. For example, this list is a simple set of numbers:

```
[100, 150, 300, 350]
```

Property lists

-- In which each element contains two values separated by a colon. The first value is a ***property***. The second value is the ***value associated*** with that property.

For example, this list could be a sprite's Stage coordinates, with a value for each one:

```
[#left:100, #top:150, #right:300, #bottom:350]
```

Lingo has many functions that operate on lists. You can display lists (`put`), find list lengths (`count()`), find largest and smallest elements in the list (`max(), min()`), add and delete items in a list (`append, add, .....`), copy (`duplicate()`), and `sort` lists. See the Lingo dictionary for complete details..

Dave Marshall

10/4/2001

Types of Scripts

Director uses four types of scripts.

Behaviors

-- Behaviors are attached to sprites or frames in the Score. Behaviors assigned to sprites are sprite behaviors. Behaviors assigned to a frame's behavior channel are frame behaviors. Director includes a set of behaviors that are already written. Using Lingo, you can create additional behaviors for your specific needs. Behaviors are script cast members that appear in a Cast window. The Cast window thumbnail for each behavior contains a behavior icon in the lower-right corner.

Behavior Icon

All behaviors appear in the Sprite Inspector's Behavior pop-up menu. (Other types of scripts don't appear in the Behavior pop-up menu.) Attach behaviors to sprites or frames in two ways:

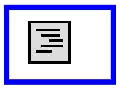
- Drag a behavior from a cast to a sprite or frame in the Score or on the Stage.
- Select the sprites or frames that you're attaching the behavior to and then choose the behavior from the Behavior pop-up menu.

You can attach the same behavior to more than one location in the Score. When you edit a behavior, the edited version is applied everywhere the behavior is attached in the Score.

Movie scripts

Movie scripts are available to the entire movie, regardless of which frame the movie is in or which sprites the user is interacting with. When a movie plays in a window or as a linked movie, a movie script is available only to its own movie. In addition to responding to events such as key presses and mouse clicks, movie scripts can control what happens when a movie starts, stops, or pauses. Handlers in a movie script can be called from other scripts in the movie as the movie plays.

Movie scripts are cast members that appear in a Cast window. A movie script icon appears in the lower-right corner of the movie script's Cast window thumbnail.



Movie script icon

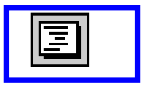
Parent scripts

Parent scripts are special scripts that contain Lingo used to create child objects. Parent scripts are cast members that appear in a Cast window. A parent script icon appears in the lower-right corner of the Cast window thumbnail:

Parent script icon

Scripts attached to cast members

Scripts attached to cast members are attached directly to a cast member, independent of the Score. Whenever the cast member is assigned to a sprite, the cast member's script is available. Unlike movie scripts, parent scripts, and behaviors, cast member scripts don't appear in the Cast window. Open scripts attached to cast members by clicking Script in the cast member's Cast Member Properties dialog box or by selecting a cast member in the Cast window and then clicking the Script button. You can also open a cast member script from the Script window.



Script button

If Show Cast Member Script Icons is selected in the Cast Window Preferences dialog box, cast members that have a script attached display a small Script icon in the lower-left corner of their thumbnails in the Cast window.

Next Up Previous

Next: Messages and Events **Up:** Multimedia Programming:Scripting (Lingo) **Previous:** Lingo Lists

Dave Marshall
10/4/2001

Messages and Events

To run the appropriate set of Lingo statements at the right time, Director must determine what is occurring in the movie and which Lingo to run in response to specific events.

Director sends messages to indicate when specific events occur in a movie, such as when sprites are clicked, keyboard keys are pressed, a movie starts, the playback head enters or exits a frame, or a script returns a certain result.

Handlers contain instructions that run when a specific message is received. The handler's name begins with the word on followed by the message name. When an object receives a message that corresponds to a handler attached to the object, Director runs the Lingo statements within the handler. For example, a handler named on `enterFrame` that is attached to a frame runs when the playback head enters the frame.

Most common events that occur in a movie have built-in message names. See the following categories in the ***Lingo Dictionary*** for the built-in messages that describe events:

- Keyboard and mouse events.
- Frame events.
- Browser and Internet events.
- Sprite events.
- Movie in a window events.
- Movie events.
- Synchronizing media events.
- Idle events.
- Timeout events.
- Authoring behavior events.

You can also define your own messages and corresponding handler names. A ***custom message*** can call another script, another handler, or the statement's own handler. When the called handler stops executing, the handler that called it resumes. Director can send a custom message from any location. The message is first available to handlers in the script from which the message was sent. If no handler is found, the message is available to movie scripts. If more than one movie script contains a handler for the message, the handler in the movie script that has the lowest cast member number is executed. A custom handler name must:

- Start with a letter
- Include alphanumeric characters only (no special characters or punctuation)
- Consist of one word or multiple words connected by an underscore-no spaces are allowed
- Not be the same as the name of a predefined Lingo element

Using Lingo keywords for handler names can create confusion. Although it is possible to explicitly replace or extend the functionality of a Lingo element by using it as a handler name, this should be done only in certain advanced situations. When you have multiple handlers with similar functions, it is useful to give them names that have similar beginnings so they appear together in an alphabetical listing, such as the listing displayed by the Find Handler option in the Edit menu.

Director follows a definite order when sending messages about events that occur during the course of a movie. When the movie first starts, events occur in the following order:

- `prepareMovie`.
- `beginSprite`. This event occurs when the playback head enters a sprite span.
- `prepareFrame`. Immediately after the `prepareFrame` event, Director plays sounds, draws sprites, and performs any transitions or palette effects. This event occurs before the `enterFrame` event. An on `prepareFrame` handler is a good location for Lingo that you want to run before the frame draws.
- `startMovie`. This event occurs in the first frame that plays.

When Director plays a frame, events occur in this order:

- `beginSprite`. This event occurs only if new sprites begin in the frame.
- `stepFrame`.
- `prepareFrame`. Immediately after the `prepareFrame` event, Director plays sounds, draws sprites, and performs any transitions or palette effects. This event occurs before the `enterframe` event.
- `enterFrame`. After `enterFrame` and before `exitFrame`, Director handles any time delays required by the tempo setting, idle events, and keyboard and mouse events.
- `exitFrame`.
- `endSprite`. This event occurs only if the playback head exits a sprites in the frame.

When a movie stops, events occur in this order:

- `endSprite`. This event occurs only if sprites currently exist in the movie.
- `stopMovie`.

A movie can contain more than one handler for the same message. Director manages this situation by sending the message to objects in a definite order.

The general order in which messages are sent to objects is as follows:

- Messages are sent first to behaviors attached to a sprite involved in the event. If a sprite has more than one behavior attached to it, behaviors respond to the message in the order in which they were attached to the sprite.
- Messages are sent next to a script attached to the cast member assigned to the sprite.
- Messages are then sent to behaviors attached to the current frame.
- Messages are sent last to movie scripts.

When a message reaches a script that contains a handler corresponding to the message, Director executes the handler's instructions.

After a handler intercepts a message, the message doesn't automatically pass on to the remaining locations. (You can use the pass command to override this default rule and pass the message to other objects.) If no matching handler is found after the message passes to all possible locations, Director ignores the message.

The exact order of objects to which Director sends a message depends on the message. See the message's ***Lingo Dictionary*** entry for details about the sequence of objects to which Director sends specific messages.

You can place handlers in any type of script. However, the following are some useful guidelines for many common situations:

- ***To set up a handler that affects a specific sprite or runs in response to an action on a specific sprite***:

  Put the handler in a behavior attached to the sprite.
- ***To set up a handler that should be available any time that the movie is in that frame***:

  Put the handler in a frame script attached to the frame.

  For example, to have a handler respond to a mouse click while the playback head is in a frame, regardless of where the click occurs, place an `on mouseDown` or `on mouseUp` handler in the frame script rather than a sprite script.
- ***To set up a handler that runs in response to an event that affects a cast member, regardless of which sprites use the cast member***:

  Put the handler in a cast member script.
- ***To set up a handler that runs in response to messages about events anywhere in the movie***:

Put the handler in a movie script.

A script can contain multiple handlers. It's a good idea to group related handlers in a single place, though, for easier maintenance.

Next Up Previous

Next: Director Example 1: Simple **Up:** Multimedia Programming:Scripting (Lingo)
Previous: Types of Scripts

Dave Marshall
10/4/2001

Director Example 1: Simple Animation

This example does not employ any scripting. It illustrates how easy it is to create a basic animation where a cast member (a ball graphic) is placed at different locations in the scene. We also introduce some features of Director that we have not yet met (these should be fairly straightforward to understand). **Note:** there are still many features of Director we will have not touched.

---

example1.dcr

Download This Example

- Shockwave Movie
- Director Movie

The following steps achieve a simple bouncing ball animation along a path:

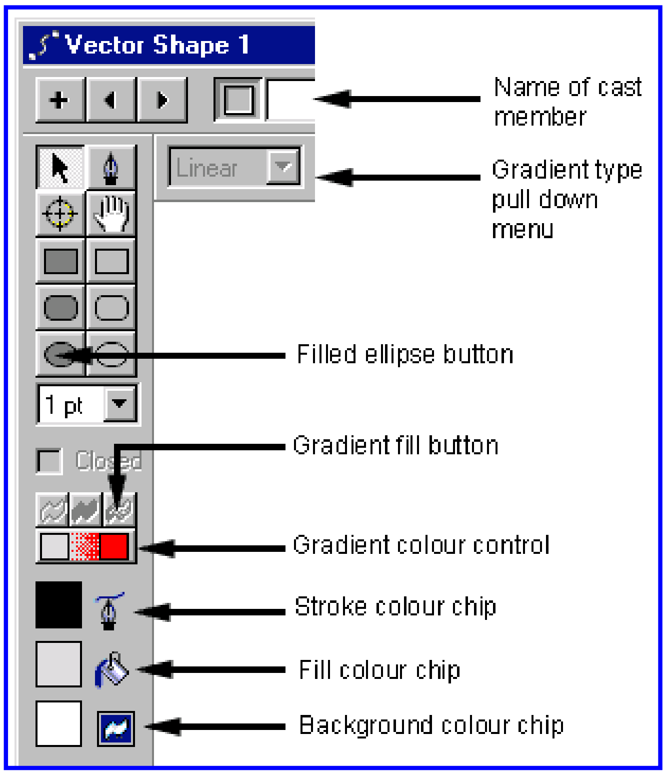
1.
Let us begin by creating a new movie and setting the Stage size:

- Start a New movie: File >New >Movie (Shortcut = Command+N)
- Choose Modify >Movie >Properties. In stage size, choose 640 x 480.

2.

Now let us create a ball, using a the vector shape tool:

- Choose Window >Vector Shape (Shortcut = Command+Shift+V)



Macromedia Director Vector Shape Window

- Click the filled ellipse button.
- Draw an ellipse (circle) about the size of the Vector Shape Window - don't worry about being exact, we will be changing the size of it later.
- Click on the Gradient fill button. This fills the ellipse with the default colours, which happen to be a light grey to red (unless someone has changed it on your computer).
- To change the colours, click the colour box on the left side of the Gradient colour control and choose a sky blue from the colour menu. You will notice the Fill colour chip changes too. Change the colour on the right side of the Gradient Colours to a dark blue.
- Change the Gradient type pull-down menu at the top of your window from Linear to Radial.
- Change the Stroke Colour to white - notice how the outline of the ellipse disappears.

3.

Now let us change a few other properties of this ellipse - for us to compare these changes, we will make a copy of this cast member.

- Close the Vector Shape window.
- In the Cast Window, select the ellipse. Choose Edit >Duplicate (Shortcut = Command+D). A copy of the cast member is produced in the next available cast slot. Double click the new cast, which opens it in the Vector Shape Tool.
- Change the Cycles to 3 and the Spread to 200. Click the Previous Cast button and compare the 2 ellipses. Experiment with different cycles and spreads to get an idea of what they mean.
- Name the latest ellipse to 'bouncing ball' - this can either be done at in the Vector Shape window or the Cast Member Window.

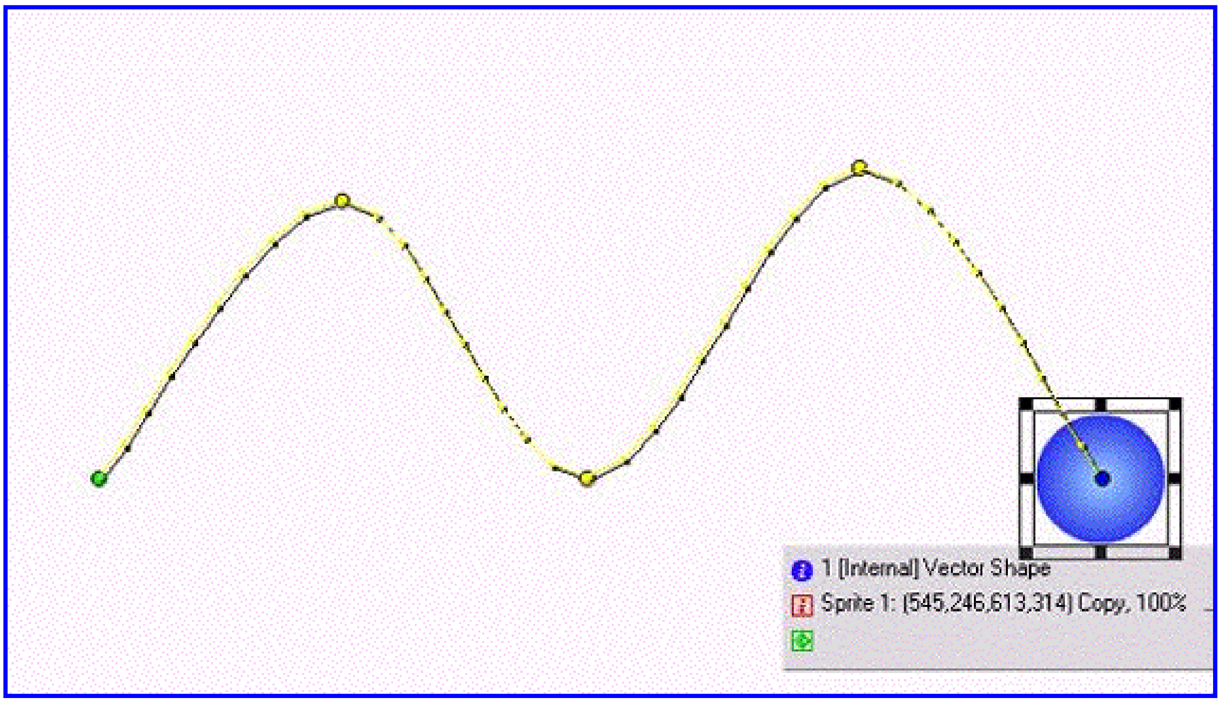
4.

Now we are going to animate the ball.

- Drag 'bouncing ball' from the cast member window to the stage. You could drag and drop it straight into the score, which would automatically centre it on the stage.
- You will notice the sprite (the object that appears in the score) is extended over 20 frames. This is a default setting that we can change. Drag the right end of the sprite to frame 40.
- Click anywhere in the middle of the sprite to select it. We are now going to resize the ellipse. Click on the top of the stage window to make it active. Press Shift and while still holding down, click on a corner handle of the spite and drag it in to make it smaller. Holding down Shift lets us resize the

object in proportion to its original dimensions. Resize the sprite to approximately the size shown in diagram 2, and move it to the left side of the stage.

- Click on frame 40 in channel 1 (the end of the sprite), hold down Option and shift and drag the ellipse to the right end of the stage. (Holding down shift restricts the movement to 90 degrees). With Option selected You will notice a line being drawn on the stage - this is the animation path. Rewind and play the movie to see what you made.
- To curve the path, we are going to insert keyframes within the sprite. Click on frame 10 of the sprite and choose Insert >Keyframe (Shortcut = Command+Option+K) Create keyframes at frame 20 and 30.
- You will notice at each keyframe, a circle appears on the path shown on the stage. Click on the keyframe 10 circle and drag it up. Do the same with keyframe 30, producing a path similar to that shown below. Rewind and play the movie.


1 [Internal] Vector Shape
Sprite 1: (545,246,613,314) Copy, 100%

Curving the Ball's Path

5.

Save the movie as ***example1.dir***.

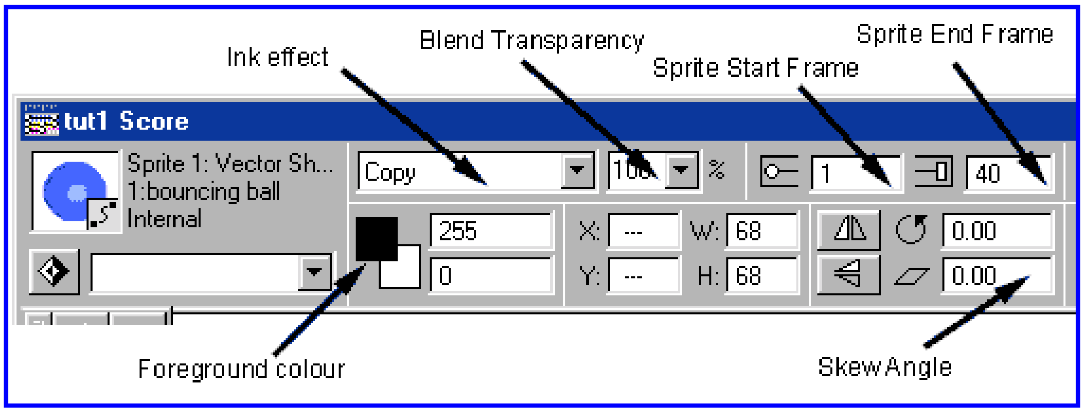
Further Animation

1. Shrinking the ball

- (Optional) Click on keyframe 40 in the score and drag it to frame 60, notice how all the keyframes spread out proportionally.
- (Optional) Click on the keyframes in the score and adjust the path if you feel like it.
- While moving the keyframes, resize the balls so they slowly get smaller. Notice while you resize the balls, the path changes and you will need to edit the path again.
- Rewind and play the movie.
- Save your movie as example2.dir

2. Animating sprite colour

- Working still with ***example1.dir***.
- Click on the keyframes in the score, and change the Foreground colour chip to different colours.
- Changing the foreground colour (refer to Fig 3.15) is like putting a coloured film over your object. The resulting colour is a mixture of the object's original colour and the 'film'. For this reason, light colours work better than dark colours for this effect..



Macromedia Director Score Property Selection

- Rewind and play the movie.
- Save as example3.dir

3. Animating sprite transparency --- Making the Ball Disappear

- Open ***example1.dir***
- Click on the keyframes in the score, and change the Blend Transparency (refer to Fig 3.15) to 100, 75, 50, 25, 0 for the consecutive keyframes.
- Rewind and play the movie.
- Save as example4.dir

4. Animating sprite shape --- Deforming The Ball

- Open ***example2.dir***
- Click on the keyframes in the score, and change the Skew Angle (refer to Fig 3.15) to 0, 20, 40, 60 and 80 for the consecutive keyframes.
- Rewind and play the movie
- Save as example5.dir

Dave Marshall
10/4/2001

Director Example 2:Importing media

To import multimedia data there are two basic ways:

- Choose File > Import ...
- Drag and drop source media into a cast member location.

The first method is useful for importing batches of data (***e.g.*** Several image sequences. The latter is clearly very intuitive.

Example: Simple Image import and Manipulation

- Drag an image into a spare cast member.
- Drag this cast member to the Score
- Manipulate as for a vector item above.
- Examples:
  - ex_dave_roll.dir

    sets up some keyframes and alters the rotation of the image
  - ex_dave_sq.dir

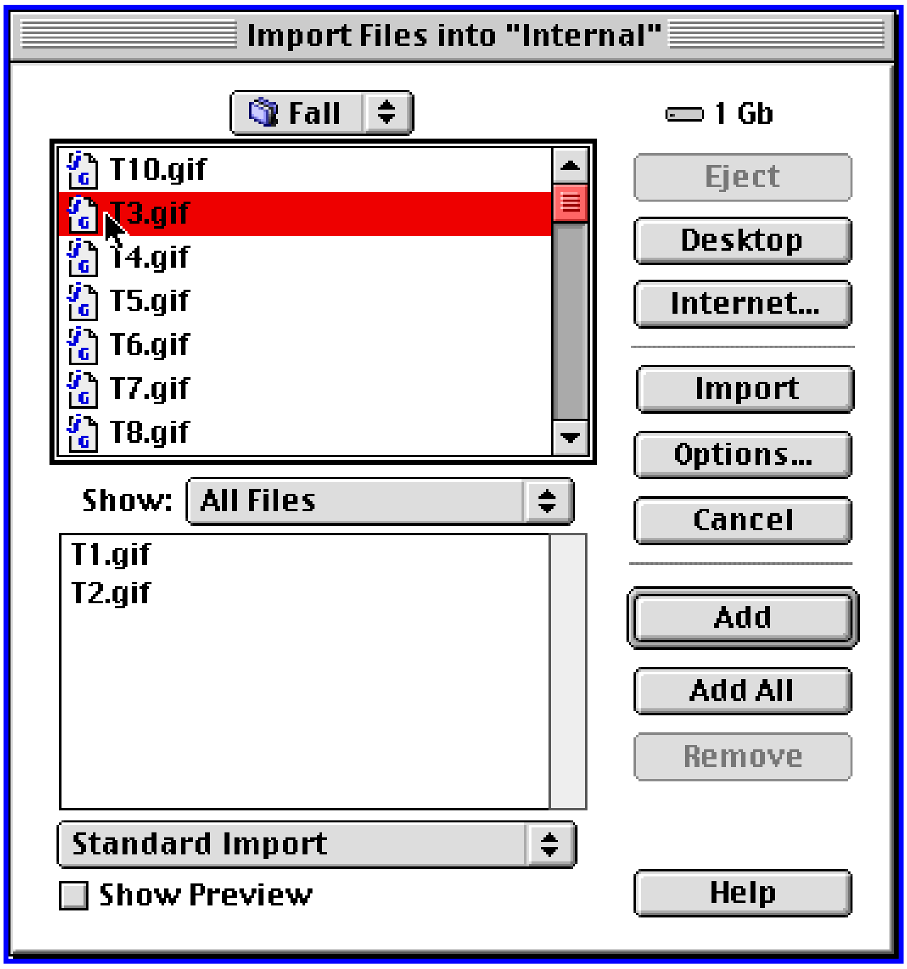
    alters the skew angle

Example: Falling Over Movie, ex_dave_movie.dir

Download This Example

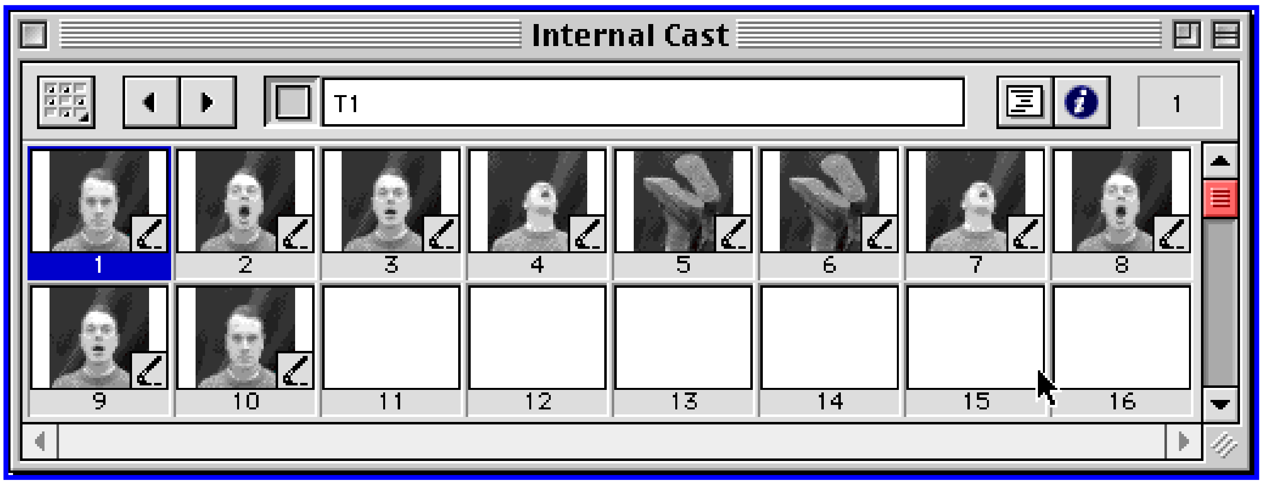
- Shockwave Movie
- Director Movie

- Several Gif images depicting sequence exist on disk.
- Choose File > Import
- Select items you wish to import by double-clicking or pressing the Add button (Fig. 3.16).



Macromedia Director Import Window

- Click on the Import Button
- Several new cast members should be added



Falling Cast Members

- Set looping on and play

4.Example: Pinching Movie Movie

To obtain other graphical effects external packages such as photoshop may be used.

- Photoshop has been used to set a pinch effect of varying degree for an image.
- Import images as above
- We now need to reverse the image set to obtain a smooth back and forth animation:
    - Select the sprite sequence in the score by clicking anywhere in the middle of the sprite sequence- press Command+C (Copy), then click on the frame just after the sprite sequence and press Command+V (Paste).
    - Click on this second sprite sequence and choose Modify > Reverse Sequence.
    - Select the 2 sprites by pressing Shift and clicking on both. Choose Modify > Join Sprites.
- Rewind and play the movie.

Dave Marshall
10/4/2001

Director Example 3:Simple Lingo Scripting

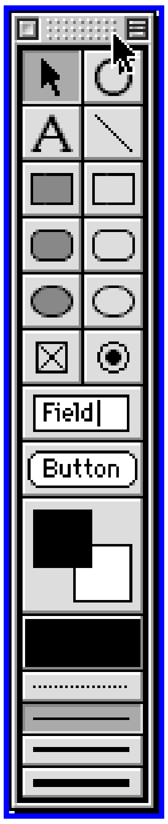
Here we illustrate the basic mechanism of scripting in Director by developing and extending a very basic example: Making a button beep and attaching a message to a button

Download This Example

- Shockwave Movie
- Director Movie

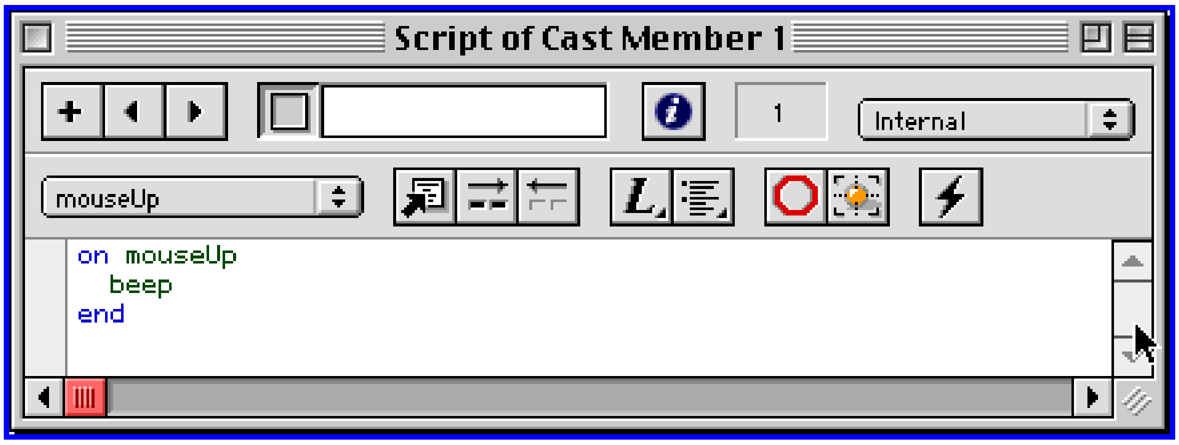
Making the a button beep

- Open a new movie.
- Turn the looping on in the control panel.
- Open the tool palette.



Macromedia Director Director Tool Palette

- Click the push button icon.
- Draw a button on the stage, and type in button (a very original name).
- Press Ctrl+click the button in the cast window and choose cast member script.



```
on mouseUp
  beep
end
```

Macromedia Director Script Cast Member Window

- Director writes the first and last line for us, add a beep command so the script look like this:

```
on mouseUp
   beep
end
```

- Close the window.
- Rewind and play the movie.
- Click the button a few times.

To pop up a message box on button press (and still beep)

Download This Example

- Shockwave Movie
- Director Movie

- Reopen the cast member script.
- Change the text so it now reads.

```
on mouseUp
  beep
  alert "Button Pressed"
end
```

- Close the window.
- Play the movie and click the button.

Dave Marshall
10/4/2001

Director Example 4:Controlling Navigation with Lingo

We begin we a preassembled Director movie:

example2.dcr

Download This Example

- Shockwave Movie
- Director Movie

- Open ***lingo_ex.3.1.dir***
- Play the movie and to see it does.

We are first going to create a ***loop the frame*** script:

- The scripting window appears. Change the text so it now reads.

  ```
  on exitFrame
     go the frame
  end
  ```

  This frame script tells Director to keep playing the same frame.
- Pressing down Alt, drag the frame script to frame 24.

Now We will create some ***markers***

As we have mentioned in above Director allows you to mark certain frames in your movie so they can be identified by buttons and links.

- Click in the marking channel in frame 1, as soon as you click a marker is created with text naming it as New Marker. Type in `scene1` over this text.
- Create markers at frame 10 and 20, naming them `scene2` and `scene3` respectively. You can delete a marker by clicking the triangle and dragging it below the marker channel.
- Create a cast member script for the next button which reads:

  ```
  on mouseUp
      go to next
  end
  ```

  The `go to next` command tells Director to go to the next consecutive marker in the score.
- Create a cast member script for the back button which reads:

  ```
  on mouseUp
      go to previous
  end
  ```

  The `go to previous` command tells Director to go to the previous marker in the score.
- Right click on the next button cast member, go to cast member properties, then check the Highlight when Clicked option. Do the same for the back button.
- Play the movie, click on the buttons and see how they work.
- Save the movie.

Now We will create some sprite scripts:

Sometimes a button will behave one way in one part of the movie and another way in a different part of the movie. That's when we use sprite scripts.

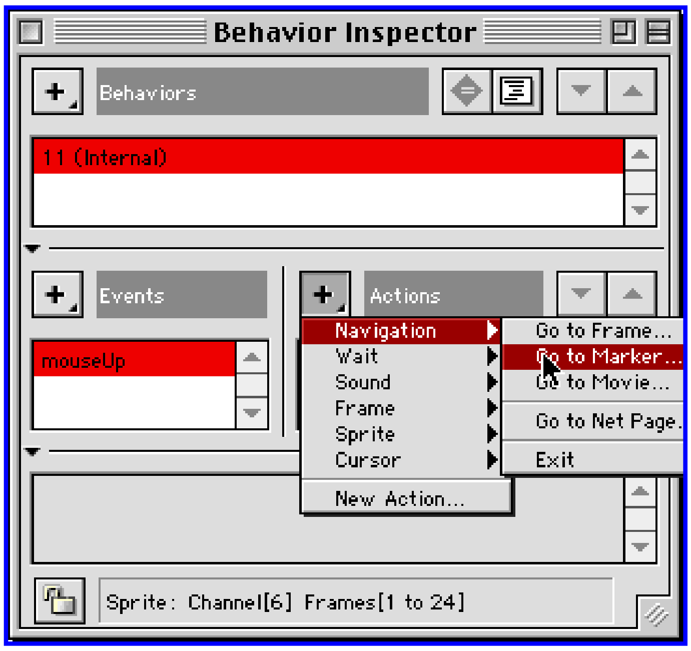
- Click on frame 10 of channel 6 (the next button) this sprite and choose Modify > Split Sprite. Do the same at frame 20.
- Select the first sprite sequence in channel 6 and select new behaviour from the behaviour pull-down in the score.
- A script window will pop up. Type `go to "scene2"` in the command area. This command tells Director to send the movie to marker `"scene2"`.

Using the ***Behaviour Inspector*** (Fig. 3.20):

Another way to write a sprite script - using the Behaviour Inspector.

- elect the second sprite sequence in channel 6.
- Open the Behaviour Inspector window - click the diamond shaped icon next to the behaviour pull-down

menu.



Macromedia Director Behaviour Inspector Window

- Expand the behaviour inspector window, so you can see events and actions. Click on the little triangle pointing to the right, when the window expands the triangle will point down.
- Click the + icon at the top left of the window and select new behaviour from the pull-down.
- Give the behaviour a name, I called mine `next2`.
- Under Events click the + icon and choose `mouseUp` from the menu.
- Under Actions click the + icon and choose Navigation > Go to marker then find scene3 on the list (Fig. 3.20)

You may realise that we now have 2 scripts attached to a single object - a cast member script as well as a sprite script. Sprite scripts take priority over cast member scripts so in our case the cast member script will be ignored.

Using Lingo `play`/`play do` to record actions:

Sometimes you may want only part an image to be linked instead of the whole object. That's when we use invisible buttons. Invisible buttons aren't magical objects, they are just shape cast members with an invisible border.

- Click on frame 1 of channel 8.
- Open the Tool palette window.
- Click on the no line button.
- Click on the rectangle button and draw a rectangle on the stage around the 1 button.
- Attach a sprite script to this shape with the command play "scene1".
- Extend the sprite sequence so it covers frame 1 to 24.
- Drag the invisible shape from the cast to the score and place it over the 2 button.
- Attach a sprite script to this sprite with the command play "scene2".
- Extend the sprite sequence so it covers frame 1 to 24.
- Repeat the steps placing the sprite over the 3 button and adding the command play "scene3".

We used a different command for the above scripts. The `play` command is similar to the `go` to command but with play,

- Director ***records*** every time a play is initiated,
- keeping track of the users' path through the movie.
- You can move back on along this path by using the `play done` command.

- Select the sprite sequence in channel 5.
- Attach a sprite script reading

```
on mouseUp
   play done
end
```

- Rewind, play the movie, click all the 1, 2, 3 buttons in various orders, click the back button also and observe the
- Save your movie.

You can see the completed example in ***lingo_ex3.2.dir***

- Shockwave Movie
- Director Movie

Dave Marshall
10/4/2001

Multimedia Programming:Tagging (SMIL)

In the last chapter we looked at scripting programming paradigms for multimedia.

In this chapter we will study ***Tagging***

- We will overview ***SMIL*** an extension of XML for synchronised media integration.

Dave Marshall
10/4/2001

What it is SMIL?

Synchronized Multimedia Integration Language (SMIL) is to synchronized multimedia what HTML is to hyperlinked text. Pronounced smile, SMIL is a simple, vendor-neutral markup language designed to let Web builders of all skill levels schedule audio, video, text, and graphics files across a timeline without having to master development tools or complex programming languages.

Whilst the SMIL language is a powerful tool for creating synchronized multimedia presentations on the web over low bandwidth connections. It is mainly meant to work with linear presentations where several types of media can be synchronized to one timeline. It does not work well with non-linear presentations and its ability to skip around in the timeline is buggy at best. However, for slideshow style mixed media presentations it the best the web has to offer.

For instance, with just a text editor and a few lines of HTML-like tags, SMIL lets Web builders specify such actions as

- ***play audio file A five seconds after video file B starts and then show image file C***.

SMIL marks a significant step toward making it easy to create low-bandwidth, TV-like content on the Web. It offers a new level of control over synchronized multimedia by allowing individual components of a presentation to be choreographed across a timeline in relation to each other. It also lets you control the layout, appearance, and exit time of each file.

What makes SMIL different from other multimedia presentation tools is that instead of forcing each component into a single video file, the text-based SMIL file merely references each file by its URL. Since the media files exist outside of the SMIL file, they retain their individual file sizes; there's no file-size bloat to slow download times.

SMIL's text-based format also makes multimedia presentations easy to edit. If you want to change when an audio component within a complex presentation begins, you can just edit the SMIL file. You don't have to rebuild the entire presentation from scratch.

As an application of XML, SMIL supports hyperlinks, which makes it the first Web-specific multimedia language to offer true interactivity.

- SMIL support

Next Up Previous

Next: SMIL support **Up:** Multimedia Programming:Tagging (SMIL) **Previous:** Multimedia Programming:Tagging (SMIL)

Dave Marshall
10/4/2001

SMIL support

The W3C recommended SMIL in June 1998, but that doesn't mean it will be universally supported across the Web. Quicktime 4.0 supports SMIL (1999)

All emerging standards have opponents, and SMIL is no exception.

No Web browser currently supports SMIL, and Microsoft and Netscape have no plans to support the standard anytime soon. In fact, even though Microsoft helped develop the SMIL 1.0 specification, Microsoft now says SMIL is not mature enough to support, claiming it overlaps with several other technologies, such as Dynamic HTML and the Document Object Model. (Microsoft's streaming media player NetShow does not support SMIL either).

Probably the real reason Microsoft stopped SMILing was the fact that Microsoft, along with Macromedia and Compaq, submitted (March 1998) its own synchronized multimedia proposal to the W3C, called HTML+TIME (Timed Interactive Multimedia Extensions for HTML). This proposal claims to address SMIL's limitations:

- SMIL is a data interchange format for media authoring tools and players
- SMIL **does not include** a means to apply the ideas to HTML and Web browsers.

However, SMIL does have an active following on the Web. RealNetworks' RealPlayer, the most common streaming media player on the Web with 85 percent of the market, supports SMIL in its G2 player.

Major media companies such as CNN Interactive, Fox News, the History Channel, and dozens of others have already begun to offer SMIL-based content via its RealPlayer G2.

Many other SMIL-compliant players, authoring tools, and servers are also becoming available.

Dave Marshall
10/4/2001

Running SMIL Applications

For this course there are basically three ways to run SMIL applications (two use the a Java Applet) so there are basically two SMIL supported mediums:

Quicktime
-- supported since Quicktime Version 4.0.

RealPlayer G2
-- integrated SMIL support

Web Browser
-- use the SOJA SMIL applet viewer with html wrapper

Applet Viewer
-- use the SOJA SMIL applet viewer with html wrapper

You will need to use both as RealPlayer and SOJA support different media (see below and Section 4.4.3).

Dave Marshall
10/4/2001

Using Quicktime

Load the SMIL file into a Quicktime plug-in (confgure Browser helper app or mime type) or the Quicktime movie player.

Dave Marshall
10/4/2001

Using RealPlayer G2

The RealPlayer G2 is installed on the applications HD in the ***RealPlayer*** folder.

Real player supports lots of file format and can use plugins. The main supported formats are:

- Real formats: RealText, RealAudio, etc...
- Images: GIF, JPEG
- Audio: AU, WAV, MIDI, etc...

To run SMIL files

Real Player uses streaming to render presentations.

The player works better when calling a SMIL file given by a Real Server, rather than from an HTTP one.

You can also open a local SMIL file or drag a SMIL file onto the RealPlayer G2 Application

---

Dave Marshall
10/4/2001

Using the SOJA applet

SOJA stands for SMIL Output in Java Applet. It was written by HELIO in 1998.

HELIO is a French association based in Melun, France. The player is free and can be downloaded from

Helio Web Site

SOJA is an applet that render SMIL in a web page or in a separate window. It supports the following formats:

- Images: GIF, JPEG
- Audio: AU and AUZ (AU zipped) -- SUN Audio files
- Text: plain text

SOJA ***does not*** use streaming. HELIO chose to store media before rendering them. You have to wait until each of them has properly been loaded but the presentation will never be stopped.

Running SOJA

To run SMIL through an applet you have to

- call the applet from an HTML file:

  ```
  <APPLET CODE="org.helio.soja.SojaApplet.class"
          ARCHIVE="soja.jar" CODEBASE="../"
          WIDTH="600" HEIGHT="300">
      <PARAM NAME="source" VALUE="cardiff_eg.smil">
      <PARAM NAME="bgcolor" VALUE="#000066">
     </APPLET>
  ```

  .
- the SOJA (`soja.jar`) archive is located in the SMIL folder on the Macintoshes.
- You may need to alter the `CODEBASE` attribute for your own applications
- The `PARAM NAME="source" VALUE="MY_SMILFILE.smil"` is how the

file is called.

There are plenty of HTML SMIL/applet wrapper example files in the SMIL demos folder.

RUNNING APPLETS

This should be easy to do

- Run the html file through a java enabled browser
- Use Apple Applet Runner
  - uses MAC OS Runtime Java (Java 1.2)
  - less fat for SMIL applications (we do really need Web connection for our examples)
  - Efficient JAVA and MAC OS run.
  - Located in ***Apple Extras:Mac OS Runtime For Java*** folder
  - ***TO RUN***: Drag files on to application, **OR**
  - ***TO RUN***: Open file from within application

Dave Marshall
10/4/2001

another SMILE viewer:GRINS

This viewer is NOT available for this course but you may wish to investigate it.

GRINS stands for Graphical Interface for SMIL. It can be found at

CWI Web Site

Dave Marshall
10/4/2001

Let us begin to SMIL -- SMIL Authoring

The notes here are essentially summaries of the SMIL Specification and tutorials available at the WWW3 consortium web site:

- http://WDVL.com/Authoring/Languages/XML/SMIL/Intro/smil.html
- http://www.w3.org/TR/REC-smil

A SMIL document is a special kind of XML 1.0 document:

Extensible Markup Language (XML) is a human-readable, machine-understandable, general syntax for describing hierarchical data, applicable to a wide range of applications (databases, e-commerce, Java, web development, searching, etc.). XML is an ISO compliant subset of SGML (Standard Generalized Markup Language). XML is extensible because it is a metalanguage, which enables someone to write a Document Type Definition (DTD) like HTML 4.0 and define the rules of the language so the document can be interpreted by the document receiver. The purpose of XML is to provide an easy to use subset of SGML that allows for custom tags to be processed. Custom tags will enable the definition, transmission and interpretation of data structures between organizations.

Further information on XML may be found at:

- Extensible Markup Language (XML) 1.0T. Bray, J. Paoli, C.M. Sperberg-McQueen

---

Dave Marshall
10/4/2001

SMIL Syntax Overview

SMIL files are usually named with `.smi` or `.smil` extensions

Dave Marshall
10/4/2001

Basic Layout

The basic Layout of a SMIL Documents is as follows:

```
<smil>
 <head>
  <meta name="copyright" content="Your Name" />
  <layout>
   <!-- layout tags -->
  </layout>
 </head>
 <body>
  <!-- media and synchronization tags -->
 </body>
</smil>
```

A source begins with `<smil>` and ends with `</smil>`.

Note that SMIL, like XML ***but*** unlike HTML, is ***case sensitive***

```
<smil>
[...]
</smil>
```

SMIL documents have two parts: head and body. Each of them must have `<smil>` as a parent.

```
<smil>
 <head>
  [...]
 </head>
 <body>
  [...]
 </body>
</smil>
```

Some tags, such as meta can have a slash at their end:

```
[...]
```

```
 <head>
  <meta name="copyright" content="Your Name" />
 </head>
[...]
```

This is because SMIL is XML-based.

Some tags are written:

- `<tag> ... </tag>`
- `<tag />`

Dave Marshall
10/4/2001

SMIL Layout

Everything concerning layout (including window settings) is stored between the `<layout>` and the `</layout>` tags in the ***header*** as shown in the above section.

A variety of Layout Tags define the presentation layout:

```
 <smil>
 <head>
<layout>
   <!-- layout tags -->
 </layout>
......
```

- Window settings

Dave Marshall
10/4/2001

Window settings

You can set width and height for the window in which your presentation will be rendered with `<root-layout>`.

The following source will create a window with a 300x200 pixels dimension and also sets the background to be white.

```
<layout>
 <root-layout width="300" height="200"
           background-color="white" />
</layout>
```

Dave Marshall
10/4/2001

Positioning Media

It is really easy to position media with SMIL.

You can position media in 2 ways:

Absolute Positioning
: -- Media are located with offsets from the origin -- the upper left corner of the window.

Relative Positioning
: -- Media are located relative to the window's dimensions.

We define position with a `<region>` tag

Dave Marshall
10/4/2001

The Region tag

--

To insert a media within our presentation we use the `<region>` tag. we must specify the region (the place) where it will be displayed. Let's say we want to insert the `Cardiff` icon (533x250 pixels) at 30 pixels from the left border and at 25 pixels from the top border.

We must also assign an `id` that identifies the region.

The header becomes:

```
<smil>
 <head>
  <layout>
   <root-layout width="600" height="300"
        background-color="white" />
   <region id="cardiff_icon" left="30" top="25"
        width="533" height="250" />
  </layout>
 </head>
 ......
```

Dave Marshall
10/4/2001

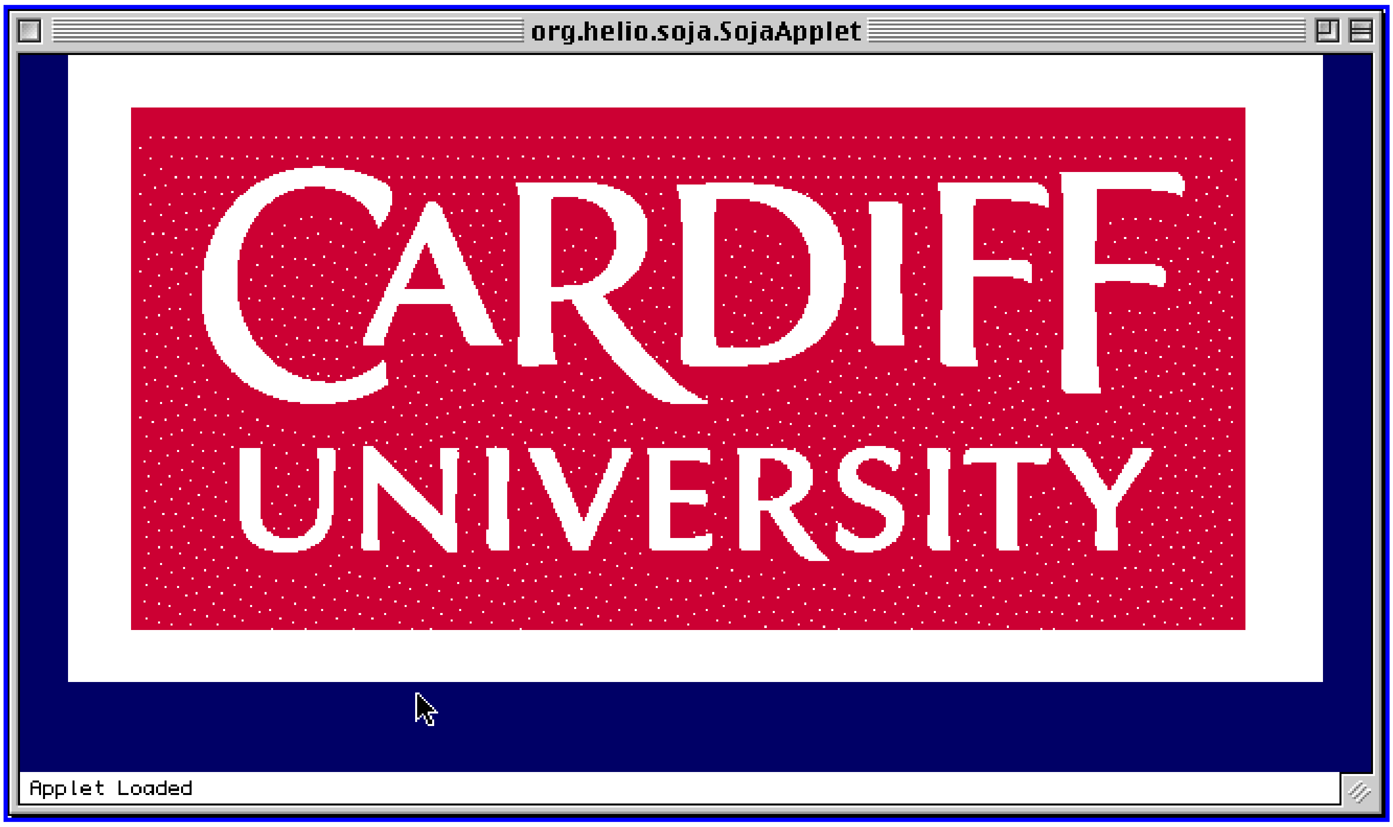
The `img` tag

To insert the Cardiff icon in the region called "cardiff_icon", we use the `<img>` tag as shown in the source below.

Note that the region attribute is a pointer to the `<region>` tag.

```
<smil>
 <head>
  <layout>
   <root-layout width="600" height="300"
        background-color="white" />
   <region id="cardiff_icon" left="30" top="25"
        width="533" height="250" />
  </layout>
 </head>
 <body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" />
 </body>
</smil>
```

This produces the following output:



Simple Cardiff Image Placement in SMIL

Dave Marshall
10/4/2001

Relative Position Example

if you wish to display the Cardiff icon at 10% from the `left` border and at 5% from the `top` border, modify the previous source and replace the left and top attributes.

```
<smil>
 <head>
  <layout>
   <root-layout width="600" height="600"
        background-color="white" />
   <region id="cardiff_icon" left="10%" top="5%"
       width="533" height="250" />
  </layout>
 </head>
 <body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" />
 </body>
</smil>
```

Dave Marshall
10/4/2001

Overlaying Regions

We have just seen how to position a media along x and y axes (left and top).

What if two regions overlap?

- Which one should be displayed on top ?

The following code points out the problem:

```
<smil>
 <head>
  <layout>
   <root-layout width="300" height="200" background-color="white" />
   <region id="region_1" left="50" top="50" width="150" height="125" />
   <region id="region_2" left="25" top="25" width="100" height="100" />
  </layout>
 </head>
 <body>
  <par>
   <text src="text1.txt" region="region_1" />
   <text src="text2.txt" region="region_2" />
  </par>
 </body>
</smil>
```

To ensure that one region is over the other, add `z-index` attribute to `<region>`.

When two region overlay:

- the one with the greater z-index is on top.
- If both regions have the same z-index, the first rendered one is below the other.

In the following code, we add `z-index` to `region_1` and `region_2`:

```
<smil>
 <head>
  <layout>
   <root-layout width="300" height="200" background-color="white" />
   <region id="region_1" left="50" top="50" width="150"
           height="125" z-index="2"/>
   <region id="region_2" left="25" top="25" width="100"
           height="100" z-index="1"/>
  </layout>
 </head>
 <body>
  <par>
   <text src="text1.txt" region="region_1" />
   <text src="text2.txt" region="region_2" />
  </par>
 </body>
</smil>
```

Dave Marshall
10/4/2001

Support for Other Media

We have used images and text media in previous presentations but other media can be introduced within SMIL.

Given the non-standard support for SMIL media support depends on the player implementation. The table below sum-up what can be seen and the corresponding tag:

Media	SMIL Tag	RealPlayer G2	GRiNS	Soja
GIF	img	OK	OK	OK
JPEG	img	OK	OK	OK
Wav	audio	OK	OK	-
.au Audio	audio	OK	OK	OK
.auz Audio Zipped	audio	-	-	OK
MP3	audio	OK	-	-
Plain text	text	OK	OK	OK
Real text	textstream	OK	-	-
Real movie	video	OK	-	-
AVI	video	OK	OK	-
MPEG	video	OK	OK	-
MOV	video	OK	-	-

Dave Marshall
10/4/2001

fitting media to regions

You can set the `fit` attribute of the `<region>` tag to force media to be resized ***etc.***

The following values are valid for `fit`:

- `fill` -- make media grow and fill the area.
- `meet` -- make media grow (without any distortion) until it meets the region frontier.
- `slice` -- media grows (without distortion) and fill entirely its region.
- `scroll` -- if media is bigger than its region area gets scrolled.
- `hidden` -- don't show media

Obviously you set the value like this:

```
<region id="region_1"  .....
     fit="fill" />
```

Dave Marshall

Synchronisation

There are two basic ways in which we may want to play media:

- play several media one after the other,
- how to play several media in parallel.

In order to do this we need to add ***synchronisation***:

- we will need to add time parameter to media elements,

Dave Marshall
10/4/2001

Adding a duration of time to media -- dur

To add a duration of time to a media element simply specify a `dur` attribute parameter in an appropriate media tag:

```
.....
<body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" dur="6s" />
</body>
.....
```

Dave Marshall
10/4/2001

Delaying Media -- the begin tag

To specify a delay ***i.e*** when to `begin` set the `begin` attribute parameter in an appropriate media tag:

If you add begin="2s" in the cardiff image tag, you will see that the Cardiff icon will appear 2 seconds after the document began and will remain during 6 other seconds. Have a look at the source:

```
.....
<body>
 <body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" dur="6s" begin="2s" />
</body>
.....
```

Dave Marshall
10/4/2001

Sequencing Media -- the seq tag

Now that we have some basic control over individual media let's see how we play them together.

The `<seq>` tag is used to define a sequence of media.

- The media are executed one after each other:

```
.....
<seq>
   <img src="img1.gif"
        region="reg1" dur="6s" />
   <img src="img2.gif"
        region="reg2" dur="4s" begin="1s" />
</seq>
.....
```

So the setting `1s` makes the `img2.gif` icon appear 1 second after `img1.gif`.

Dave Marshall
10/4/2001

Parallel Media -- the par tag

We use the `<par>` to play media at the same time:

```
<par>
   <img src="cardiff.gif" alt="The cardiff icon"
        region="cardiff_icon" dur="6s" />
   <audio src="music.au" alt="Some Music"
         dur="6s" />
</par>
```

This will display an image and play some music along with it.

Dave Marshall
10/4/2001

Synchronisation Example 1: Planets Soundtrack

The following SMIL code plays on long soundtrack along with as series of images.

Essentially:

- The audio file and
- image sequences are played in `par`allel
- The Images are run in `sequence` with no break (`begin = 0s`)

The files are stored on the MACINTOSHES in the Multimedia Lab (in the SMIL folder) as follows:

- `planets.html` -- call smil source (below) with the SOJA applet. This demo uses zipped (SUN) audio files (`.auz`) which are not supported by RealPlayer.
- `planets.smil` -- smil source (listed below),

```
<smil>
 <head>
  <layout>
   <root-layout height="400" width="600" background-color="#000000" title="Dreaming
out Loud"/>
   <region id="satfam" width="564" height="400" top="0" left="0" background-
color="#000000" z-index="2" />
   <region id="jupfam" width="349" height="400" top="0" left="251" background-
color="#000000" z-index="2" />
   <region id="redsun" width="400" height="400" top="0" left="100" background-
color="#000000" z-index="2" />
   <region id="ngc3918" width="484" height="400" top="0" left="58" background-
color="#000000" z-index="2" />
   <region id="lagoon1" width="394" height="396" top="2" left="103" background-
color="#000000" z-index="2" />
   <region id="lagoon2" width="436" height="308" top="46" left="82" background-
color="#000000" z-index="2" />
   <region id="m33" width="371" height="400" top="0" left="114" background-
color="#000000" z-index="2" />
   <region id="orion" width="371" height="400" top="0" left="114" background-
color="#000000" z-index="2" />
   <region id="hubble5" width="455" height="400" top="0" left="72" background-
color="#000000" z-index="2" />

   <region id="pillars" width="409" height="400" top="0" left="0" background-
color="#000000" z-index="2" />

   <region id="blank" width="191" height="400" top="0" left="409" background-
color="#ffffff" z-index="2" />
   <region id="music" width="100" height="25" top="30" left="453" background-
color="#ffffff" z-index="3" />
   <region id="dreamland" width="150" height="25" top="55" left="453" background-
color="#ffffff" z-index="3"/>
   <region id="by1" width="100" height="25" top="80" left="453" background-
color="#ffffff" z-index="3" />
   <region id="don" width="100" height="25" top="105" left="453" background-
color="#ffffff" z-index="3" />

   <region id="images" width="100" height="25" top="140" left="453" background-
color="#ffffff" z-index="3" />
```

```
   <region id="nasa" width="100" height="25" top="165" left="453" background-
color="#ffffff" z-index="3" />

   <region id="smil" width="100" height="25" top="200" left="453" background-
color="#ffffff" z-index="3" />
   <region id="by2" width="100" height="25" top="225" left="453" background-
color="#ffffff" z-index="3" />
   <region id="me" width="100" height="25" top="250" left="453" background-
color="#ffffff" z-index="3" />
   <region id="jose" width="100" height="25" top="250" left="453" background-
color="#ffffff" z-index="3" />

   <region id="title" width="125" height="25" top="40" left="237" background-
color="#ffffff" z-index="2" />
   </layout>
  </head>

  <body>
   <par>
    <audio src="media/dreamworldb.auz" dur="61.90s" begin="3.00s" system-
bitrate="14000" />
     <seq>
      <img src="media/satfam1a.jpg" region="satfam" begin="1.00s" dur="4.50s" />
      <img src="media/jupfam1a.jpg" region="jupfam" begin="1.50s" dur="4.50s" />
      <img src="media/redsun.jpg" region="redsun" begin="1.00s" dur="4.50s" />
      <img src="media/ngc3918a.jpg" region="ngc3918" begin="1.00s" dur="4.50s" />
      <img src="media/lagoon1c.jpg" region="lagoon1" begin="1.00s" dur="4.50s" />
      <img src="media/lagoon2b.jpg" region="lagoon2" begin="1.00s" dur="4.50s" />
      <img src="media/m33c.jpg" region="m33" begin="1.00s" dur="4.50s" />
      <img src="media/hubble5a.jpg" region="hubble5" begin="1.00s" dur="4.50s" />
      <img src="media/orion.jpg" region="orion" begin="1.00s" dur="4.50s" />
       <par>
        <img src="media/pillarsb.jpg" region="pillars" begin="1.00s" end="50s" />

                <img src="media/blank.gif" region="blank" begin="2.00s" end="50.00s"
/>
        <text src="media/music.txt" region="music" begin="3.00s" end="50.00s" />
        <text src="media/dreamland.txt" region="dreamland" begin="4.00s" end="50.00s"
/>
        <text src="media/by.txt" region="by1" begin="7.00s" end="50.00s" />
        <text src="media/don.txt" region="don" begin="8.00s" end="50.00s" />

        <text src="media/images.txt" region="images" begin="14.00s" end="50.00s" />
        <text src="media/nasa.txt" region="nasa" begin="15.00s" end="50.00s" />

        <text src="media/smil.txt" region="smil" begin="18.00s" end="50.00s" />
        <text src="media/by.txt" region="by2" begin="19.00s" end="50.00s" />
        <text src="media/me.txt" region="me" begin="20.00s" dur="3.00s" />
        <text src="media/jose.txt" region="jose" begin="23.00s" end="50.00s" />
       </par>
      <text src="media/title.txt" region="title" begin="3.00s" end="25.00s" />
     </seq>
    </par>
   </body>
 </smil>
```

Dave Marshall
10/4/2001

Synchronisation Example 2: Slides 'N' Sound

Dr John Rosbottom of Plymouth Univ has come up with a novel way of giving lectures.

This has

- one long sequence of
- parallel pairs of images and audio files

Try out the online version:

http://www.dis.port.ac.uk/~johnr/lal2/start.htm

You may examine the files on the Macintoshes in the Multimedia lab (in SMIL folder)

- `slides_n_sound.smil` -- smil source (listed below), play with RealPlayer G2. **NOTE:** This demo uses real audio files which are not supported by SOJA:

```
<smil>
<head>
<layout>
<root-layout height="400" width="600" background-color="#000000" title="Slides and
Sound"/>
</layout>
</head>
  <body>



  <seq>
    <par>
        <audio src="audio/leconlec.rm" dur="24s" title="slide 1"/>
        <img src="slides/img001.GIF" dur="24s"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="24s" clip-end="51s" dur="27s"
title="slide 2"/>
        <img src="slides/img002.GIF" dur="27s"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="51s" clip-end="67s" dur="16s"
title="slide 3"/>
        <img src="slides/img003.GIF" dur="16s"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="67s" clip-end="116s" dur="49s"
title="slide 4"/>
        <img src="slides/img004.GIF" dur="49s"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="116s" clip-end="186s" dur="70s"
```

```
title="slide 5"/>
        <img src="slides/img005.GIF" clip-begin="116s" clip-end="186s" dur="70s"
title="slide 5"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="186s" clip-end="290s" dur="104s"
title="slide 6"/>
        <img src="slides/img006.GIF" clip-begin="186s" clip-end="290s" dur="104s"
title="slide 6"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="290s" clip-end="357s" dur="67s"
title="The Second Reason"/>
        <img src="slides/img007.GIF" clip-begin="290s" clip-end="357s" dur="67s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="357s" clip-end="373s" dur="16s"
title="The Second Reason"/>
        <img src="slides/img008.GIF" clip-begin="357s" clip-end="373s" dur="16s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="373s" clip-end="466s" dur="93s"
title="The Second Reason"/>
        <img src="slides/img009.GIF" clip-begin="373s" clip-end="466s" dur="93s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="466s" clip-end="497s" dur="31s"
title="The Second Reason"/>
        <img src="slides/img010.GIF" clip-begin="466s" clip-end="497s" dur="31s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="497s" clip-end="518s" dur="21s"
title="The Second Reason"/>
        <img src="slides/img011.GIF" clip-begin="497s" clip-end="518s" dur="21s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="518s" clip-end="536s" dur="18s"
title="The Second Reason"/>
        <img src="slides/img012.GIF" clip-begin="518s" clip-end="536s" dur="18s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="536s" clip-end="553s" dur="17s"
title="The Second Reason"/>
        <img src="slides/img013.GIF" clip-begin="536s" clip-end="553s" dur="17s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="553s" clip-end="562s" dur="9s"
title="The Second Reason"/>
        <img src="slides/img014.GIF" clip-begin="553s" clip-end="562s" dur="9s"
```

```
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="562s" clip-end="568s" dur="6s"
title="The Second Reason"/>
        <img src="slides/img015.GIF" clip-begin="562s" clip-end="568s" dur="6s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="568s" clip-end="578s" dur="10s"
title="The Second Reason"/>
        <img src="slides/img016.GIF" clip-begin="568s" clip-end="578s" dur="10s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="578s" clip-end="610s" dur="32s"
title="The Second Reason"/>
        <img src="slides/img017.GIF" clip-begin="578s" clip-end="610s" dur="32s"
title="The Second Reason"/>
    </par>

    <par>
        <audio src="audio/leconlec.rm" clip-begin="610s" clip-end="634s" dur="24s"
title="The Second Reason"/>
        <img src="slides/img018.GIF" clip-begin="610s" clip-end="634s" dur="24s"
title="The Second Reason"/>
    </par>

   <par>
        <audio src="audio/leconlec.rm" clip-begin="634s" clip-end="673s" dur="39s"
title="Slide 19"/>
        <img src="slides/img019.GIF" clip-begin="634s" clip-end="673s" dur="39s"
title="Slide 19"/>
    </par>

    <img src="slides/img006.GIF" fill="freeze" title="And finally..."
      author="Abbas Mavani (dis80047@port.ac.uk)"
     copyright="Everything is so copyright protected (c)1999"/>

    <!-- kept this in to remind me that you can have single things
    <audio src="audio/AbbasTest.rm" dur="50.5s"/>
    -->
  </seq>


  </body>
</smil>
```

Dave Marshall
10/4/2001

SMIL Events

Smiles supports event based synchronisation:

Dave Marshall
10/4/2001

begin events

When a media begins, it sends a `begin` event. If another media waits for this event, it catches it. To make a media wait to an event, one of its synchronisation attributes (begin or end) should be written as follows:

```
<!-- if you want tag to start when
     another tag begins -->
<tag begin="id(specifiedId)(begin)" />

<!-- if you want tag to start 3s after
     another tag begins -->
<tag begin="id(specifiedId)(3s)" />

<!-- if you want tag to start when
     another tag ends -->
<tag begin="id(specifiedId)(end)" />
```

For example:

```
 <body>
  <par>

   <img src="cardiff.gif" region="cardiff"
        id="cf" begin="4s" />

   <img src="next.gif" region="next"
        begin="id(cf)(2s)" />

  </par>
 </body>
```

will make the `next.gif` image begin 2s after `cardiff.gif` begins.

Dave Marshall
10/4/2001

Next Up Previous
Next: SMIL ON **Up:** SMIL Events **Previous:** begin events

The `switch` Tag

The syntax for the `switch` tag is:

```
<switch>
 <!-- child1 testAttributes1 -->
 <!-- child2 testAttributes2 -->
 <!-- child3 testAttributes3 -->
</switch>
```

The rule is:

- The first of the `<switch>` tag children whose test attributes are all evaluated to TRUE is executed.
- A tag with no test attributes is evaluated to TRUE.
- See SMIL reference for list of valid test attributes

For example you may wish to provide presentations in english or welsh:

```
 <body>
  <switch>

   <!-- English only -->
   <par system-language="en">
       <img src="cardiff.gif" region="cardiff"/>
       < audio src ="english.au" />
   </par>

  <!-- Welsh only -->
   <par system-language="cy">
       <img src="caerdydd.gif" region="cardiff"/>
       <audio src ="cymraeg.au" />
   </par>
```

somewhere in code you will (or it will be set) set the `system-language`

Dave Marshall

10/4/2001

SMIL ON

Please refer to online sources and the SMIL reference handout for further SMIL features.

Some good online SMIL resources include:

- Synchronized Multimedia
- Slides and SOund
- Content Creation and Programmer's Guide
- The CWI SMIL Page
- SMIL
- Synchronized Multimedia
- Jeff Rule's Dynamic HTML and SMIL Site
- SMIL syntax validator
- Synchronized Multimedia Integration Language
- CNET Builder.com - Web Authoring - Emerging Web standards - Synchronized Multimedia Integration Language (SMIL): multimedia made easy
- SMIL Tutorial
- Just SMIL: News

Dave Marshall
10/4/2001

Multimedia Technology

Dave Marshall
10/4/2001

Multimedia Systems Technology

Dave Marshall
10/4/2001

Next Up Previous
Next: Analog and Digital Signals **Up:** Multimedia Systems Technology **Previous:** Multimedia Systems Technology

Discrete v continuous media

Multimedia systems deal with the generation, manipulation, storage, presentation, and communication of information in digital form.

The data may be in a variety of formats: text, graphics, images, audio, video.

A majority of this data is large and the different media may need synchronisation -- the data may have temporal relationships as an integral property.

Some media is time independent or ***static*** or ***discrete*** media: normal data, text, single images, graphics are examples.

Video, animation and audio are examples of ***continuous*** media.

Dave Marshall
10/4/2001

Analog and Digital Signals

We will discuss the mechanism and issues involved in transforming signals from analog-digital in Chapter 6. Here we will introduce some basic definitions before overviewing the technology required to perform such tasks.

The world we sense is full of analog signal, electrical sensors such as transducers, thermocouples, microphones convert the medium they sense into electrical signals. These are usually continuous and still analog. These analog signals must be converted or ***digitised*** into discrete digital signals that computer can readily deal with.

Special hardware devices called ***Analog-to-Digital*** converters perform this task.

For playback ***Digital-to-Analog*** must perform a converse operation.

Note that Text, Graphics and some images are generated directly by computer and ***do not*** require digitising: they are generated directly in binary format.

Handwritten text would have to digitised either by electronic pen sensing of scanning of paper based form.

Dave Marshall
10/4/2001

Input Devices and Storage

Let us now consider each media in turn and summarise how it may be input into a Multimedia system. We also briefly analyse the basic storage requirements for each type of data. We do not yet consider any effect of compression on the files. Note that storage requirements a large for many forms of media.

Dave Marshall
10/4/2001

Text and Static Data

The sources of this media are the keyboard, floppies, disks and tapes. Text files are usually stored and input character by character. Files may contain raw text or formatted text ***e.g*** HTML, Rich Text Format (RTF) or a program language source (C, Pascal, ***etc.***.

Even though to data medium does not include any temporal constraints there may be an natural implied sequence ***e.g.*** HTML format sequence, Sequence of C program statements.

The basic storage of text is 1 byte per character (text or format character). For other forms of data ***e.g.*** Spreadsheet files some formats may store format as text (with formatting) others may use binary encoding.

Even the the storage requirements of this data is never high when data is stored on disk small files may take larger disk storage requirements due to block and sector sizes of disk partitions.

Dave Marshall
10/4/2001

Graphics

Graphics are usually constructed by the composition of primitive objects such as lines, polygons, circles, curves and arcs. Graphics are usually generated by a graphics editor program (***e.g.*** Freehand) or automatically by a program (***e.g.*** Postscript usually generated this way). Graphics are usually editable or revisable (unlike Images).

Graphics input devices include: keyboard (for text and cursor control), mouse, trackball or graphics tablet.

Graphics files may adhere to a graphics standard (OpenGL, PHIGS, GKS) Text may need to stored also. Graphics files usually store the primitive assembly and do not take up a very high overhead.

Dave Marshall
10/4/2001

Images

Images are still pictures which (uncompressed) are represented as a bitmap (a grid of pixels).

Images may be generated by programs similar to graphics or animation programs. But images may be scanned for photographs or pictures using a digital scanner or from a digital camera. Some Video cameras allow for still image capture also. Analog sources will require digitising.

Images may be stored at 1 bit per pixel (Black and White), 8 Bits per pixel (Grey Scale, Colour Map) or 24 Bits per pixel (True Colour) (See Chapter 6).

Thus a 512x512 Grey scale image takes up 1/4 Mb, a 512x512 24 bit image takes 3/4 Mb with no compression. This overhead soon increases with image size so compression is commonly applied (See Chapter 7)

Dave Marshall
10/4/2001

Audio

Audio signals are continuous analog signals. They are first captured by a microphones and then digitised and store -- usually compressed as CD quality audio requires 16-bit sampling at 44.1 KHz (There are other audio sampling rates -- Chapter 6). So 1 Minute of Mono CD quality audio requires 60*44100*2 Bytes which is approximately 5 Mb.

Dave Marshall
10/4/2001

Video

Analog Video is usually captured by a video camera and then digitised. There are a variety of video (analog and digital) formats (Chapter 6)

Raw video can be regarded as being a series of single images. There are typically 25, 30 or 50 frames per second. Therefore a 512x512 size monochrome video images take 25*0.25 = 6.25Mb for a minute to store uncompressed. Digital video clearly needs to be compressed.

Dave Marshall
10/4/2001

Output Devices

The output devices for a basic multimedia system include

- A High Resolution Colour Monitor
- CD Quality Audio Output
- Colour Printer
- Video Output to save Multimedia presentations to (Analog) Video Tape, CD-ROM DVD.
- Audio Recorder (DAT, DVD, CD-ROM, (Analog) Cassette)
- Storage Medium (Hard Disk, Removable Drives, CD-ROM) -- see next section.

Dave Marshall
10/4/2001

Storage Media

Let us first recap the major problems that affect storage media:

- Large volume of date
- Real time delivery
- Data format
- Storage Medium
- Retrieval mechanisms

First two factors are the real issues that storage media have to deal and we have discussed these factors already. Due to the volume of data the Data format will include compression (see Chapters 6 and 7).

The type of storage medium and underlying retrieval mechanism will affect how the media is stored and delivered. Ultimately any system will have to deliver high performance I/O. We discuss this issue next before going on to discuss actual Multimedia storage devices.

Dave Marshall
10/4/2001

High performance I/O

There are four factors that influence I/O performance:

Data
-- Data is high volume, maybe continuous and may require contiguous storage. Direct relationship between size of data and how long it takes to handle. Compression and also distributed storage (See RAID technology (Section 5.5.3 below).

Data Storage
-- The strategy for data storage depends of the storage hardware and the nature of the data. The following storage parameters affect how data is stored:

- Storage Capacity
- Read and Write Operations of hardware
- Unit of transfer of Read and Write
- Physical organisation of storage units
- Read/Write heads, Cylinders per disk, Tracks per cylinder, Sectors per Track
- Read time
- Seek time

Data Transfer
-- Depend how data generated and written to disk, and in what sequence it needs to retrieved. Writing/Generation of Multimedia data is usually sequential ***e.g.*** streaming digital audio/video direct to disk. Individual data (***e.g.*** audio/video file) usually streamed.

RAID architecture can be employed to accomplish high I/O rates by exploiting parallel disk access (Section 5.5.3)

Operating System Support
-- Scheduling of processes when I/O is initiated. Time critical operations can adopt special procedures. Direct disk transfer operations free up CPU/Operating system space.

Dave Marshall

Basic Storage

Basic storage units have problems dealing with large multimedia data

- Single Hard Drives -- SCSI/IDE Drives. So called ***AV*** (Audio-Visual) drives, which avoid thermal recalibration between read/writes, are suitable for desktop multimedia. New drives are fast enough for direct to disk audio and video capture. But not adequate for commercial/professional Multimedia. Employed in RAID architectures (Section 5.5.3)
- Removable Media -- Jaz/Zip Drives, CD-ROM, DVD. Conventional (dying out?) floppies not adequate due 1.4 Mb capacity. Other media usually ok for backup but usually suffer from worse performance than single hard drives.

RAID -- Redundant Array of Inexpensive Disks

This concept of RAID has been developed to fulfill the needs of current multimedia and other data hungry application programs, and which require fault tolerance to be built into the storage device. Further, techniques of parallel processing are also suitable to exploiting the benefits of such an arrangement of hard disks.

Raid technology offers some significant advantages as a storage medium:

- Affordable alternative to mass storage
- High throughput and reliability

The cost per megabyte of a disk has been constantly dropping, with smaller drives playing a larger role in this improvement. Although larger disks can store more data, it is generally more power effective to use small diameter disks (as less power consumption is needed to spin the smaller disks). Also, as smaller drives have fewer cylinders, seek distances are correspondingly lower. Following this general trend, a new candidate for mass storage has appeared on the market, based on the same technology as magnetic disks, but with a new organisation. These are arrays of small and inexpensive disks placed together, based on the idea that disk throughput can be increased by having many disk drives with many heads operating in parallel. The distribution of data over multiple disks automatically forces access to several disks at one time improving throughput. Disk arrays are therefore obtained by placing small disks together to obtain the performance of more expensive high end disks.

The key components of a RAID System are:

- Set of disk drives, disk arrays, viewed by user as one or more logical drives.
- Data may be distributed across drives
- Redundancy added in order to allow for disk failure

Disk arrays can be used to store large amounts of data, have high I/O rates and take less power per megabyte (when compared to high end disks) due to their size, but they have very poor reliability.

What do you think is the reason of this low reliability?

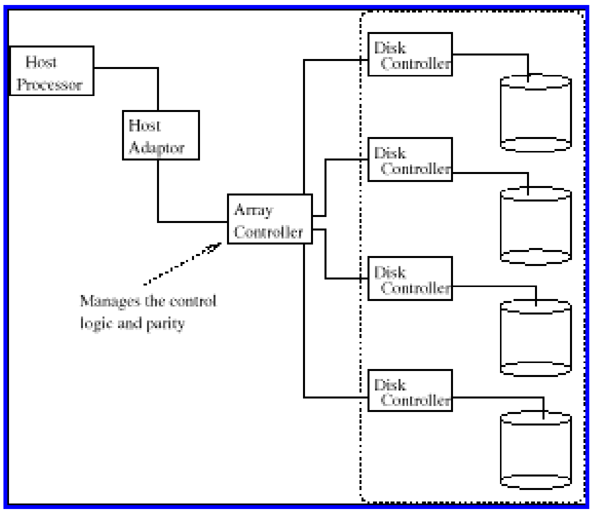
As more devices are added, reliability deteriorates (N devices generally have $\frac{1}{N}$ the reliability of a single device).

Files stored on arrays may be *striped* across multiple spindles. Since a high capacity is available due to the availability of more disks, it is possible to create redundancy within the system, so that if a disk fails the contents of a file may be reconstructed from the redundant information. This off course leads to a penalty in capacity (when storing redundant information) and in bandwidth (to update the disk). Four main techniques are available to overcome the lack of reliability of arrays:

- *Mirroring* or shadowing of the contents of disk, which can be a *capacity kill* approach to the problem. Each disk within the array is mirrored and a write operation performs a write on two disks - resulting in a 100% capacity overhead. Reads to disks may however be optimised. This solution is aimed at *high bandwidth, high availability environments*.
- *Horizontal Hamming Codes*: A special means to reconstruct information using an error correction encoding technique. This may be an overkill, as it is complex to compute over a number of disks.
- *Parity and Reed-Soloman Codes*: Also an error correction coding mechanism. Parity may be computed in a number of ways, either horizontally across disks or through the use of an interleaved parity block. Parity information also has to be stored on disk, leading to a 33% capacity cost for parity. Use of wider arrays reduces the capacity cost, but leads to a decrease

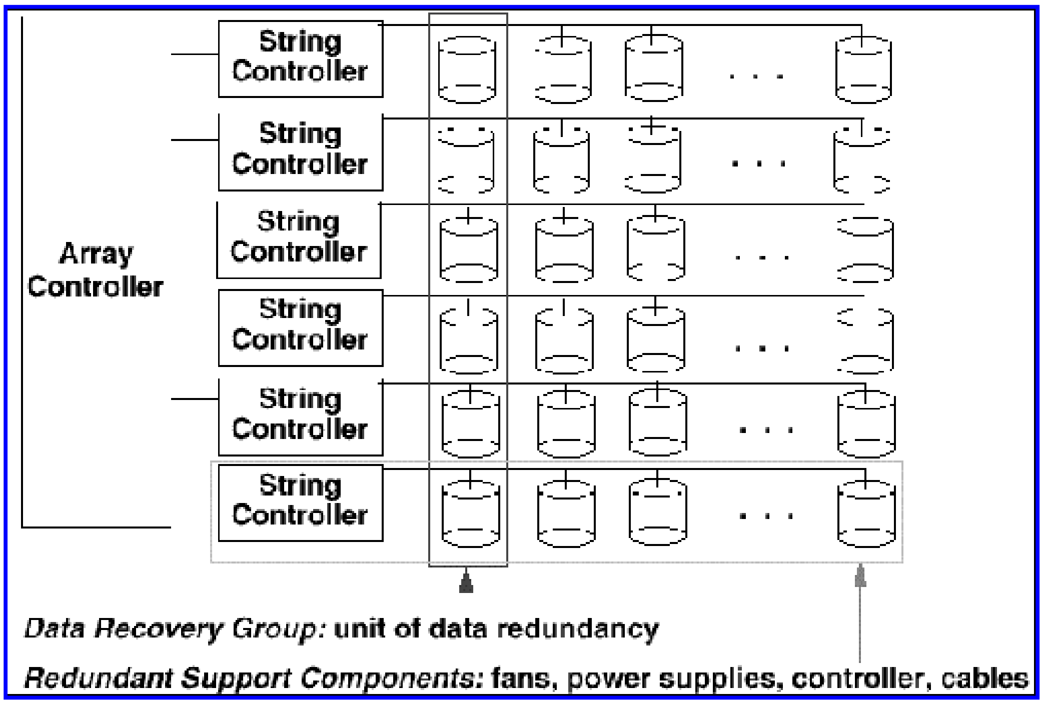
  in the expected availability and increased reconstruction times. This approach is generally aimed at high bandwidth scientific applications (such as image processing).
- *Failure Prediction*: There is no capacity overhead in this technique, though it is controversial in nature, as its use cannot be justified if all errors or failures can be forecast correctly.

Each disk within the array needs to have its own I/O controller, but interaction with a host computer may be mediated through an *array controller* as shown in figure 5.1.



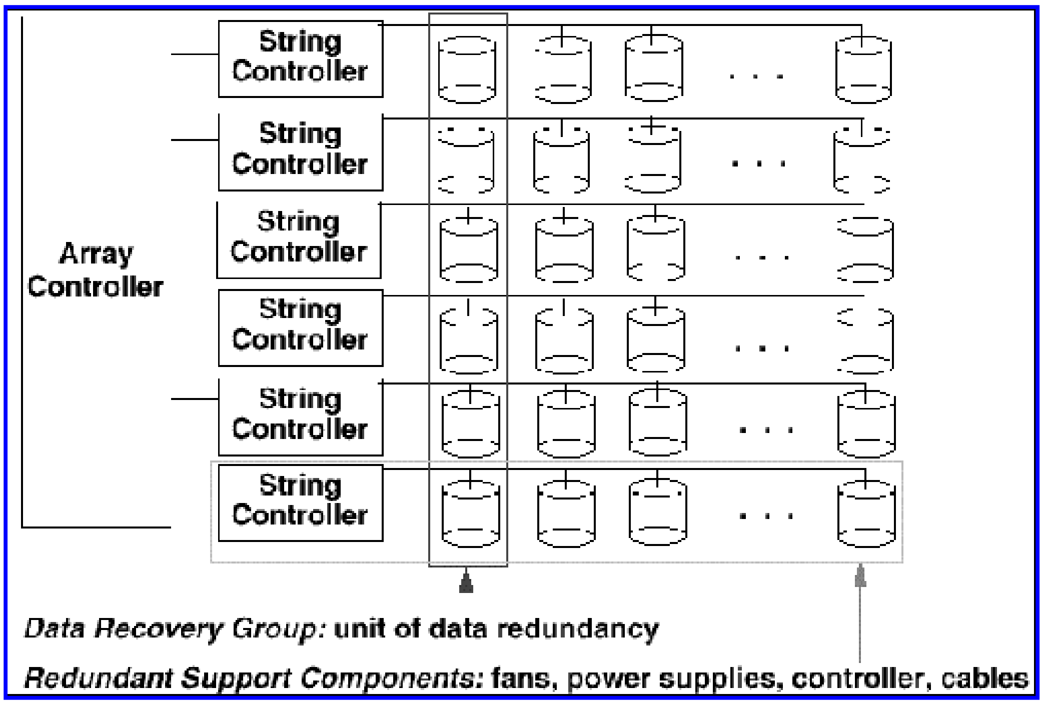
A disk array link to the host processor

It may also be possible to combine the disks together to produce a collection of devices, where each vertical array is now the unit of data redundancy. Such an arrangement is called an *orthogonal RAID* and shown in figure 5.2; other arrangements of disks are also possible.



Orthogonal RAID

Figure 5.3 identifies disk performance for a number of machines and operating systems. The Convex supercomputer seems to provide the best performance in terms of throughput (or megabytes transferred per second) based on a 32KB read operation due to the use of a RAID disk. The figure also illustrates transfer rate dependencies between hardware and the particular operating system being used.



Orthogonal RAID

There are now 8 levels of RAID technology, with each level providing a greater amount of resilience then the lower levels:

Level 0: Disk Striping

-- distributing data across multiple drives. Level 0 is an independent array without parity redundancy that accesses data across all drives in the array in a block format. To accomplish this, the first data block is read/written from/to the first disk in the array, the second block from/to the second disk, and so on. RAID 0 only addresses improved data throughput, disk capacity and disk performance. In RAID 0 data is striped across the various drives present. Although striping can improve performance on request rates, it does not provide fault tolerance. If a single disk fails, the entire system fails. This would therefore be equivalent to storing the complete set of data on a single drive, though with a lower access rate.

Level 1: Disk Mirroring

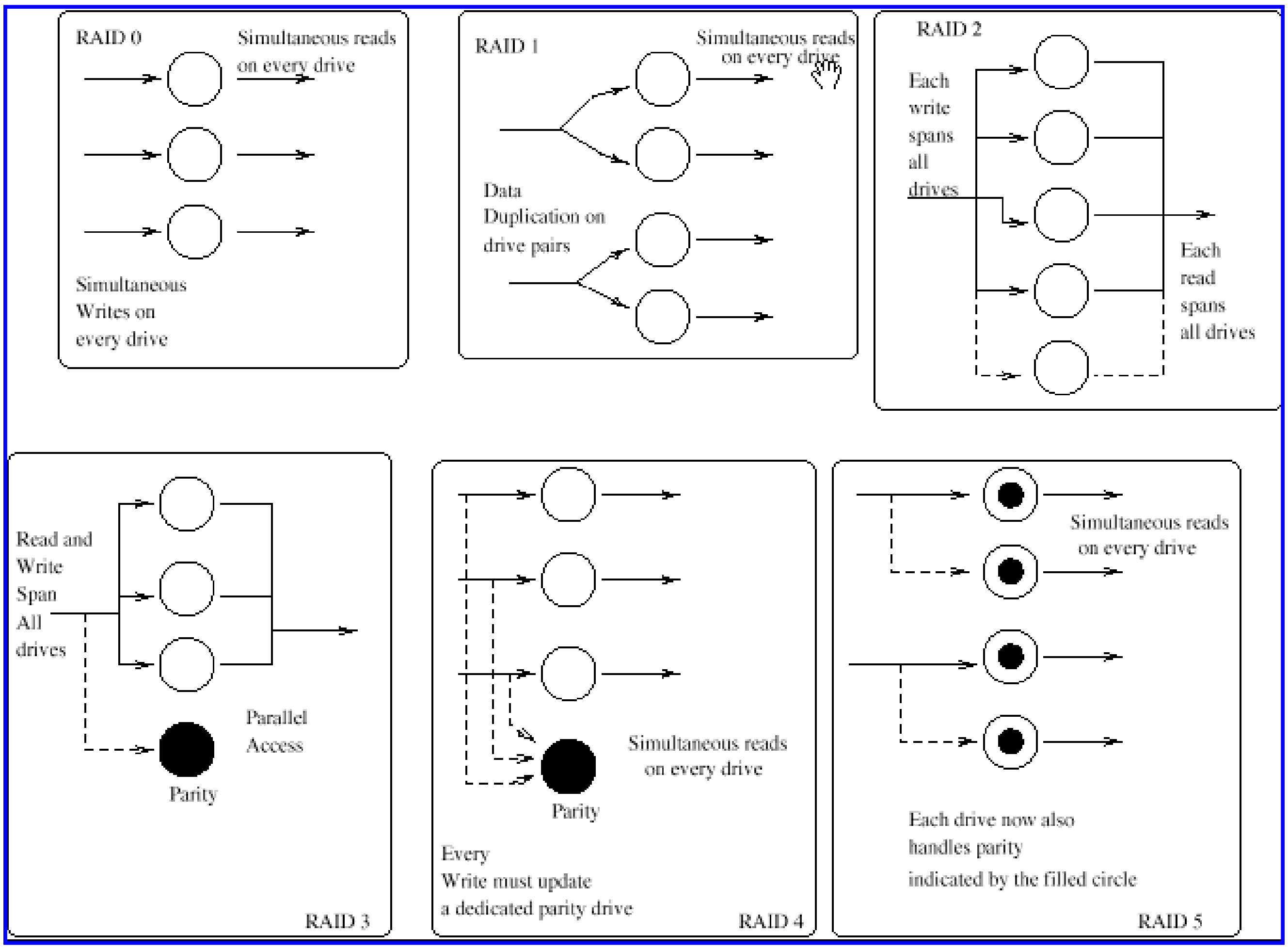
-- Level 1 focusses on ***fault tolerancing*** and involves a second duplicate write to a mirror

disk each time a write request is made. The write is performed automatically and is transparent to the user, application and system. The mirror disk contains an exact replica of the data on the *actual disk*. Data is recoverable if a drive fails and may be recoverable if both drives fail. The biggest disadvantage is that only half of the disk capacity is available for storage. Also, capacity can only be expanded in pairs of drives.

Of the RAID levels, level 1 provides the highest data availability since two complete copies of all information are maintained. In addition, read performance may be enhanced if the array controller allows simultaneous reads from both members of a mirrored pair. During writes, there will be a minor performance penalty when compared to writing to a single disk. Higher availability will be achieved if both disks in a mirror pair are on separate I/O busses.

Level 2: Bit Interleaving and HEC Parity

-- Level 2 stripes data to a group of disks using a bite stripe. A Hamming code symbol for each data stripe is stored on the check disk. This code can correct as well as detect data errors and permits data recovery without complete duplication of data. This RAID level is also sometimes referred to as Level 0+1. It combines the benefits of both striping and Level 1 - with both high availability and high performance. It can be tuned for either a request rate intensive or transfer rate intensive environment. Level 2 arrays *sector-stripe* data across groups of drives, with some drives being dedicated to storing Error Checking and Correction (ECC) information within each sector. However, since most disk drives today embed ECC information within each sector as standard, Level 2 offers no significant advantages over Level 3 architecture. At the present time there are no manufacturers of Level 2 arrays.


RAID 0
Simultaneous reads on every drive
Simultaneous Writes on every drive
RAID 1
Simultaneous reads on every drive
Data Duplication on drive pairs
RAID 2
Each write spans all drives
Each read spans all drives
Read and Write Span All drives
Parallel Access
Parity
RAID 3
Simultaneous reads on every drive
Parity
Every Write must update a dedicated parity drive
RAID 4
Simultaneous reads on every drive
Each drive now also handles parity indicated by the filled circle
RAID 5

Orthogonal RAID

Level 3: Bit Interleaving with XOR Parity

-- Level 3 is a striped parallel array where data is distributed by bit or byte. One drive in the array provides data protection by storing a parity check byte for each data stripe.

Level 3 has the advantage over lower RAID levels in that the ratio of check disk capacity to data disk capacity decreases as the number of data drives increases. It has parallel data paths and therefore offers high transfer rate performance for applications that transfer large files. Array capacity can be expanded in single drive or group increments. With Level 3, data chunks are much smaller than the average I/O size and the disk spindles are synchronised to enhance throughput in transfer rate intensive environments. Level 3 is well suited for CAD/CAM or imaging type applications.

Level 4: Block Interleaving with XOR Parity

-- In Level 4 parity is interleaved at the sector or transfer level. As with Level 3, a single drive is used to store redundant data using a parity check byte for each data stripe.

Level 4 offers high read performance and good write performance. Level 4 is a general solution, especially where the ratio of reads to writes is high. This makes Level 4 a good choice for small block transfers, which are typical for transaction processing applications. Write performance is low because the parity drive has to be written for each data write. Thus the parity drive becomes a performance bottleneck when multiple parity write I/Os are required. In this instance, Level 5 is a better solution because parity information is spiralled across all available disk drives. Level 4 systems are almost never implemented mainly because it offers no significant advantages over Level 5.

Level 5: Block Interleaving with Parity Distribution

-- Level 5 combines the throughput of block interleaved data

striping of Level 0 with the parity reconstruction mechanism of Level 3 without requiring an extra parity drive. In Level 5, both parity and data are striped across a set of disks. Data chunks are much larger than the average I/O size. Disks are able to satisfy requests independently which provides high read performance in a request-rate intensive environment. Since parity information is used, a Level 5 stripe can withstand a single disk failure without losing data or access to data.

Level 5's strength lies in handling large numbers of small files. It allows improved I/O transfer performance because the parity drive bottleneck of Level 4 is eliminated. While Level 5 is more cost effective because a separate parity drive is not used, write performance suffers. In graphic art and imaging applications, the weakness of Level 5 versus Level 3 is the write penalty from the striped parity information. In Level 3 there is no write penalty. Level 5 is usually seen in applications with large numbers of small read/write calls. Level 5 offers higher capacity utilisation when the array has less than 7 drives. With a full array, utilisation is about equal between Level 3 and 5.

Level 6: Fault Tolerant System

-- additional error recovery. This is an improvement of Level 5. Disks are considered to be in a matrix formation and parity is generated for each row and column of the matrix. Multidimensional parity is then computed and distributed among the disks in the matrix.

Level 6 became a common feature in many systems but the advent of Level 7 has led to the abandonment of Level 6 in many cases.

Level 7: Heterogeneous System

-- Fast access across whole system. Level 7 allows each individual drive to access data as fast as possible by incorporating a few crucial features:

- Each I/O drive is connected to high speed data bus which posses a central cache store capable of supporting multiple host I/O paths.
- A real time process-oriented OS is embedded into the disk array architecture **--** frees up drives, allowing independent drive head operation. Substantial improvement.
- All parity checking and check/microprocessor/bus/cache control logic embedded in this OS.
- OS designed to support multiple host interfaces **--** other RAID levels support only one.
- Additional ability to reconstruct data in the event of dependent drive failure increased due to separate cache/device control, and secondary, tertiary and beyond parity calculation **--** up to four simultaneous disk failures supported.
- ***Dynamic Mapping*** used. In conventional storage a block of data, once created, is written to fixed memory location. All operations then rewrite data back to this location. In Dynamic Memory this constraint is freed and new write locations logged and mapped. This frees additional disk accesses and the potential for a bottleneck.

These First 6 RAID levels are illustrated in Figure 5.4, where each circle represents a single disk drive, and arrows represent data flows.

Optical Storage

Optical storage has been the most popular storage medium in the multimedia context due its compact size, high density recording, easy handling and low cost per MB.

CD is the most common and we discuss this below. Laser disc and recently DVD are also popular.

CD Storage

There a now various formats of CD:

In the beginning, there was CD-DA (Compact Disc-Digital Audio), or standard music CDs. CD-DA moved onto CD-ROM when people realized that you could store a whole bunch of computer data on a 12cm optical disk (650mb). CD-ROM drives are simply another kind of digital storage media for computers, albeit read-only. They are peripherals just like hard disks and floppy drives. (Incidentally, the convention is that when referring to magnetic media, it is spelled ***disc***. Optical media like CDs, LaserDisc, and all the other formats I'm about to explain are spelled disc.)

CD-I (Compact Disc-Interactive) came next. This is a consumer electronics format that uses the optical disk in combination with a computer to provide a home entertainment system that delivers music, graphics, text, animation, and video in the living room. Unlike a CD-ROM drive, a CD-I player is a standalone system that requires no external computer. It plugs directly into a TV and stereo system and comes with a remote control to allow the user to interact with software programs sold on disks. It looks and feels much like a CD player except that you get images as well as music out of it and you can actively control what happens. In fact, it ***is*** a CD-DA player and all of your standard music CDs will play on a CD-I player; there is just no video in that case.

Next came CD-ROM/XA (eXtended Architecture). Now we go back to computer peripherals - a CD-ROM drive but with some of the compressed audio capabilities found in a CD-I player (called ADPCM). This allows interleaving of audio and other data so that an XA drive can play audio and display pictures (or other things) simultaneously. There is special hardware in an XA drive controller to handle the audio playback. This format came from a desire to inject some of the features of CD-I back into the professional market.

Now, along comes the idea from Kodak for Photo CD - digital pictures on compact disk. They teamed up with Philips to develop the standard for Photo CD disks. At this point, a new problem enters the picture, if you'll pardon the expression. All of the disk formats mentioned so far are read-only; there is no way for anyone but the producer of one of these disks to store his/her own content on the disk - that is, to write to it. But there already existed a technology called WORM (Write Once Read Many). This is an optical disk that can be written to, but exactly once. You can ***burn*** data on it, but once burned the data can not be erased, although it can then be used like a CD-ROM disk and read forever. (Depending on your definition of forever, of course.)

CD-ROM, CD-ROM/XA, and CD-I disks are normally ***mastered***, as opposed to burned. That means that one master copy is made and then hundreds, or thousands, or millions (if you're lucky enough to need that many) of replicates are pressed from the master. This process is much cheaper than burning for quantities above a few dozen or so. Generally, disk pressing plants can handle all of these formats as the underlying technology is the same; the only difference is in the data and disk format.

The reason that WORM technology was critical for Photo CD is obvious - the content of these disks is not determined by the manufacturer or publisher. For Photo CD, each disk will be different - a roll or few rolls of film per disk from a customer.

Kodak and Philips wanted Photo CD disks to be playable on both computer peripherals for desktop publishing uses AND on a consumer device for home viewing. For the former, CD-ROM/XA was chosen as a carrier and for the latter CD-I, which was already designed as a consumer electronics device, and dedicated Photo CD players. This desire for a hybrid disk, or one with multi-platform compatibility, led to the development of the CD-I Bridge disk format. A Bridge disk is one that is readable on both a CD-I player and a CD-ROM/XA drive.

This Bridge format is the reason there is so much confusion about CD-ROM drives for Photo CD. A drive that supports Photo CD must be a CD-ROM/XA drive that is also Bridge-compatible. (The technical description of Bridge disks calls for supporting certain kinds of sectors identified by ***form*** and ***mode*** bits, which is what you usually hear instead of the ***Bridge*** disk label.) That almost completes the picture, except for the concept of sessions.

Although a WORM disk can only be written to once, it is not necessary to write, or burn, the entire disk all at once. You can burn the disk initially with, say, a few hundred megabytes of data, and then go back later and burn some more data onto it. Of course, each burn must be to a virgin part of the disk; once a spot on the disk is burned, it can not be re-burned. Each burn operation is referred to as a ***session***, and a drive or disk that supports this multiple burning operation is called ***multisession***.

Originally, all WORMs were single session only. That is, you could not go back and add data to a WORM disk once it was burned, even if it was not full. For Photo CD, they wanted the consumer to be able to add more pictures to an existing disk as additional rolls of film were processed. So the extension of WORM technology to multisession was developed and adopted for the Bridge disk format. This required hardware changes to CD-ROM/XA drives and that is why there are a fair number of single session XA drives on the market and multisession ones appearing more and more.

A single session drive can read a multisession disk, but it can only read the contents of the first session that was burned. Incidentally, all Philips CD-I players are multisession, although all current CD-I disks have only a single session on them.

The capacity of a CD-ROM is 620-700 Mbs depending on CD material, Drives that read and write the CD-ROMS. 650 Mb (74 Mins) is a typical write once CD-ROM size.

- CD Standards

CD Standards

There are several CD standard for different types of media:

Red Book
    -- Digital Audio: Most Music CDs.
Yellow Book
    -- CD-ROM: Model 1 - computer data, Model 2 - compress audio/video data.
Green Book
    -- CD-I
Orange Book
    -- write once CDs
Blue Book
    -- LaserDisc

DVD

DVD, which stands for ***Digital Video Disc***, ***Digital Versatile Disc***, or nothing, depending on whom you ask, is the next generation of optical disc storage technology.

DVD has become a major new medium for a whole host of multimedia system:

It's essentially a bigger, faster CD that can hold video as well as audio and computer data. DVD aims to encompass home entertainment, computers, and business information with a single digital format, eventually replacing audio CD, videotape, laserdisc, CD-ROM, and perhaps even video game cartridges. DVD has widespread support from all major electronics companies, all major computer hardware companies, and most major movie and music studios, which is unprecedented and says much for its chances of success (or, pessimistically, the likelihood of it being forced down our throats).

It's important to understand the difference between DVD-Video and DVD-ROM. DVD-Video (often simply called DVD) holds video programs and is played in a DVD player hooked up to a TV. DVD-ROM holds computer data and is read by a DVD-ROM drive hooked up to a computer. The difference is similar to that between Audio CD and CD-ROM. DVD-ROM also includes future variations that are recordable one time (DVD-R) or many times (DVD-RAM). Most people expect DVD-ROM to be initially much more successful than DVD-Video. Most new computers with DVD-ROM drives will also be able to play DVD-Videos.

There's also a DVD-Audio format. The technical specifications for DVD-Audio are not yet determined.

- Interactive DVD features

What are the features of DVD-Video?

The main features of DVD include:

- Over 2 hours of high-quality digital video (over 8 on a double-sided, dual-layer disc).
- Support for widescreen movies on standard or widescreen TVs (4:3 and 16:9 aspect ratios).
- Up to 8 tracks of digital audio (for multiple languages), each with as many as 8 channels.
- Up to 32 subtitle/karaoke tracks.
- Automatic ***seamless*** branching of video (for multiple story lines or ratings on one disc).
- Up to 9 camera angles (different viewpoints can be selected during playback).
- Menus and simple interactive features (for games, quizzes, etc.).
- Multilingual identifying text for title name, album name, song name, cast, crew, etc.
- ***Instant*** rewind and fast forward, including search to title, chapter, track, and timecode.
- Durability (no wear from playing, only from physical damage).
- Not susceptible to magnetic fields. Resistant to heat.
- Compact size (easy to handle and store, players can be portable, replication is cheaper).

Note: Most discs do not contain all features (multiple audio/subtitle tracks, seamless branching, parental control, etc.). Some discs may not allow searching or skipping.

Most players support a standard set of features:

- Language choice (for automatic selection of video scenes, audio tracks, subtitle tracks, and menus) which must be supported by additional content on the disc. Some players include additional features:
- Special effects playback: freeze, step, slow, fast, and scan (no reverse play or reverse step).
- Parental lock (for denying playback of discs or scenes with objectionable material) which again must be supported by additional content on the disc. Some players include additional features
- Programmability (playback of selected sections in a desired sequence).
- Random play and repeat play.
- Digital audio output (PCM stereo and Dolby-Digital).

- Compatibility with audio CDs.
- Component (YUV or RGB) output for highest-quality picture.
- Compatibility with Video CDs.
- Six-channel analog output from internal audio decoder.
- Compatibility with laserdiscs and CDVs.
- Reverse single frame stepping.
- RF output (for TVs with no direct video input).
- Multilingual on-screen display.

Quality of DVD-Video

DVD has the capability to produce near-studio-quality video and better-than-CD-quality audio. DVD is vastly superior to videotape and generally better than laserdisc. However, quality depends on many production factors. Until compression experience and technology improves we will occasionally see DVDs that are inferior to laserdiscs. Also, since large amounts of video have already been encoded for Video CD using MPEG-1, a few low-budget DVDs will use that format (which is no better than VHS) instead of higher-quality MPEG-2 (See Chapters 6 and 7)

DVD video is compressed from digital studio master tapes to MPEG-2 format. This is a ***lossy*** compression (see Chapter 7) which removes redundant information (such as sections of the picture that don't change) and information that's not readily perceptible by the human eye. The resulting video, especially when it is complex or changing quickly, may sometimes contain ***artifacts*** such as blockiness, fuzziness, and video noise depending on the processing quality and amount of compression (Chapter 7). At average rates of 3.5 Mbps (million bits/second), artifacts may be occasionally noticeable. Higher data rates result in higher quality, with almost no perceptible difference from the original master at rates above 6 Mbps. As MPEG compression technology improves, better quality is being achieved at lower rates.

Some DVD demos have visible artifacts such as color banding, blurriness, shimmering, missing detail, and even effects such as a face which ***floats*** behind the rest of the moving picture. This is sometimes caused by poor MPEG encoding, but is just as often caused by a poorly adjusted TV or by sloppy digital noise reduction prior to encoding. The Free Willy and Twister excerpts on the Panasonic demo disc are good examples of this. In any case, bad demos are not an indication that DVD quality is bad, since other demos show no artifacts or other problems. Bad demos are simply an indication of how bad DVD can be if not properly processed and correctly reproduced. Early demos were shown on prototype players based on prerelease hardware and firmware. Many demo discs were rushed through the encoding process in order to be distributed as quickly as possible. Contrary to popular opinion, and as stupid as it may seem, these demos are not carefully ***tweaked*** to show DVD at its best. Also, most salespeople are incapable of properly adjusting a television set. Most TVs have the sharpness set too high for the clarity of DVD. This exaggerates high-frequency video and causes distortion, just as the treble control set too high for a CD causes it to sound harsh. DVD video has exceptional color fidelity, so muddy or washed-out colors are almost always a problem in the display, not in the DVD player or disc.

DVD audio quality is excellent. One of DVD's audio formats is LPCM (linear pulse code modulation) with sampling sizes and rates higher than audio CD. Alternately, audio for most movies is stored as discrete multi-channel surround sound using Dolby Digital or MPEG-2 audio compression similar to the surround sound formats used in theaters. As with video, audio quality depends on how well the encoding was done. Most audio on DVD will be in Dolby Digital format, which is close to CD quality.

The final assessment of DVD quality is in the hands of consumers. Most initial reports consistently rate it better than laserdisc. No one can guarantee the quality of DVD, just as no one should dismiss it based on demos or hearsay. In the end it's a matter of individual perception.

What are the disadvantages of DVD?

Despite several positive attributes mentioned above there are some potential disadvantages of DVD:

- It will take years for movies and software to become widely available.
- It can't record (yet).
- It has built-in copy protection and regional lockout.
- It uses digital compression. Poorly compressed audio or video may be blocky, fuzzy, harsh,
- or vague.
- The audio downmix process for stereo/Dolby Surround can reduce dynamic range.
- It doesn't fully support HDTV.
- Some DVD players and drives may not be able to read CD-Rs.
- First-generation DVD players and drives can't read DVD-RAM discs.
- Current players can't play in reverse at normal speed.

Compatibility of DVD

DVD is compatible with most other optical media storage (but there is a distinction between DVD and DVD-ROM below:

- CD audio (CD-DA) -- All DVD players and drives will read audio CDs (Red Book). This is not actually required by the DVD spec, but so far all manufacturers have stated that their DVD hardware will read CDs. On the other hand, you can't play a DVD in a CD player. (The pits are smaller, the tracks are closer together, the data layer is a different distance from the surface, the modulation is different, the error correction coding is new, etc.)
- CD-ROM is compatible with DVD-ROM -- All DVD-ROM drives will read CD-ROMs (Yellow Book). However, DVD-ROMs are not readable by CD-ROM drives.
- CD-R maybe compatible with DVD-ROM -- The problem is that CD-Rs (Orange Book Part II) are ***invisible*** to DVD laser wavelength because the dye used in CD-Rs doesn't reflect the beam. This problem is being addressed in many ways. Sony has developed a twin-laser pickup in which one laser is used for reading DVDs and the other for reading CDs and CD-Rs. Samsung has also announced dual-laser using a holographic annular masked lens. These solutions provide complete backwards compatibility with millions of CD-R discs. Philips has also stated that its DVD-ROM drives will read CD-Rs. In addition, new CD-R Type II blanks that will work with CD-ROM and DVD are supposedly in development. In the meantime, some first-generation DVD-ROM drives and many first-generation DVD-Video players will not read CD-R media.
- Is CD-RW is compatible with DVD -- CD-Rewritable (Orange Book Part III) discs can not be read by existing CD-ROM drives and CD players. CD-RW has a lower reflectivity difference, requiring automatic-gain-control (AGC) circuitry. The new ***MultiRead*** standard addresses this and some DVD manufacturers have already suggested they will support it. Supposedly the optical circuitry of DVD-ROM drives and DVD players is good enough to read CD-RW. CD-RW does not have the ***invisibility*** problem of CD-R .
- Video CD maybe compatible with DVD -- It's not required by the DVD spec, but it's trivial to support the White Book standard since any MPEG-2 decoder can also decode MPEG-1 from a Video CD. Panasonic, RCA, Samsung, and Sony models play Video CDs. Japanese Pioneer models play Video CDs but American models don't. Toshiba players don't play Video CDs

  VCD resolution is 352x288 for PAL and 352x240 for NTSC. The way most DVD players and Video CD players deal with the difference is to chop off the extra lines or add blank lines. When playing PAL VCDs, the Panasonic and RCA

NTSC players apparently cut the entire 48 lines (17bottom and none off the top. The Sony looks better. [Does anyone know if it cuts 24 lines off the top and the bottom, or if it scales the full picture to fit?]

Most DVD-ROM computers will be able to play Video CDs (with the right software), since its already possible with current-model CD-ROM computers.

Note: Many Asian VCDs achieve ***two*** soundtracks by putting one language on the left channel and another on the right. They will be mixed together into babel on a stereo system unless you adjust the balance to get only one channel.

- Photo CD is **NOT** compatible with DVD -- DVD players could support Photo CD with a few extra chips and a license from Kodak. No one has announced such a player. Most DVD-ROM drives will read Photo CDs (if they read CD-Rs) since it's trivial to support the XA and Orange Book multisession standards. The more important question is, ``Does the OS or application support Photo CD"
- CD-I **NOT** compatible with DVD -- Most DVD players will not play CD-I (Green Book) discs. However, Philips has announced that it will make a DVD player that supports CD-I. Some people expect Philips to create a ***DVD-I*** format in attempt to breathe a little more life into CD-I (and recover a bit more of the billion or so dollars they've invested in it).
- Enhanced CD is compatible with DVD -- DVD players will play music from Enhanced Music CDs (Blue Book, CD Plus, CD Extra), and DVD-ROM drives will play music and read data from Enhanced CDs. Older ECD formats such as mixed mode and track zero (pregap, hidden track) should also be compatible, but there may be a problem with DVD-ROM drivers skipping track zero (as has been the case with some new CD-ROM drivers).
- Laserdisc is **NOT** compatible with DVD -- Standard DVD players will not play laserdiscs, and you can't play a DVD disc on any standard laserdisc player. (Laserdisc uses analog video, DVD uses digital video; they are very different formats.)

  However, Pioneer and Samsung have announced combo players that will play laserdiscs and DVDs (and also CDVs and audio CDs). Denon is rumored to have an LD/DVD player in the works also.

Sizes and capacities of DVD

There are many variations on the DVD theme. There are two physical sizes: 12 cm (4.7 inches) and 8 cm (3.1 inches), both 1.2 mm thick. These are the same form factors as CD. A DVD disc can be single-sided or double-sided. Each side can have one or two layers of data. The amount of video a disc can hold depends on how much audio accompanies it and how heavily the video and audio are compressed. The oft-quoted figure of 133 minutes is apocryphal: a DVD with only one audio track easily holds over 160 minutes, and a single layer can actually hold up to 9 hours of video and audio if it's compressed to VHS quality.

At a rough average rate of 4.7 Mbps (3.5 Mbps for video, 1.2 Mbps for three 5.1-channel soundtracks), a single-layer DVD holds around 135 minutes. A two-hour movie with three soundtracks can average 5.2 Mbps. A dual-layer disc can hold a two-hour movie at an average of 9.5 Mbps (very close to the 10.08 Mbps limit).

Capacities of DVD:

For reference, a CD-ROM holds about 650 MB (megabytes), which is 0.64 GB (gigabytes) or 0.68 G bytes (billion bytes). In the list below, SS/DS means single-/double-sided, SL/DL means single-/dual-layer, GB means gigabytes ($2^{3}0$), G means billions of bytes (10^{9}).

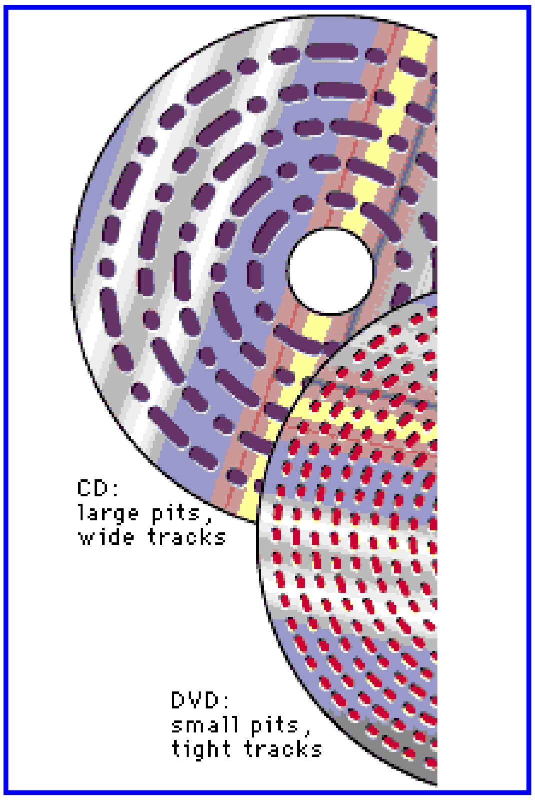
```
DVD-5 (12cm, SS/SL): 4.38 GB (4.7 G) of data, over 2 hours of video
DVD-9(12cm, SS/DL): 7.95 GB (8.5 G), about 4 hours
DVD-10 (12cm, DS/SL): 8.75 GB (9.4 G), about 4.5 hours
DVD-18 (12cm, DS/DL): 15.90 GB (17 G), over 8 hours
DVD-1? (8cm, SS/SL): 1.36 (1.4 G), about half an hour
DVD-2? (8cm, SS/DL): 2.48 GB (2.7 G), about 1.3 hours
DVD-3? (8cm, DS/SL): 2.72 GB (2.9 G), about 1.4 hours
DVD-4? (8cm, DS/DL): 4.95 GB (5.3 G), about 2.5 hours
DVD-R (12cm, SS/SL): 3.68 GB (3.95 G)
DVD-R (12cm, DS/SL): 7.38 GB (7.9 G)
DVD-R (8cm, SS/SL): 1.15 GB (1.23 G)
DVD-R (8cm, DS/SL): 2.29 GB (2.46 G)
DVD-RAM (12cm, SS/SL): 2.40 GB (2.58 G)
DVD-RAM (12cm, DS/SL): 4.80 GB (5.16 G)
```

Note: It takes about two gigabytes to store one hour of average video (Chapter 6).

The increase in capacity from CD-ROM is due to:

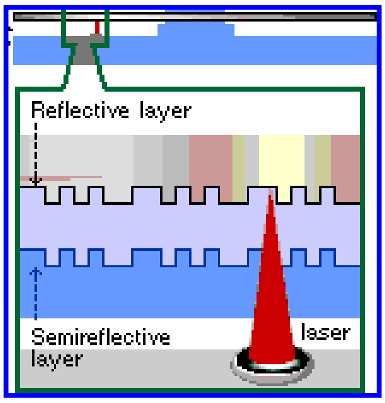
- smaller pit length ($\sim$ 2.08x) (Fig.(5.5),

- tighter tracks (~ 2.16x),
- slightly larger data area (~ 1.02x),
- slightly larger data area (~ 1.02x),



DVD vs CD-ROM Pit Length

- discs single or double sided
- another data layer added to each side creating a potential for four layers of data per disc (Fig. 5.6)



DVD layers

- more efficient channel bit modulation (~ 1.06x),
- more efficient error correction (~ 1.32x),
- less sector overhead (1.06x). Total increase for a single layer is about 7 times a standard CD-ROM. There's a slightly different explanation at http://www.mpeg.org/MPEG/DVD/General/Gain.html

The capacity of a dual-layer disc is slightly less than double that of a single-layer disc. The laser has to read through the outer layer to the inner layer (a distance of 20 to 70 microns). To reduce inter-layer crosstalk, the minimum pit length of both layers is increased from .4 um to .44 um. In addition, the reference scanning velocity is slightly faster - 3.84 m/s, as opposed to 3.49 m/s for single layer discs. Bigger pits, spaced farther apart, are easier to read correctly and are less susceptible to jitter. Bigger pits and fewer of them mean reduced capacity per layer.

Next Up Previous
Next: DVD audio **Up:** DVD **Previous:** Sizes and capacities of

DVD video details

We will look at the details of video and audio in general in Chapter 6 and compression (MPEG formats) in Chapter 7. These concepts are mentioned below and will be explained in further detail in these later chapters.

DVD-Video is an application of DVD-ROM. DVD-Video is also an application of MPEG-2. This means the DVD format defines subsets of these standards to be applied in practice as DVD-Video. DVD-ROM can contain any desired digital information, but DVD-Video is limited to certain data types designed for television reproduction.

A disc has one track (stream) of MPEG-2 constant bit rate (CBR) or variable bit rate (VBR) compressed digital video. A limited version of MPEG-2 Main Profile at Main Level (MP@ML) is used. MPEG-1 CBR and VBR video is also allowed. 525/60 (NTSC, 29.97 interlaced frames/sec) and 625/50 (PAL, 25 interlaced frames/sec) video systems are supported. Coded frame rates of 24 fps progressive or interlaced from film, 25 fps interlaced from PAL video, and 29.97 fps interlaced from NTSC video are supported. In the case of 24 fps source, the encoder embeds MPEG-2 repeat_first_field flags into the video stream to make the decoder either perform 3-2 pulldown for 60 (59.94) Hz displays or 2-2 pulldown (with 4displays. In other words, the player doesn't really know what the encoded rate is, it simply follows the MPEG-2 encoder's instructions to arrive at the predetermined display rate of 25 fps or 29.97 fps. (No current players convert from PAL to NTSC or NTSC to PAL.) See the Chapter 7 for more information on MPEG-2 video.

Picture dimensions are max 720x480 (29.97 frames/sec) or 720x576 (25 frames/sec). Pictures are subsampled from 4:2:2 ITU-R 601 down to 4:2:0, allocating an average of 12 bits/pixel. (Color depth is still 24 bits, since color samples are shared across 4 pixels.) The uncompressed source is 124.416 Mbps for video source (720x480x12x30 or 720x576x12x25), or either 99.533 or 119.439 Mbps for film source (720x480x12x24 or 720x576x12x24). Using the traditional (and rather subjective) television measurement of ***lines of horizontal resolution*** DVD can have 540 lines on a standard TV (720/(4/3)) and 405 on a widescreen TV (720/(16/9)). In practice, most DVD players provide about 500 lines because of filtering. VHS has about 230 (172 w/s) lines and laserdisc has about 425 (318 w/s).

Different players use different numbers of bits for the video digital-to-analog converter. (Sony and Toshiba use 10 bits, Pioneer and Panasonic use 9 bits.) This has nothing to do with the MPEG decoding process. It provides more ***headroom*** and more analog signal levels which supposedly give a better picture.

Maximum video bitrate is 9.8 Mbps. The ***average*** bitrate is 3.5 but depends entirely on the length, quality, amount of audio, etc. This is a 36:1 reduction from uncompressed 124 Mbps (or a 28:1 reduction from 100 Mbps film source). Raw channel data is read off the disc at a constant 26.16 Mbps. After 8/16 demodulation it's down to 13.08 Mbps. After error correction the user data stream goes into the track buffer at a constant 11.08 Mbps. The track buffer feeds system stream data out at a variable rate of up to 10.08 Mbps. After system overhead, the maximum rate of combined elementary streams (audio + video + subpicture) is 10.08. MPEG-1 video rate is limited to 1.856 Mbps with a typical rate of 1.15 Mbps.

Still frames (encoded as MPEG-2 I-frames) are supported and can be displayed for a specific amount of time or indefinitely. These are generally used for menus. Still frames can be accompanied by audio.

A disc also can have up to 32 subpicture streams that overlay the video for subtitles, captions for the hard of hearing, captions for children, karaoke, menus, simple animation, etc. These are full-screen, run-length-encoded bitmaps limited to four pixel types. For each group of subpictures, four colors are selected from a palette of 16 (from the YCrCb gamut), and four contrast values are selected out of 16 levels from transparent to opaque. Subpicture display command sequences can be used to create effects such as scroll, move, color/highlight, and fade. The maximum subpicture data rate is 3.36 Mbps, with a maximum size per frame of 53220 bytes.

Video can be stored on a DVD in 4:3 format (standard TV shape) or 16:9 (widescreen). The 16:9 format is ***anamorphic***, meaning the picture is squeezed horizontally to fit a 4:3 rectangle then unsqueezed during playback. DVD players can output video in four different ways:

- full frame (4:3 video for 4:3 display)
- letterbox (16:9 video for 4:3 display)
- pan and scan (16:9 video for 4:3 display)
- widescreen (16:9 video for 16:9 display)

Video stored in 4:3 format is not changed by the player. It will appear normally on a standard 4:3 display. Widescreen systems will either enlarge it or add black bars to the sides. 4:3 video may have been formatted in various ways before being transferred to DVD. For example, it may have been letterboxed to hold video with a wider shape. Or it may have been panned and scanned from film composed for a wider theatrical presentation. All formatting done to the video prior to it being stored on the disc is transparent to the player. It merely reproduces the signal in standard form.

For automatic letterbox mode, the player creates black bars at the top and the bottom of the picture (60 lines each for NTSC, 72 for PAL). This leaves 3/4 of the height remaining, creating a shorter but wider rectangle. In order to fit this shorter rectangle, the picture is squeezed vertically using a ***letterbox filter*** that combines every 4 lines into

3. This compensates for the original horizontal squeezing, resulting in the movie being shown in its full width. The vertical resolution is reduced from 480 lines to 360.

For automatic pan and scan mode, the video is unsqueezed to 16:9 and a portion of the image is shown at full height on a 4:3 screen by following a `center of interest' offset that's encoded in the video stream according to the preferences of the people who transferred the film to video. The pan and scan ***window*** is 75% of the full width, which reduces the horizontal pixels from 720 to 540. The pan and scan window can only travel laterally. This does not duplicate a true pan and scan process in which the window can also travel up and down and zoom in and out. Therefore, most DVD producers choose to put a separate pan and scan version on the disc in addition to the widescreen version.

For widescreen mode, the anamorphic video is stretched back out by widescreen equipment to its original width. If anamorphic video is shown on a standard 4:3 display, people will look like they have been on a crash diet. Widescreen mode is complicated because most movies today are shot with a ***soft matte***. (The cinematographer has two sets of frame marks in her viewfinder, one for 1.33 (4:3) and one for 1.85, so she can allow for both formats). A few movies are even wider, such as the 2.35 ratio of Panavision. Since most movies are wider than 1.78 (16:9), one of at least 4 methods must be used during transfer to make it fit the 1.78 rectangle: 1) add additional thin black bars to the top and bottom; 2) include a small amount of extra picture at the top and bottom from the soft matte area; 3) crop the sides; 4) pan and scan with a 1.78 window. With the first two methods, the difference between 1.85 and 1.78 is so small that the letterbox bars or extra picture are hidden in the overscan area of most televisions. Nevertheless, and especially with 2.35 movies, many DVD producers put 16:9 source on one side (or layer) of the disc and 4:3 source on the other. This way the full-frame version of the film can be used for a horizontal and vertical pan and scan, and zoom process with no letterbox bars and no reduction in resolution.

Anamorphosis causes no problems with line doublers, which simply double the lines before they are stretched out by the widescreen display.

For anamorphic video, the pixels are fatter. Different pixel aspect ratios (none of them square) are used for each aspect ratio and resolution. 720-pixel and 704-pixel sizes have the same aspect ratio because the first includes overscan. Note that conventional values of 1.0950 and 0.9157 are for height/width (and are tweaked to match scanning rates). The table below uses less-confusing width/height values (y/x * h/w).

	720x480 704x480	720x576 704x486	352x480	352x576
4:3	0.909	1.091	1.818	2.182
16:9	1.212	1.455	2.424	2.909

Playback of widescreen material can be restricted. Programs can be marked for the following display modes:

- 4:3 full frame
- 4:3 LB (for automatically setting letterbox expand mode on widescreen TV)
- 16:9 LB only (player not allowed to pan and scan on 4:3 TV)
- 16:9 PS only (player not allowed to letterbox on 4:3 TV)
- 16:9 LB or PS (viewer can select pan and scan or letterbox on 4:3 TV)

Next Up Previous
Next: Interactive DVD features **Up:** DVD **Previous:** DVD video details

DVD audio

The DVD-Audio format is not yet specified. The International Steering Committee announced it expects to have a final draft specification by December 1997. This means DVD-Audio products may show up around 1999.

The following details are for audio tracks on DVD-Video. Some DVD manufacturers such as Pioneer are developing audio-only players using the DVD-Video format.

A disc can have up to 8 audio tracks (streams). Each track can be in one of three formats:

- Dolby Digital (formerly AC-3): 1 to 5.1 channels
- MPEG-2 audio: 1 to 5.1 or 7.1 channels
- PCM: 1 to 8 channels.

Two additional optional formats are supported: DTS and SDDS. Both require external decoders.

The ***.1*** refers to a low-frequency effects (LFE) channel that connects to a subwoofer. This channel carries an emphasized bass audio signal.

All five audio formats support karaoke mode, which has two channels for stereo (L and R) plus an optional guide melody channel (M) and two optional vocal channels (V1 and V2).

Discs containing 525/60 (NTSC) video must use PCM or Dolby Digital on at least one track. Discs containing 625/50 (PAL/SECAM) video must use PCM or MPEG audio on at least one track. Additional tracks may be in any format. The DVD Forum has clarified that only stereo MPEG audio is mandatory for 625/50 discs, while multichannel MPEG-2 audio is recommended. Since multichannel MPEG-2 decoders are not yet available, most 625/50 discs include Dolby Digital audio.

For stereo output (analog or digital), all NTSC players and all PAL players (so far) have a built-in Dolby Digital decoder which downmixes from 5.1 channels (if present on the disc) to Dolby Surround stereo (i.e., 5 channels are matrixed into 2 channels to be decoded to 4 by an external Dolby Pro Logic processor). Both Dolby Digital and MPEG-2 support 2-channel Dolby Surround as the source in cases where the disc producer can't or doesn't want to remix the original onto discrete channels. This means that a DVD labelled as having Dolby Digital sound may only use the L/R channels for surround or ***plain*** stereo. Even movies with old monophonic soundtracks may use Dolby Digital -

but only 1 or 2 channels.

The downmix process does not include the LFE channel and may compress the dynamic range in order to improve dialog audibility and keep the sound from becoming ***muddy*** on average home audio systems. This can result in reduced sound quality on high-end audio systems. Some players have the option to turn off the dynamic range compression. The downmix is auditioned when the disc is prepared, and if the result is not acceptable the audio may be tweaked or a separate L/R Dolby Surround track may be added. Experience has shown that minor tweaking is sometimes required to make the dialog more audible within the limited dynamic range of a home stereo system, but that a separate track is not usually necessary. If surround audio is important to you, you will hear significantly better results from multichannel discs if you have a Dolby Digital system.

Linear PCM is uncompressed (lossless) digital audio, the same format used on CDs. It can be sampled at 48 or 96 kHz with 16, 20, or 24 bits/sample. (Audio CD is limited to 44.1 kHz at 16 bits.) There can be from 1 to 8 channels. The maximum bitrate is 6.144 Mbps, which limits sample rates and bit sizes with 5 or more channels. It's generally felt that the 96 dB dynamic range of 16 bits or even the 120 dB range of 20 bits combined with a frequency response of up to 22,000 Hz from 48 kHz sampling is adequate for high-fidelity sound reproduction. However, additional bits and higher sampling rates are useful in studio work, noise shaping, advanced digital processing, and three-dimensional sound field reproduction. DVD players are required to support all the variations of LPCM, but some of them may subsample 96 kHz down to 48 kHz, and some may not use all 20 or 24 bits. The signal provided on the digital output for external digital-to-analog converters may be limited to less than 96 kHz or less than 24 bits.

Dolby Digital is multi-channel digital audio, compressed using AC-3 coding technology from original PCM with a sample rate of 48 kHz at 16 bits. The bitrate is 64 kbps to 448 kbps, with 384 being the normal rate for 5.1 channels and 192 being the normal rate for stereo (with or without surround encoding). The channel combinations are (front/surround): 1/0, 1+1/0 (dual mono), 2/0, 3/0, 2/1, 3/1, 2/2, and 3/2. The LFE channel is optional with all 8 combinations.

MPEG audio is multi-channel digital audio, compressed from original PCM format with sample rate of 48 kHz at 16 bits. Both MPEG-1 and MPEG-2 formats are supported. The variable bitrate is 32 kbps to 912 kbps, with 384 being the normal average rate. MPEG-1 is limited to 384 kbps. Channel combinations are (front/surround): 1/0, 2/0, 2/1, 2/2, 3/0, 3/1, 3/2, and 5/2. The LFE channel is optional with all combinations. The 7.1 channel format adds left-center and right-center channels, but will probably be rare for home use. MPEG-2 surround channels are in an extension stream matrixed onto the MPEG-1 stereo channels, which makes MPEG-2 audio backwards compatible with MPEG-1 hardware (an MPEG-1 system will only see the two stereo channels.)

DTS is an optional multi-channel (5.1) digital audio format, compressed from PCM at 48 kHz. The data rate is from 64 kbps to 1536 kbps. Channel combinations are (front/surround): 1/0, 2/0, 3/0, 2/1, 2/2, 3/2. The LFE channel is optional with all 6 combinations.

SDDS is an optional multi-channel (5.1 or 7.1) digital audio format, compressed from PCM at 48 kHz. The data rate can go up to 1280 kbps.

A DVD-5 with only one surround stereo audio stream (at 192 kbps) can hold over 55 hours of audio. A DVD-18 can hold over 200 hours.

Interactive DVD features

DVD-Video players (and software DVD-Video navigators) support a command set that provides rudimentary interactivity. The main feature is menus, which are present on almost all discs to allow content selection and feature control. Each menu has a still-frame graphic and up to 36 highlightable, rectangular buttons (only 12 if widescreen, letterbox, and pan and scan modes are used). Remote control units have four arrow keys for selecting onscreen buttons, plus numeric keys, select key, menu key, and return key. Additional remote functions may include freeze, step, slow, fast, scan, next, previous, audio select, subtitle select, camera angle select, play mode select, search to program, search to part of title (chapter), search to time, and search to camera angle. Any of these features can be disabled by the producer of the disc.

Additional features of the command set include simple math (add, subtract, multiply, divide, modulo, random), bitwise and, bitwise or, bitwise xor, plus comparisons (equal, greater than, etc.), and register loading, moving, and swapping. There are 24 system registers for information such as language code, audio and subpicture settings, and parental level. There are 16 general registers for command use. A countdown timer is also provided. Commands can branch or jump to other commands. Commands can also control player settings, jump to different parts of the disc, and control presentation of audio, video, subpicture, camera angles, etc.

DVD-V content is broken into ***titles*** (movies or albums), and ***parts of titles*** (chapters or songs). Titles are made up of ***cells*** linked together by one or more ***program chains*** (PGC). A PGC can be defined as sequential play, random play (may repeat), or shuffle play (random order but no repeats). Individual cells may be used by more than one PGC, which is how parental management and seamless branching are accomplished: different PGCs define different sequences through mostly the same material.

Additional material for camera angles and seamless branching is interleaved together in small chunks. The player jumps from chunk to chunk, skipping over unused angles or branches, to stitch together the seamless video. Since angles are stored separately, they have no direct effect on the bitrate but they do affect the playing time. Adding 1 camera angle for a program roughly doubles the amount of space it requires (and cuts the playing time in half).

DVD and computers

SO far we have focussed on the media representation and standard DVD players. DVD and DVD-ROM in particular is beginning to have a huge impact on computers.

For a computer to employ DVD it must have the following features:

In addition to a DVD-ROM drive, you must have extra hardware to decode MPEG-2 video and Dolby Digital/MPEG-2/PCM audio. The computer operating system or playback system must support regional codes and be licensed to decrypt copy-protected movies. You may also need software that can read the MicroUDF format used to store DVD data files and interpret the DVD control codes. It's estimated that 10-30% of new computers with DVD-ROM drives will include decoder hardware, and that most of the remaining DVD-ROM computers will include movie playback software.

Some DVD-Videos and many DVD-ROMs will use video encoded using MPEG-1 instead of MPEG-2. Many existing computers have MPEG-1 hardware built in or are able to decode MPEG-1 with software.

CompCore Multimedia and Mediamatics make software to play DVD-Video movies (SoftDVD, DVD Express). Both require at least a 233 MHz Pentium MMX with AGP and an IDE/SCSI DVD-ROM drive with bus mastering DMA support to achieve about 20 frame/sec film rates (or better than 300 MHz for 30 frame/sec video), and can decrypt copy-protected movies. Oak's software requires hardware support. The software ***navigators*** support most DVD-Video features (menus, subpictures, etc.) and can emulate a DVD-Video remote control.

CompCore, Mediamatics, and Oak Technology have defined standards to allow certain MPEG decoding tasks to be performed by hardware on a video card and the remainder by software. Video graphics controllers with this feature are being called ***DVD MPEG-2 accelerated***. (The Mediamatics standard is called MVCCA.)

If you have at least a 433 MHz Alpha workstation you'll be able to play DVD movies at full 30 fps in software.

DVD-ROM Drives:

Most DVD-ROM drives have a seek time of 150-200 ms, access time of 200-250 ms, and data transfer rate of 1.3 MB/s ($11.08*10^6/8/2^{20}$) with burst transfer rates of up to 12 MB/s or higher. The data transfer rate from DVD-ROM discs is roughly equivalent to a

9x CD-ROM drive. DVD spin rate is about 3 times faster than CD, so when reading CD-ROMs, some DVD-ROM drives transfer data at 3x speed while others are faster. 2x and 3x DVD-ROM drives are already in the works. Hitachi is shipping samples of a 2x DVD-ROM drive which also reads CDs at 20x.

Connectivity is similar to that of CD-ROM drives: EIDE (ATAPI), SCSI-2, etc. All DVD-ROM drives have audio connections for playing audio CDs. No DVD-ROM drives have been announced with DVD audio or video outputs (which would require internal audio/video decoding hardware).

DVD-ROMs use a MicroUDF/ISO 9660 bridge file system. The OSTA UDF file system will eventually replace the ISO 9660 system of CD-ROMs, but the bridge format provides backwards compatibility until operating systems support UDF.

Recordable DVD-ROM: DVD-R and DVD-RAM:

There are two recordable versions of DVD-ROM: DVD-R (record once) and DVD-RAM (erase and record many times), with capacities of 3.95 and 2.58 G bytes. Both specifications have been published. DVD-R and DVD-RAM are not currently usable for home video recording.

DVD-R uses organic dye polymer technology like CD-R and is compatible with almost all DVD drives. The technology will improve to support 4.7 G bytes in 1 to 2 years, which is crucial for desktop DVD-ROM and DVD-Video production.

- Further Information

Further Information

Further information on DVD can be obtained from:

http://www.dvddigital.com/

In fact this section is heavily based on the FAQs section at

http://www.dvddigital.com/facts/

Multimedia Data

Dave Marshall
10/4/2001

Multimedia Data Representations

In this chapter we focus on the underlying representations of common forms of media Audio, Graphics and Video. In the next chapter we focus on compression techniques as we shortly understand why these data can be large in storage size (even when compressed).

The topics we consider here are specifically:

- Digital Audio
- Sampling/Digitisation
- Compression (Details of Compression algorithms Next Chapter)
- Graphics/Image Formats
- Digital Video (Basics of Video end of this chapter but more on Didgiatl Video in next Chapter as it so closely entwined with Compression)

Dave Marshall
10/4/2001

Basics of Digital Audio

Dave Marshall
10/4/2001

Application of Digital Audio -- Selected Examples

Music Production

- Hard Disk Recording
- Sound Synthesis
- Samplers
- Effects Processing

Video

- Audio Important Element: Music and Effects

Web

-- Many uses on Web

- Spice up Web Pages
- Listen to Cds
- Listen to Web Radio

Many More Uses

-- try and think of some?

Dave Marshall
10/4/2001

Digitization of Sound

Let us first analyse what a sound actually is:

- Sound is a continuous wave that travels through the air
- The wave is made up of pressure differences. Sound is detected by measuring the pressure level at a location.
- Sound waves have normal wave properties (reflection, refraction, diffraction, etc.).

A variety of sound sources:

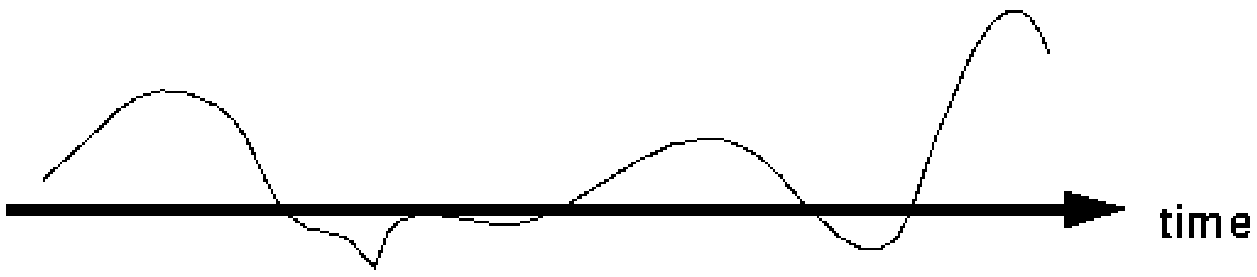
Source
-- Generates Sound
  - Air Pressure changes
  - ***Electrical*** **--** Loud Speaker
  - ***Acoustic*** **--** Direct Pressure Variations

The destination receives (sensed the sound wave pressure changes) and has to deal with accordingly:

Destination
-- Receives Sound
  - ***Electrical*** **--** Microphone produces electric signal
  - ***Ears*** **--** Responds to pressure **hear** sound

Sound is required input into a computer: it needs to sampled or digitised:

- Microphones, video cameras produce *analog signals* (continuous-valued voltages) as illustrated in Fig 6.1

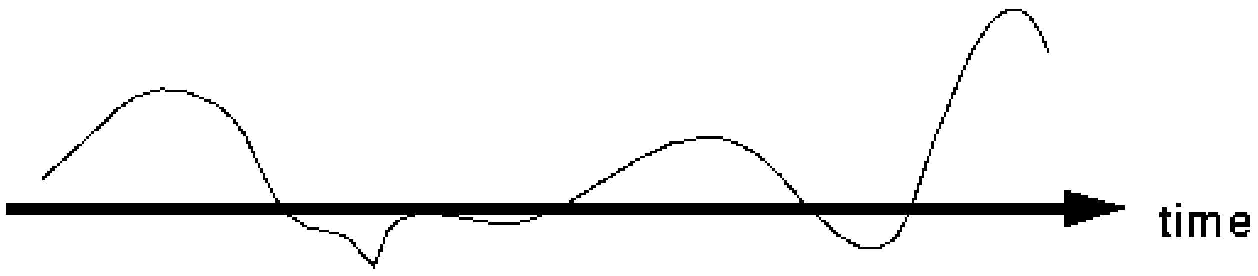


Continuous Analog Waveform

- To get audio or video into a computer, we have to *digitize* it (convert it into a stream of numbers) **Need to convert Analog-to-Digital --** Specialised Hardware

- So, we have to understand *discrete sampling* (both time and voltage)

- *Sampling* - divide the horizontal axis (the time dimension) into discrete pieces. Uniform sampling is ubiquitous.
- *Quantization* - divide the vertical axis (signal strength) into pieces. Sometimes, a non-linear function is applied.
    - 8 bit quantization divides the vertical axis into 256 levels. 16 bit gives you 65536 levels.



Continuous Analog Waveform

Next Up Previous
Next: Digitizing Audio **Up:** Basics of Digital Audio **Previous:** Application of Digital Audio
Dave Marshall
10/4/2001

Digitizing Audio

That is the basic idea of digitizing a sound unfortunately things are (practically speaking) not so simple.

- Questions for producing digital audio (Analog-to-Digital Conversion):

  1. How often do you need to sample the signal?
  2. How good is the signal?
  3. How is audio data formatted?

Dave Marshall
10/4/2001

Computer Manipulation of Sound

Once Digitised processing the digital sound is essentially straightforward although it depends on the processing you wish to do (***e.g.*** volume is easier to code than accuarte reverb)

Essentially they all operate on the 1-D array of digitised samples, typical examples include:

- Volume
- Cross-Fading
- Looping
- Echo/Reverb/Delay
- Filtering
- Signal Analysis

Soundedit Demos

- Volume
- Cross-Fading
- Looping
- Echo/Reverb/Delay
- Filtering

Dave Marshall
10/4/2001

Sample Rates and Bit Size

How do we store each sample value (***Quantisation***)?

8 Bit Value
(0-255)

16 Bit Value
(Integer) (0-65535)

How many Samples to take?

11.025 KHz
-- Speech (Telephone 8KHz)

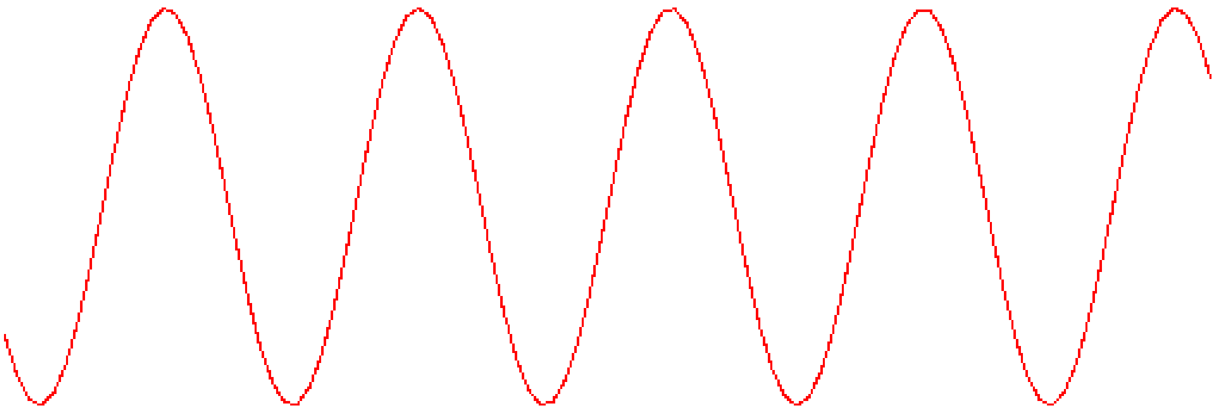
22.05 KHz
-- Low Grade Audio
(WWW Audio, AM Radio)

44.1 KHz
-- CD Quality

Dave Marshall
10/4/2001

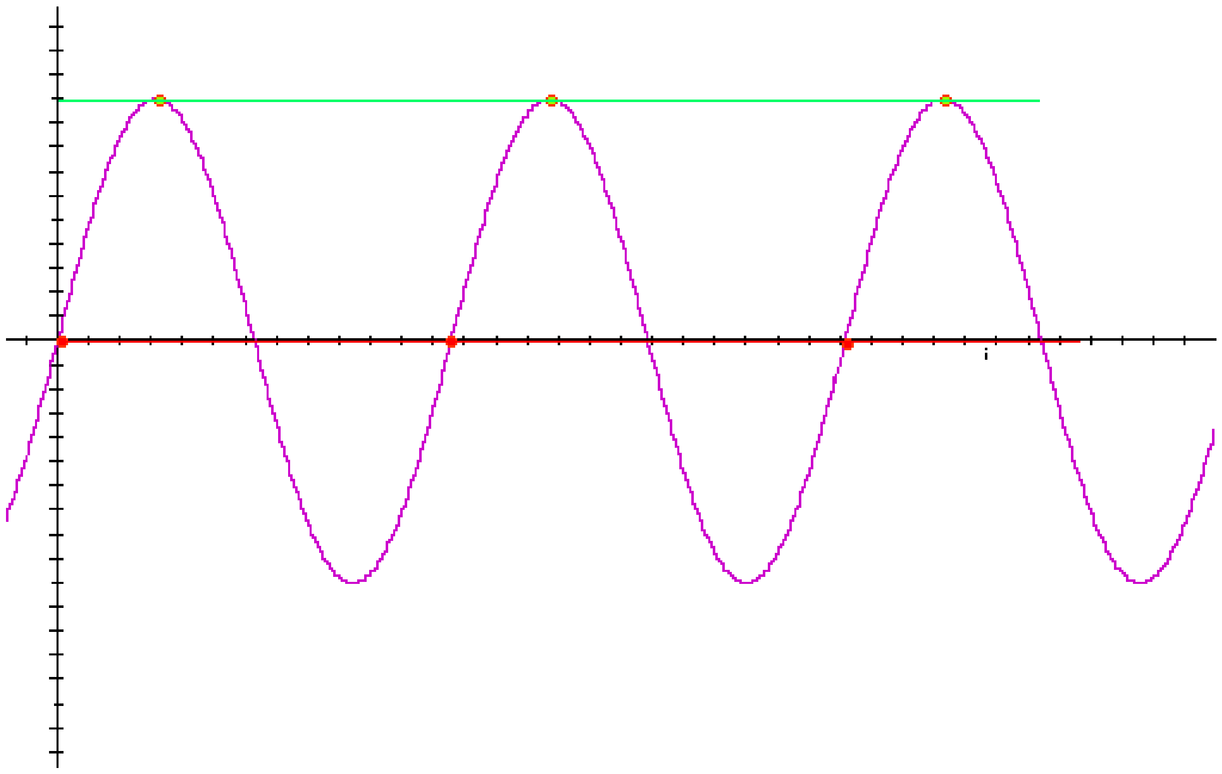
Nyquist's Sampling Theorem

- Suppose we are sampling a sine wave (Fig 6.3. How often do we need to sample it to figure out its frequency?



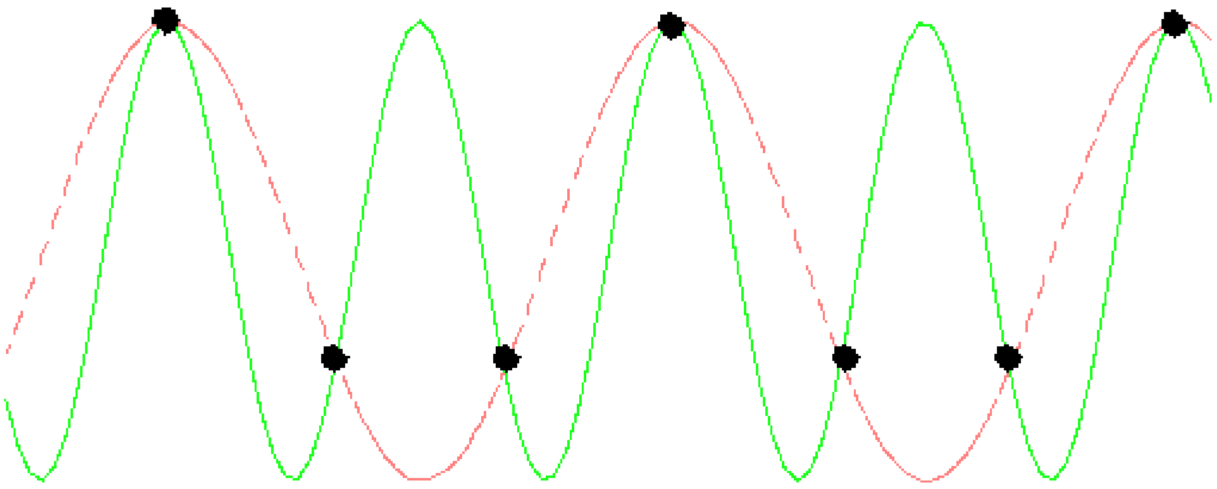
A Sine Wave

- If we sample at 1 time per cycle, we can think it's a constant (Fig 6.4)



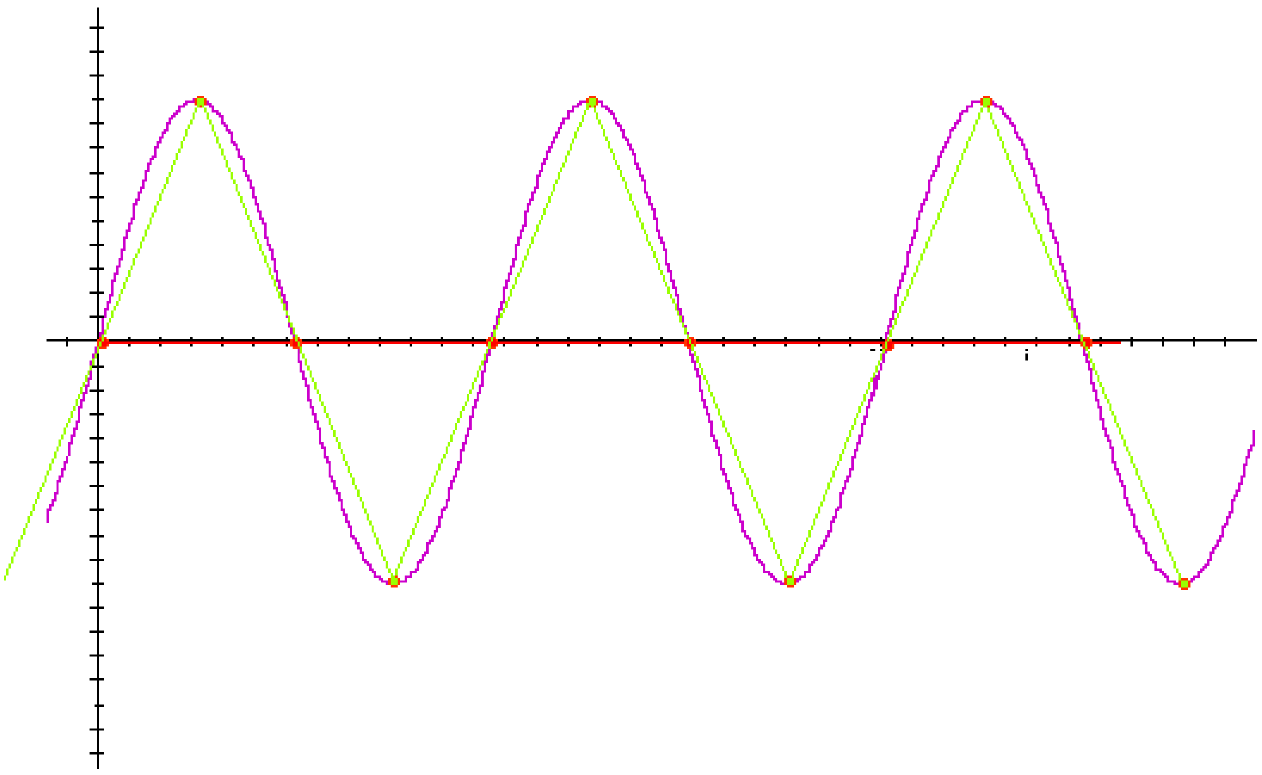
Sampling at 1 time per cycle

- If we sample at 1.5 times per cycle, we can think it's a lower frequency sine wave (Fig. 6.6)



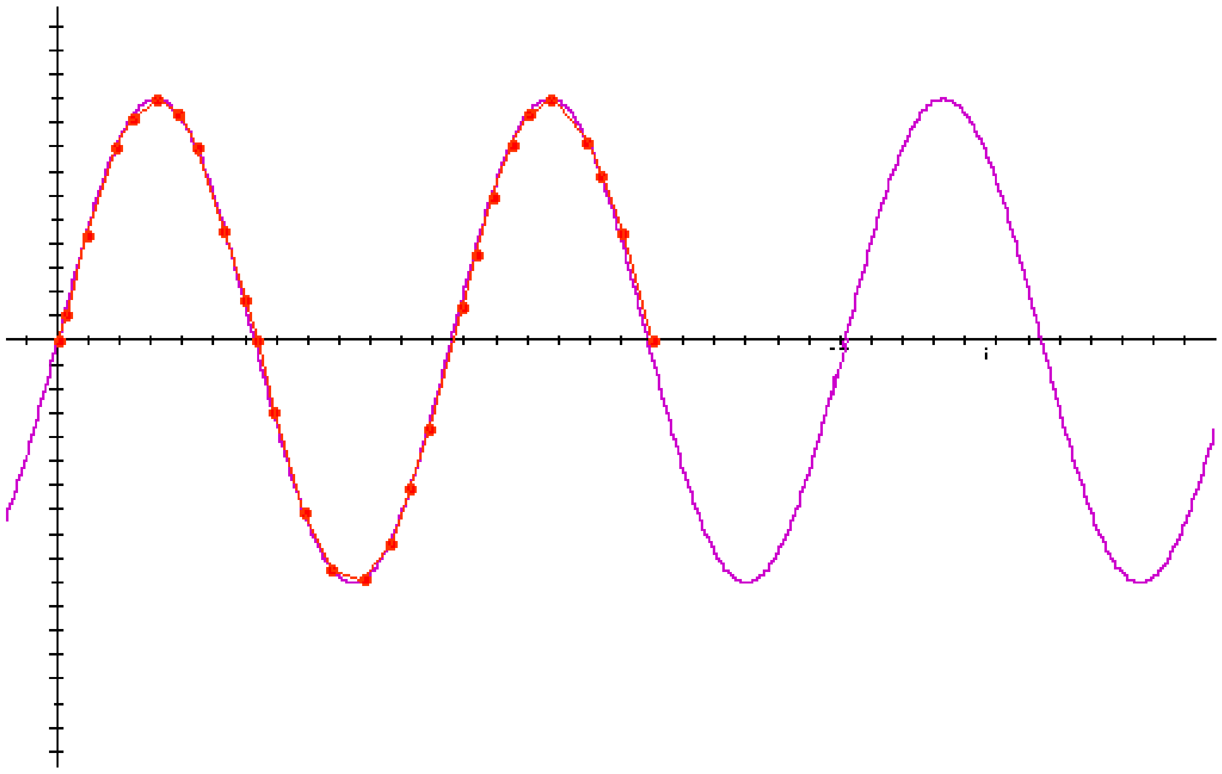
Sampling at 1.5 times per cycle

- Now if we sample at twice the sample frequency, i.e Nyquist Rate, we start to make some progress. An alternative way of viewing thr waveform (re)genereation is to think of straight lines joining up the peaks of the samples. In this case (at these sample points) we see we get a sawtooth wave that begins to start crudely approximating a sine wave



Sampling at 2 times per cycle

- **Nyquist rate** -- For lossless digitization, the sampling rate should be ***at least twice*** the maximum frequency responses. Indeed many times more the better.



Sampling at many times per cycle

Dave Marshall
10/4/2001

Implications of Sample Rate and Bit Size

Affects Quality of Audio

- Ears do not respond to sound in a linear fashion
- Decibel (**dB**) a logarithmic measurement of sound
- 16-Bit has a signal-to-noise ratio of 98 dB -- virtually inaudible
- 8-bit has a signal-to-noise ratio of 50 dB
- Therefore, 8-bit is roughly 8 times as noisy
  - 6 dB increment is twice as loud

Signal to Noise Ratio (SNR)

- In any analog system, some of the voltage is what you want to measure (*signal*), and some of it is random fluctuations (*noise*).
- Ratio of the power of the two is called the *signal to noise ratio* (*SNR*). SNR is a measure of the quality of the signal.
- SNR is usually measured in decibels (*dB*).

$$SNR = 10 \log \frac{V_{signal}^2}{V_{noise}^2} = 20 \log \frac{V_{signal}}{V_{noise}}$$

- Typically 8 bits or 16 bits.
- Each bit adds about 6 dB of resolution, so 16 bits => 96 dB.
- Samples are typically stored as raw numbers (*linear format*), or as logarithms (*u-law* (or *A-law* in Europe)).
  - Logarithmic representation approximates *perceptual uniformity*.

Affects Size of Data

File Type	*44.1 KHz*	*22.05 KHz*	*11.025 KHz*
16 Bit Stereo	10.1 Mb	5.05 Mb	2.52 Mb
16 Bit Mono	5.05 Mb	2.52 Mb	1.26 Mb
8 Bit Mono	2.52 Mb	1.26 Mb	630 Kb

Memory Required for 1 Minute of Digital Audio

There is therfore is a trade off between ***Audio Quality vs. Data Rate***

Some typical applications of sample bit size and sample rate are listed below:

Quality	Sample Rate (KHz)	Bits per Sample	Mono/ Stereo	Data Rate (Uncompressed)	Frequency Band
Telephone	8	8	Mono	8 KBytes/sec	200-3,400 Hz
AM Radio	11.025	8	Mono	11.0 KBytes/sec	
FM Radio	22.050	16	Stereo	88.2 KBytes/sec	
CD	44.1	16	Stereo	176.4 KBytes/sec	20-20,000 Hz
DAT	48	16	Stereo	192.0 KBytes/sec	20-20,000 Hz

AUDIO DEMO: Comparison of Sample Rate and Bit Size

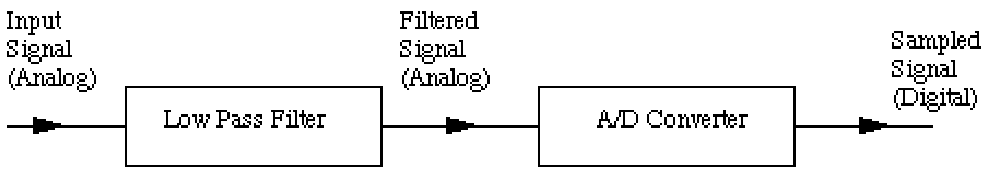
Click on the file links below to audibly hear the difference in the sampling rates/bit sizes indicated for each file type:

File Type	**File Size (all mono)**
44Hz 16 bit	3.5 Mb
44KHz 8-bit	13.Mb
22 KHz 16-bit	740 Kb
22KHz 8-Bit	424 Kb
11KHz 8-bit	120 K

- Telephone uses *u-law* encoding, others use linear. So the dynamic range of digital telephone signals is effectively 13 bits rather than 8 bits.
- CD quality stereo sound -> 10.6 MB / min.

Practical Implications of Nyquist Sampling Theory

- Must (low pass) filter signal before sampling:



Practical Implication of Nyquist;Must Low Pass Filter Signal before Sampling

- Otherwise strange artifacts from high frequency signals appear and are audible.

We'll finish off with a question: ***Why are CD Sample Rates 44.1 KHz?***

The answer should be obvious if you have paid attention to the above notes (Answer in Lecture).

Dave Marshall
10/4/2001

Typical Audio Formats

- Popular audio file formats include .au (Unix workstations), .aiff (MAC, SGI), .wav (PC, DEC workstations)
- A simple and widely used audio compression method is Adaptive Delta Pulse Code Modulation (ADPCM). Based on past samples, it predicts the next sample and encodes the difference between the actual value and the predicted value.

Dave Marshall
10/4/2001

Delivering Audio over a Network

- **Trade off between desired fidelity and file size**
- Bandwidth Considerations for Web and other media.
- Compress Files:
  - Could affect live transmission on Web

Dave Marshall
10/4/2001

Streaming Audio

- Buffered Data:
  - Trick get data to destination before it's needed
  - Temporarily store in memory (Buffer)
  - Server keeps feeding the buffer
  - Client Application reads buffer
- Needs Reliable Connection, moderately fast too.
- Specialised client, Steaming Audio Protocol (PNM for real audio).

Dave Marshall
10/4/2001

Synthetic Sounds

- FM (Frequency Modulation) Synthesis - used in low-end Sound Blaster cards, OPL-4 chip, Yamaha DX Synthesiser range popular in Early 1980's.

- Wavetable synthesis - wavetable generated from sound waves of real instruments
- Modern Synthesiser use a mixture of sample and synthesis.

Dave Marshall
10/4/2001

Introduction to MIDI (Musical Instrument Digital Interface)

Definition of MIDI: a protocol that enables computer, synthesizers, keyboards, and other musical device to communicate with each other.

Dave Marshall
10/4/2001

Components of a MIDI System

Synthesizer:

- It is a sound generator (various pitch, loudness, tone colour).
- A good (musician's) synthesizer often has a microprocessor, keyboard, control panels, memory, etc.

Sequencer:

- It can be a stand-alone unit or a software program for a personal computer. (It used to be a storage server for MIDI data. Nowadays it is more a software *music editor* on the computer.
- It has one or more MIDI INs and MIDI OUTs.

Track:

- Track in sequencer is used to organize the recordings.
- Tracks can be turned on or off on recording or playing back.

Channel:

- MIDI channels are used to separate information in a MIDI system.
- There are 16 MIDI channels in one cable.
- Channel numbers are coded into each MIDI message.

Timbre:

- The quality of the sound, e.g., flute sound, cello sound, etc.
- Multitimbral - capable of playing many different sounds at the same time (e.g., piano, brass, drums, etc.)

Pitch:

- musical note that the instrument plays

Voice:

- Voice is the portion of the synthesizer that produces sound.
- Synthesizers can have many (12, 20, 24, 36, etc.) voices.
- Each voice works independently and simultaneously to produce sounds of different timbre and pitch.

Patch:

- the control settings that define a particular timbre.

Next Up Previous
Next: Hardware Aspects of MIDI **Up:** Introduction to MIDI (Musical **Previous:** Introduction to MIDI (Musical
Dave Marshall
10/4/2001

Hardware Aspects of MIDI

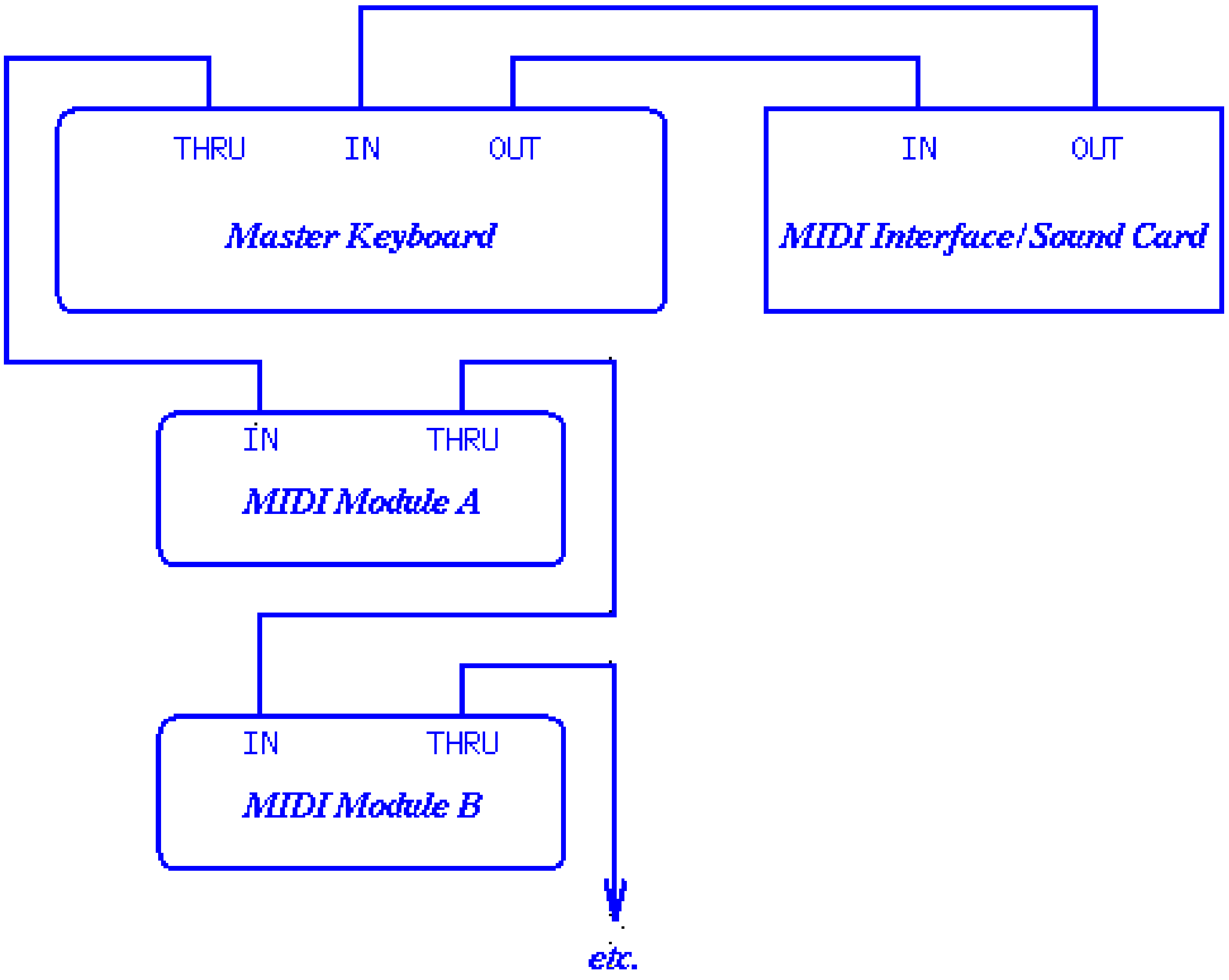
MIDI connectors:

- three 5-pin ports found on the back of every MIDI unit

- MIDI IN: the connector via which the device receives all MIDI data.
- MIDI OUT: the connector through which the device transmits all the MIDI data it generates itself.
- MIDI THROUGH: the connector by which the device echoes the data receives from MIDI IN.

Note: It is only the MIDI IN data that is echoed by MIDI through. All the data generated by device itself is sent through MIDI OUT.

Figure 6.9 illustrates a typical setup where:



A Typical MIDI Sequencer Setup

- MIDI OUT of synthesizer is connected to MIDI IN of sequencer.
- MIDI OUT of sequencer is connected to MIDI IN of synthesizer and through to each of the additional sound modules.
- During recording, the keyboard-equipped synthesizer is used to send MIDI message to the sequencer, which records them.
- During play back: messages are send out from the sequencer to the sound modules and the synthesizer which will play back the music.

Dave Marshall
10/4/2001

MIDI Messages

MIDI messages are used by MIDI devices to communicate with each other.

Structure of MIDI messages:

- MIDI message includes a status byte and up to two data bytes.
- Status byte
  - The most significant bit of status byte is set to 1.
  - The 4 low-order bits identify which channel it belongs to (four bits produce 16 possible channels).
  - The 3 remaining bits identify the message.
- The most significant bit of data byte is set to 0.

Classification of MIDI messages:

```
                                                        ----- voice messages
                  ---- channel messages -----|
                 |                              ----- mode messages
                 |
MIDI messages ----|
                 |                              ---- common messages
                  ----- system messages -----|---- real-time messages
                                                ---- exclusive messages
```

A. Channel messages:

- messages that are transmitted on individual channels rather that globally to all devices in the MIDI network.

A.1. Channel voice messages:

- Instruct the receiving instrument to assign particular sounds to its voice
- Turn notes on and off
- Alter the sound of the currently active note or notes

Voice Message	Status Byte	Data Byte1	Data Byte2
Note off	8x	Key number	Note Off velocity
Note on	9x	Key number	Note on velocity
Polyphonic Key Pressure	Ax	Key number	Amount of pressure
Control Change	Bx	Controller number	Controller value
Program Change	Cx	Program number	None
Channel Pressure	Dx	Pressure value	None
Pitch Bend	Ex	MSB	LSB

Notes: `x' in status byte hex value stands for a channel number.

Example: a Note On message is followed by two bytes, one to identify the note, and on to specify the velocity.

To play note number 80 with maximum velocity on channel 13, the MIDI device would send these three hexadecimal byte values: 9C 50 7F

A.2. Channel mode messages: - Channel mode messages are a special case of the Control Change message (Bx or 1011nnnn). The difference between a Control message and a Channel Mode message, which share the same status byte value, is in the first data byte. Data byte values 121 through 127 have been reserved in the Control Change message for the channel mode messages.

- Channel mode messages determine how an instrument will process MIDI voice messages.

1st Data Byte	Description	Meaning of 2nd Data Byte
79	Reset all controllers	None; set to 0
7A	Local control	0 = off; 127 = on
7B	All notes off	None; set to 0
7C	Omni mode off	None; set to 0
7D	Omni mode on	None; set to 0
7E	Mono mode on (Poly mode off)	**
7F	Poly mode on (Mono mode off)	None; set to 0

** if value = 0 then the number of channels used is determined by the receiver; all other values set a specific number of channels, beginning with the current basic channel.

B. System Messages:

- System messages carry information that is not channel specific, such as timing signal for synchronization, positioning information in pre-recorded MIDI sequences, and detailed setup information for the destination device.

B.1. System real-time messages:

- messages related to synchronization

System Real-Time Message	Status Byte
Timing Clock	F8
Start Sequence	FA
Continue Sequence	FB
Stop Sequence	FC
Active Sensing	FE
System Reset	FF

B.2. System common messages:

- contain the following unrelated messages

System Common Message	Status Byte	Number of Data Bytes
MIDI Timing Code	F1	1
Song Position Pointer	F2	2
Song Select	F3	1
Tune Request	F6	None

B.3. System exclusive message:

- (a) Messages related to things that cannot be standardized, (b) addition to the original MIDI specification.
- It is just a stream of bytes, all with their high bits set to 0, bracketed by a pair of system exclusive start and end messages (F0 and F7).

Dave Marshall
10/4/2001

General MIDI

- MIDI + Instrument Patch Map + Percussion Key Map -> a piece of MIDI music sounds the same anywhere it is played
  - Instrument patch map is a standard program list consisting of 128 patch types.
  - Percussion map specifies 47 percussion sounds.
  - Key-based percussion is always transmitted on MIDI channel 10.
- Requirements for General MIDI Compatibility:
  - Support all 16 channels.
  - Each channel can play a different instrument/program (multitimbral).
  - Each channel can play many voices (polyphony).
  - Minimum of 24 fully dynamically allocated voices.

Dave Marshall
10/4/2001

Next Up Previous
Next: Digital Audio and MIDI **Up:** General MIDI **Previous:** General MIDI

Additional MIDI Specifications

General MIDI Instrument Patch Map

Prog No.	Instrument	Prog No.	Instrument
(1-8	PIANO)	(9-16	CHROM PERCUSSION)
1	Acoustic Grand	9	Celesta
2	Bright Acoustic	10	Glockenspiel
3	Electric Grand	11	Music Box
4	Honky-Tonk	12	Vibraphone
5	Electric Piano 1	13	Marimba
6	Electric Piano 2	14	Xylophone
7	Harpsichord	15	Tubular Bells
8	Clav	16	Dulcimer
(17-24	ORGAN)	(25-32	GUITAR)
17	Drawbar Organ	25	Acoustic Guitar(nylon)
18	Percussive Organ	26	Acoustic Guitar(steel)
19	Rock Organ	27	Electric Guitar(jazz)
20	Church Organ	28	Electric Guitar(clean)
21	Reed Organ	29	Electric Guitar(muted)
22	Accoridan	30	Overdriven Guitar
23	Harmonica	31	Distortion Guitar
24	Tango Accordian	32	Guitar Harmonics
(33-40	BASS)	(41-48	STRINGS)
33	Acoustic Bass	41	Violin
34	Electric Bass(finger)	42	Viola
35	Electric Bass(pick)	43	Cello
36	Fretless Bass	44	Contrabass
37	Slap Bass 1	45	Tremolo Strings
38	Slap Bass 2	46	Pizzicato Strings
39	Synth Bass 1	47	Orchestral Strings
40	Synth Bass 2	48	Timpani
(49-56	ENSEMBLE)	(57-64	BRASS)
49	String Ensemble 1	57	Trumpet
50	String Ensemble 2	58	Trombone
51	SynthStrings 1	59	Tuba
52	SynthStrings 2	60	Muted Trumpet
53	Choir Aahs	61	French Horn
54	Voice Oohs	62	Brass Section
55	Synth Voice	63	SynthBrass 1
56	Orchestra Hit	64	SynthBrass 2
(65-72	REED)	(73-80	PIPE)
65	Soprano Sax	73	Piccolo
66	Alto Sax	74	Flute

67	Tenor Sax	75	Recorder
68	Baritone Sax	76	Pan Flute
69	Oboe	77	Blown Bottle
70	English Horn	78	Skakuhachi
71	Bassoon	79	Whistle
72	Clarinet	80	Ocarina
(81-88	SYNTH LEAD)	(89-96	SYNTH PAD)
81	Lead 1 (square)	89	Pad 1 (new age)
82	Lead 2 (sawtooth)	90	Pad 2 (warm)
83	Lead 3 (calliope)	91	Pad 3 (polysynth)
84	Lead 4 (chiff)	92	Pad 4 (choir)
85	Lead 5 (charang)	93	Pad 5 (bowed)
86	Lead 6 (voice)	94	Pad 6 (metallic)
87	Lead 7 (fifths)	95	Pad 7 (halo)
88	Lead 8 (bass+lead)	96	Pad 8 (sweep)
(97-104	SYNTH EFFECTS)	(105-112	ETHNIC)
97	FX 1 (rain)	105	Sitar
98	FX 2 (soundtrack)	106	Banjo
99	FX 3 (crystal)	107	Shamisen
100	FX 4 (atmosphere)	108	Koto
101	FX 5 (brightness)	109	Kalimba
102	FX 6 (goblins)	110	Bagpipe
103	FX 7 (echoes)	111	Fiddle
104	FX 8 (sci-fi)	112	Shanai
(113-120	PERCUSSIVE)	(121-128	SOUND EFFECTS)
113	Tinkle Bell	121	Guitar Fret Noise
114	Agogo	122	Breath Noise
115	Steel Drums	123	Seashore
116	Woodblock	124	Bird Tweet
117	Taiko Drum	125	Telephone Ring
118	Melodic Tom	126	Helicopter
119	Synth Drum	127	Applause
120	Reverse Cymbal	128	Gunshot

General MIDI Percussion Key Map

MIDI Key	Drum Sound	MIDI Key	Drum Sound
35	Acoustic Bass Drum	59	Ride Cymbal 2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low Floor Tom	65	High Timbale
42	Closed Hi-Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low Tom	69	Cabasa

46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle
49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Claves
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		

Dave Marshall
10/4/2001

Digital Audio and MIDI

There are many application os DIgital Audio and Midi being used together:

- Modern Recording Studio -- Hard Disk Recording and MIDI
  - Analog Sounds (Live Vocals, Guitar, Sax etc) -- DISK
  - Keyboards, Drums, Samples, Loops Effects -- MIDI
- Sound Generators: use a mix of
  - Synthesis
  - Samples
- Samplers -- Digitise (Sample) Sound then
  - Playback
  - Loop (beats)
  - Simulate Musical Instruments

Dave Marshall
10/4/2001

Digital Audio, Synthesis, Midi and Compression -- MPEG 4 Structured Audio

- We have seen the need for compression already in Digital Audio -- Large Data Files
- Basic Ideas of compression (see next Chapter) used as integral part of audio format -- MP3, real audio ***etc.***
- Mpeg-4 audio -- actually combines compression synthesis and midi to have a massive impact on compression.
- Midi, Synthesis encode what note to play and how to play it with a small number of parameters -- Much greater reduction than simply having some encoded bits of audio.
- Responsibility to create audio delegated to generation side.

Dave Marshall
10/4/2001

MPEG 4 Structured Audio

MPEG-4 covers the the whole range of digital audio:

- from very low bit rate speech
- to full bandwidth high quality audio
- built in anti-piracy measures
- ***Structured Audio***

Structured Audio Tools

MPEG-4 comprises of 6 ***Structured Audio tools*** are:

SAOL
, the Structured Audio Orchestra Language

SASL
, the Structured Audio Score Language

SASBF
, the Structured Audio Sample Bank Format

a set of MIDI semantics
which describes how to control SAOL with MIDI

a scheduler
, which describes how to take the above parts and create sound

the AudioBIFS
part of BIFS, which lets you make audio soundtracks in MPEG-4 using a variety of tools and effects-processing techniques

Very briefly each of the above tools have a specific function

SAOL (Structured Audio Orchestra Language)

SAOL is pronounced like the English word "sail" and is the central part of the Structured Audio toolset. It is a new software-synthesis language; it was specifically designed it for use in MPEG-4. You can think of SAOL as a language for describing synthesizers; a program, or instrument, in SAOL corresponds to the circuits on the inside of a particular hardware synthesizer.

SAOL is not based on any particular method of synthesis. It is general and flexible enough that any known method of synthesis can be described in SAOL. Examples of FM synthesis, physical-modeling synthesis, sampling synthesis, granular synthesis, subtractive synthesis, FOF synthesis, and hybrids of all of these in SAOL.

There's a page on using MIDI to control SAOL available at

http://sound.media.mit.edu/mpeg4-old/

SASL (Structured Audio Score Language)

SASL is a very simple language that was created for MPEG-4 to control the synthesizers specified by SAOL instruments. A SASL program, or score, contains instructions that tell SAOL what notes to play, how loud to play them, what tempo to play them at, how long they last, and how to control them (vary them while they're playing).

SASL is like MIDI in some ways, but doesn't suffer from MIDI's restrictions on temporal resolution or bandwidth. It also has a more sophisticated controller structure than MIDI; since in SAOL, you can write controllers to do anything, you need to be able to flexibly control them in SASL.

SASL is simpler (or more "lightweight") than many other score protocols. It doesn't have any facilities for looping, sections, repeats, expression evaluation, or some other things. Most SASL scores will be created by automatic tools, and so it's easy to make those tools map from the intent of the composer ("repeat this block") to the particular arrangement of events that implement the intent.

SASBF (Structured Audio Sample Bank Format)

SASBF (pronounces "sazz-biff"!!!) is a format for efficiently transmitting banks of sound samples to be used in wavetable, or sampling, synthesis. The format is being re-examined right now in hopes of making it at least partly compatible with the MIDI Downloaded Sounds (DLS) format.

The most active participants in this activity are E-Mu Systems and the MIDI Manufacturers Association (MMA).

MIDI Semantics

As well as controlling synthesis with SASL scripts, it can be controlled with MIDI files and scores in MPEG-4. MIDI is today's most commonly used representation for music score data, and many sophisticated authoring tools (such as sequencers) work with MIDI.

The MIDI syntax is external to the MPEG-4 Structured Audio standard; only references to the MIDI Manufacturers Association's definition in the standard. But in order to make the MIDI controls work right in the MPEG context, some semantics (what the instructions "mean") have been redefined in MPEG-4. The new semantics are carefully defined as part of the MPEG-4 specification.

Scheduler

The scheduler is the "guts" of the Structured Audio definition. It's a set of carefully defined and somewhat complicated instructions that specify how SAOL is used to create sound when it is driven by MIDI or SASL. It's in the style of "when this instruction arrives, you have to remember this, then execute this program, then do this other thing".

This component of Structured Audio is crucial but very dull unless you're a developer who wants to implement a SAOL system.

AudioBIFS

BIFS is the MPEG-4 ***Binary Format for Scene Description***. It's the component of MPEG-4 Systems which is used to describe how the different "objects" in a structured media scene fit together. To explain this a little more: in MPEG-4, the video clips, sounds, animations, and other pieces each have special formats to describe them. But to have something to show, we need to put the pieces together - the background goes in the back, this video clip attaches to the side of this "virtual TV" object, the sound should sound like it's coming from the speaker over there. BIFS lets you describe how to put the pieces together.

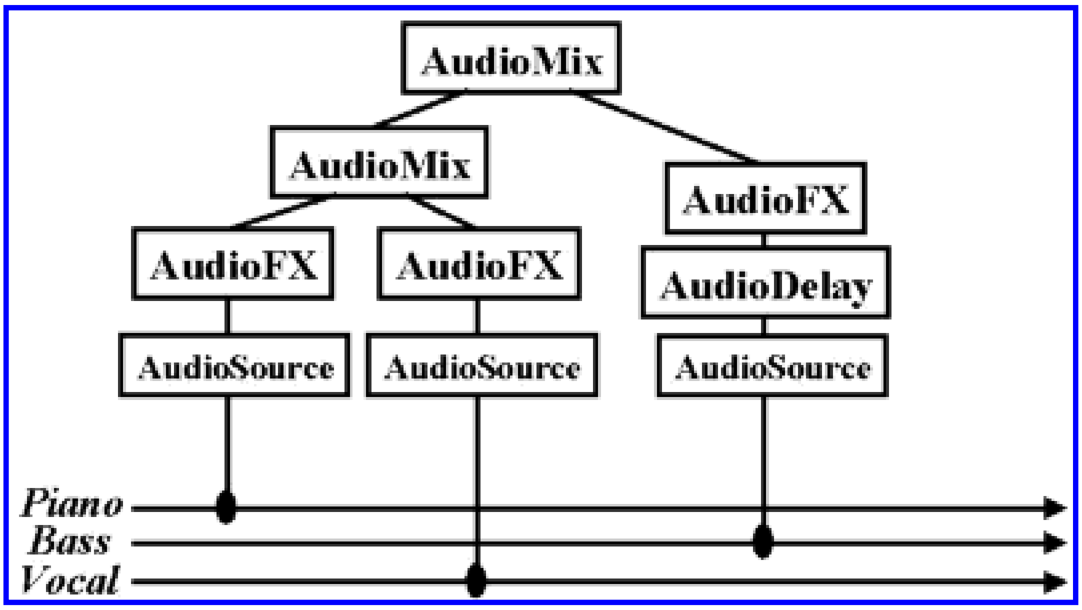
AudioBIFS is a major piece of MPEG-4 that has designed for specifying the mixing and post-production of audio scenes as they're played back. Using AudioBIFS, we can specify how the voice-track is mixed with the background music, and that it fades out after 10 seconds and this other music comes in and has a nice reverb on it.

BIFS is generally based on the Virtual Reality Modeling Language (VRML) (See Later in Course), but has extended capabilities for streaming and mixing audio and video data into a virtual-reality scene. The AudioBIFS functions are very advanced compared to VRML's sound model, which is rather simple, and are being tentatively considered for use in a future version of VRML.

In MPEG-4, AudioBIFS allows you to describe a sound as the combination of a number of sound objects. These sound objects may be coded using different coders (for example, CELP-coded voice and synthetic background music), and combined together in many ways. We can mix sounds together, or apply special filters and other processing functions written in SAOL.

Like the rest of BIFS, AudioBIFS is based on a scene graph. However, unlike in visual BIFS, the nodes in the AudioBIFS scene graph don't represent a bunch of objects which are presented to the user. Each AudioBIFS sound subgraph represents one sound object which is created by mixing and processing the elementary sound streams on which it is based.

For example, Fig 6.10 audio subgraph which shows how a simple sound is created from three elementary sound streams:



AudioBIFS Subgraph

Each of the rectangles show a node in the audio scene subgraph. Each node has a certain function, like mixing some sounds together, or delaying a sound, or doing some effects-processing. The arrows along the bottom represent the three elementary sound streams which make up the sound object. Each sound stream can be coded a different way. For example, we might code the piano sound with the Structured Audio decoder, the bass sound with the MPEG-4 Parametric HILN coder, and the vocal track with the MPEG-4 CELP coder.

These three sound streams are just like a "multitrack" recording of the final music sound object. The sound of each instrument is represented separately, then the scene graph mixes them all together. The processing in the audio subgraph is like a "data-flow" diagram. The sounds flow from the streams at the bottom, up through the nodes, and turn into a single sound at the top.

This single, final sound can be put into an audiovisual scene: it can be given a 3-D spatial location, moved around, and so on.

There's a page on AudioBIFS available at

http://www.risc.rockwell.com/349/343/MPEG4/

Dave Marshall
10/4/2001

Further Reading/Information for Digital Audio and Midi

Some good texts on these areas include:

- A programmer's Guide to Sound, T. Kientzle, Addison Wesley, 1997 (ISBN 0-201-41972-6)
- Audio on the Web -- The official IUMA Guide, Patterson and Melcher, Peachpit Press.
- The Art of Digital Audio, Watkinson, Focal/Butterworth-Heinmann.
- Synthesiser Basics, GPI Publications.
- Signal Processing: Principles and Applications, Brook and Wynne, Hodder and Stoughton.
- Digital Signal Processing, Oppenheim and Schafer, Prentice Hall.
- E. D. Scheirer, "Structured audio and effects processing in the MPEG-4 multimedia standard", ACM Multimedia Systems, in press.
- B. L. Vercoe, W. G. Gardner, E. D. Scheirer, "Structured Audio: Creation, transmission, and rendering of parametric sound representations", Proc. IEEE, in press.

Try some good sources for locating Digital Audio/internet sound/music materials at

- Digital Audio on the Web
- Audio File Formats
- Harmony Central: MIDI Tools and Resources - excellent resource. full documentation of the MIDI specification, guides to making a MIDI interface, links to development tools, keyboard specific resources and more.
- Exploring Midi. Excellent Midi/Audio Information Resource.
- MIDI Farm - hub for MIDI on the Web. Includes their own HTML newsletter.
- MIDIWeb - resources for the MIDI community.
- MIDIWorld

- Andre's MIDI Page - files and utilities.
- AWEsome MIDI - source of files, awe 32, awe 64 patches/utilities and related software and tools.
- Best of MIDI - dedicated to the definitions and standards of MIDI and of course,

MIDI files.

- Charles Belov's MIDI Tips
- Computer Karaoke - Karaoke on your computer
- Great MIDI Conspiracy - find out why MIDI sites are shutting down.
- Grinding Wall - MIDI Vault of the Forge - renditions and remakes of various genres of the underground.
- Günter Nagler's MIDI Utilities - MIDI converters with C++ source code, executable for PC (MSDOS and Windows95) midi format 1<->0 conversion, error check, file repairs, etc.
- Introduction to MIDI - wondering what MIDI is? Here you go.
- Jim's MIDI Links - arranged by artist (135+) in the following genres. rock, alternative, pop, r&b, rap, dance, hard rock, and metal.
- Jon's Amazing MIDI Quiz Thingy - guess the name of the show/film from the crappy MIDI version of it's title song.
- Macintosh MIDI User's Internet Guide
- MIDI Archive, The - documentation, synth patches, MIDI programmes and utilities, information on MIDI and much more.
- MIDI Business On-Line - source for midi files and information and resources related to the enjoyment and making of midi files.
- MIDI Connection - links and mailing lists concerning sounds and information files for synths and samplers.
- MIDI Explorer Search Engine - finds rare songs or organizes all versions of a popular tune.
- MIDI Guitar - Info & resources for guitar synthesizers.
- MIDI Land - discussion board.
- MIDI Mania - Betty's MIDI Page - original MIDI compositions and lots of interesting links to other MIDI sites.
- MIDI Music for Worship - A resource page for people using MIDI Music for worship services with or without musicians.
- MIDI Music Web Site - files, glossary, explanation of MIDI, tools, shareware, MPEG 3 information and much more.
- MIDI Technical Fanatic's Brainwashing Center - tutorials about MIDI and audio, technical/programming info, files, software, etc.
- MIDI Tips for Webmeisters - explains the realities of embedding MIDI files in a web page.
- MIDI Universe - a midi-seeking robot has been exploring the MIDI Universe. Universal questions such as the largest stellar sites, the most duplicated stars and more are answered.
- MIDI Zone - home of MidiSeek, a midi search engine. Also includes more files than we have space to mention.
- MIDI4U - place to expand your MIDI collection.
- MIDIbase - searchable database for midi and computer-music related websites. Hardware, software, events, education, events, midi-files, sounds, techno,

ambient, etc.

- MIDIs-R-Us - bands, cartoons, t.v., movies and much more.
- Moolly's MIDI Page - explains what MIDI is, techniques, connecting MIDI to a PC, and more.
- Name that MIDI - name five MIDI songs and have your name put up on the MIDI Hall of Fame board.
- Pianocorder to MIDI Project - Description of a project to digitally convert Pianocorder tapes into MIDI files.
- Project Beethoven - collection of MIDIs written without the use of hearing, written by various composers around the internet.
- TidBITS Guide to MIDI and the Macintosh
- Twin Cities MIDI - MIDI-users contact list, providing names and URL's of other MIDI users on the internet, as well as a special Macintosh MIDI section.
- Ultimate MIDI Page - includes explanatory information about midi as well as an extensive index of other midi sites.

MPEG 4 Pages:

- --- Main MPEG 4 Audio Page
- --- MPEG 4 Home Page
- --- MPEG FAQ's
- --- MPEG 4 Useful Links
- --- MPEG 4 Page

Dave Marshall
10/4/2001

Graphic/Image File Formats

This section introduces some of the most common graphics and image file formats. Some of them are restricted to particular hardware/operating system platforms, others are *cross-platform* independent formats. While not all formats are cross-platform, there are conversion applications that will recognize and translate formats from other systems.

The document (***http://www.cica.indiana.edu/graphics/image.formats.html***) by CICA at Indiana Univ. provides a fairly comprehensive listing of various formats.

Most image formats incorporate some variation of a *compression* technique due to the large storage size of image files. Compression techniques can be classified into either **lossless** or **lossy**. We will study various video and audio compression techniques in the Next Chapter.

Dave Marshall
10/4/2001

Graphic/Image Data Structures

A digital image consists of many picture elements, termed **pixels**. The number of pixels that compose a monitor image determine the quality of the image (**resolution**). Higher resolution always yields better quality.

A ***bit-map*** representation stores the graphic/image data in the same manner that the computer monitor contents are stored in video memory.

Dave Marshall
10/4/2001

Monochrome/Bit-Map Images

An example 1 bit monochrome image is illustrated in Fig. 6.11 where:

Sample Monochrome Bit-Map Image

- Each pixel is stored as a single bit (0 or 1)
- A 640 x 480 monochrome image requires 37.5 KB of storage.
- *Dithering* is often used for displaying monochrome images

Dave Marshall
10/4/2001

Next Up Previous
Next: 8-bit Colour Images **Up:** Graphic/Image Data Structures **Previous:** Monochrome/Bit-Map Images

Gray-scale Images

An example gray-scale image is illustrated in Fig. 6.12 where:

Example of a Gray-scale Bit-map Image

- Each pixel is usually stored as a byte (value between 0 to 255)
- A 640 x 480 greyscale image requires over 300 KB of storage.

Dave Marshall
10/4/2001

8-bit Colour Images

An example 8-bit colour image is illustrated in Fig. 6.13 where:

Example of 8-Bit Colour Image

- One byte for each pixel
- Supports 256 out of the millions s possible, acceptable colour quality
- Requires Colour Look-Up Tables (LUTs)
- A 640 x 480 8-bit colour image requires 307.2 KB of storage (the same as 8-bit greyscale)

Dave Marshall
10/4/2001

24-bit Colour Images

An example 24-bit colour image is illustrated in Fig. 6.14 where:

Example of 24-Bit Colour Image

- Each pixel is represented by three bytes (e.g., RGB)
- Supports 256 x 256 x 256 possible combined colours (16,777,216)
- A 640 x 480 24-bit colour image would require 921.6 KB of storage
- Most 24-bit images are 32-bit images, the extra byte of data for each pixel is used to store an *alpha* value representing special effect information

---

Dave Marshall
10/4/2001

Standard System Independent Formats

The following brief format descriptions are the most commonly used formats. Follow some of the document links for more descriptions.

- GIF (GIF87a, GIF89a)
- JPEG
- TIFF
- Graphics Animation Files
- Postscript/Encapsulated Postscript

Dave Marshall
10/4/2001

GIF (GIF87a, GIF89a)

- Graphics Interchange Format (GIF) devised by the UNISYS Corp. and Compuserve, initially for transmitting graphical images over phone lines via modems
- Uses the Lempel-Ziv Welch algorithm (a form of Huffman Coding), modified slightly for image scan line packets (line grouping of pixels)
- Limited to only 8-bit (256) colour images, suitable for images with few distinctive colours (e.g., graphics drawing)
- Supports *interlacing*

Dave Marshall
10/4/2001

JPEG

- A standard for photographic image compression created by the Joint Photographics Experts Group
- Takes advantage of limitations in the human vision system to achieve high rates of compression
- Lossy compression which allows user to set the desired level of quality/compression
- Detailed discussions in next chapter on compression.

Dave Marshall
10/4/2001

TIFF

- Tagged Image File Format (TIFF), stores many different types of images (e.g., monochrome, greyscale, 8-bit & 24-bit RGB, etc.) -> tagged
- Developed by the Aldus Corp. in the 1980's and later supported by the Microsoft
- TIFF is a lossless format (when not utilizing the new JPEG tag which allows for JPEG compression)
- It does not provide any major advantages over JPEG and is not as user-controllable it appears to be declining in popularity

---

Dave Marshall
10/4/2001

Graphics Animation Files

- FLC - main animation or moving picture file format, originally created by Animation Pro
- FLI - similar to FLC
- GL - better quality moving pictures, usually large file sizes

---

Dave Marshall
10/4/2001

Postscript/Encapsulated Postscript

- A typesetting language which includes text as well as vector/structured graphics and bit-mapped images
- Used in several popular graphics programs (Illustrator, FreeHand)
- Does not provide compression, files are often large

Dave Marshall
10/4/2001

System Dependent Formats

Many graphical/imaging applications create their own file format particular to the systems they are executed upon. The following are a few popular system dependent formats:

Dave Marshall
10/4/2001

Microsoft Windows: BMP

- A system standard graphics file format for Microsoft Windows
- Used in PC Paintbrush and other programs
- It is capable of storing 24-bit bitmap images

Dave Marshall
10/4/2001

Macintosh: PAINT and PICT

- PAINT was originally used in MacPaint program, initially only for 1-bit monochrome images.
- PICT format is used in MacDraw (a vector based drawing program) for storing structured graphics

---

Dave Marshall
10/4/2001

X-windows: XBM

- Primary graphics format for the X Window system
- Supports 24-bit colour bitmap
- Many public domain graphic editors, e.g., *xv*
- Used in X Windows for storing icons, pixmaps, backdrops, etc.

Dave Marshall
10/4/2001

Further Reading/Information

- Intro. to Computer Picturesfrom Allison Zhang at the School of Library and Information Studies, Dalhousie University, Halifax, N.S., Canada
- Image File Formats

Dave Marshall
10/4/2001

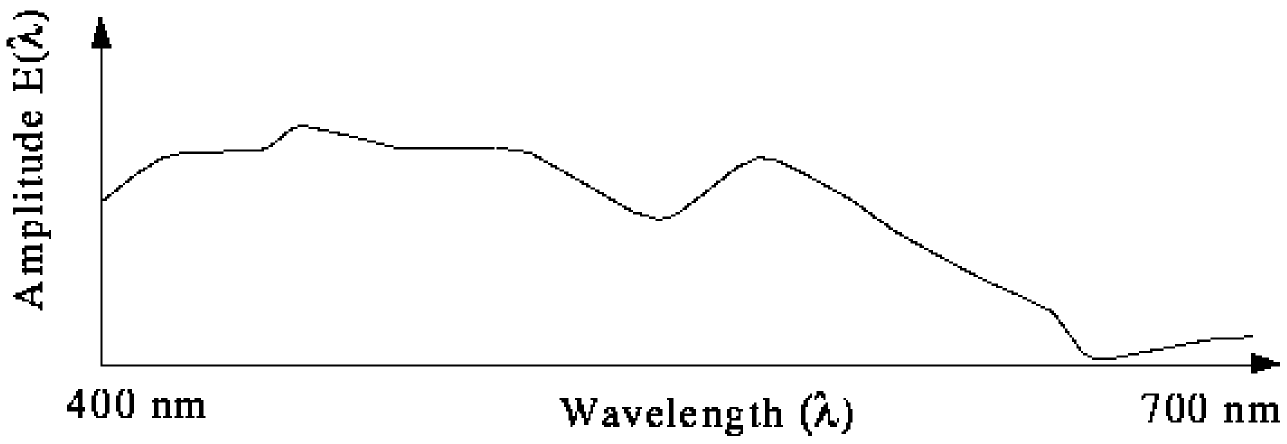
Colour in Image and Video

Dave Marshall
10/4/2001

Basics of Colour

Light and Spectra

- Visible light is an electromagnetic wave in the 400nm - 700 nm range.
- Most light we see is not one wavelength, it's a combination of many wavelengths (Fig. 6.15).



Light Wavelengths

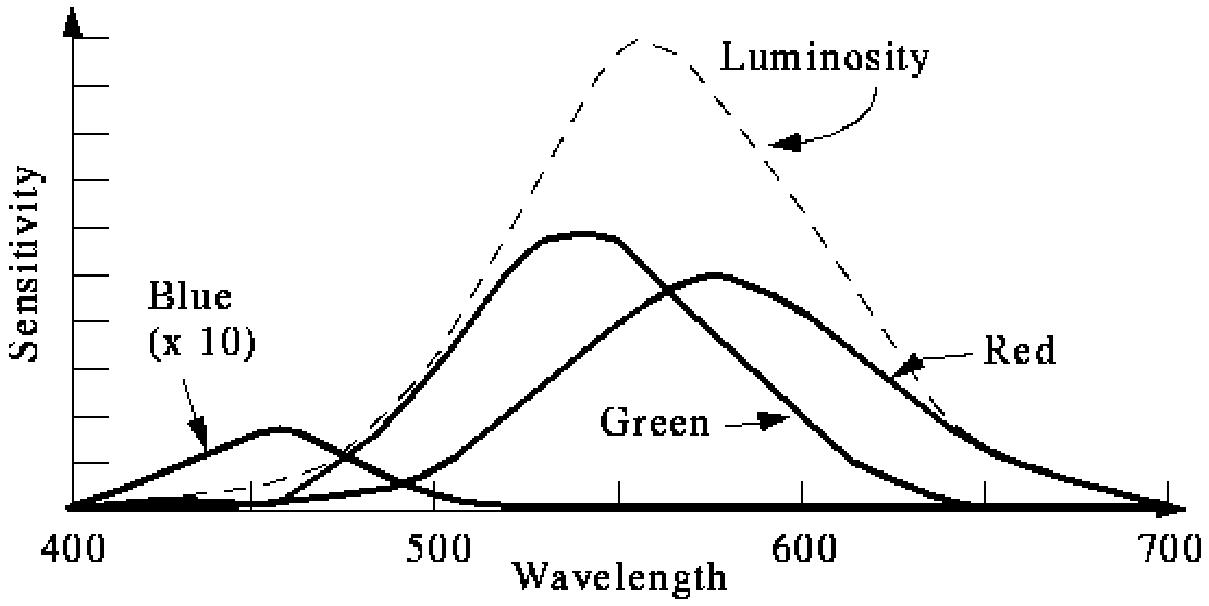
- The profile above is called a *spectra*.

The Human Retina

- The eye is basically just a camera
- Each neuron is either a *rod* or a *cone*. Rods are not sensitive to colour.

Cones and Perception

- Cones come in 3 types: red, green and blue. Each responds differently to various frequencies of light. The following figure shows the spectral-response functions of the cones and the luminous-efficiency function of the human eye (Fig. 6.16.

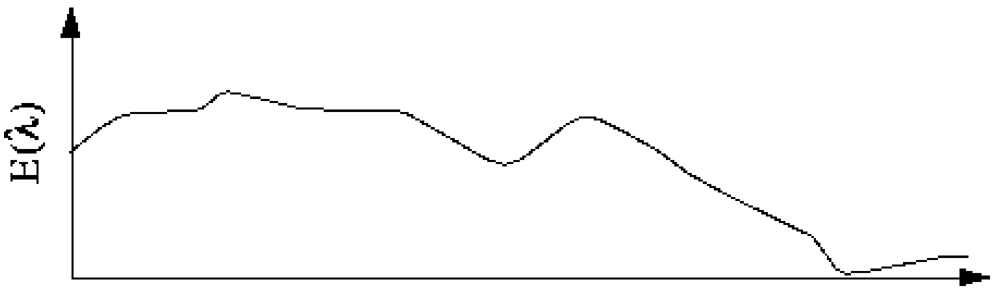


Cones and Luminous-efficiency Function of the Human Eye

- The profile above is called a *spectra*.

- The colour signal to the brain comes from the response of the 3 cones to the spectra being observed (Fig 6.17). That is, the signal consists of 3 numbers:

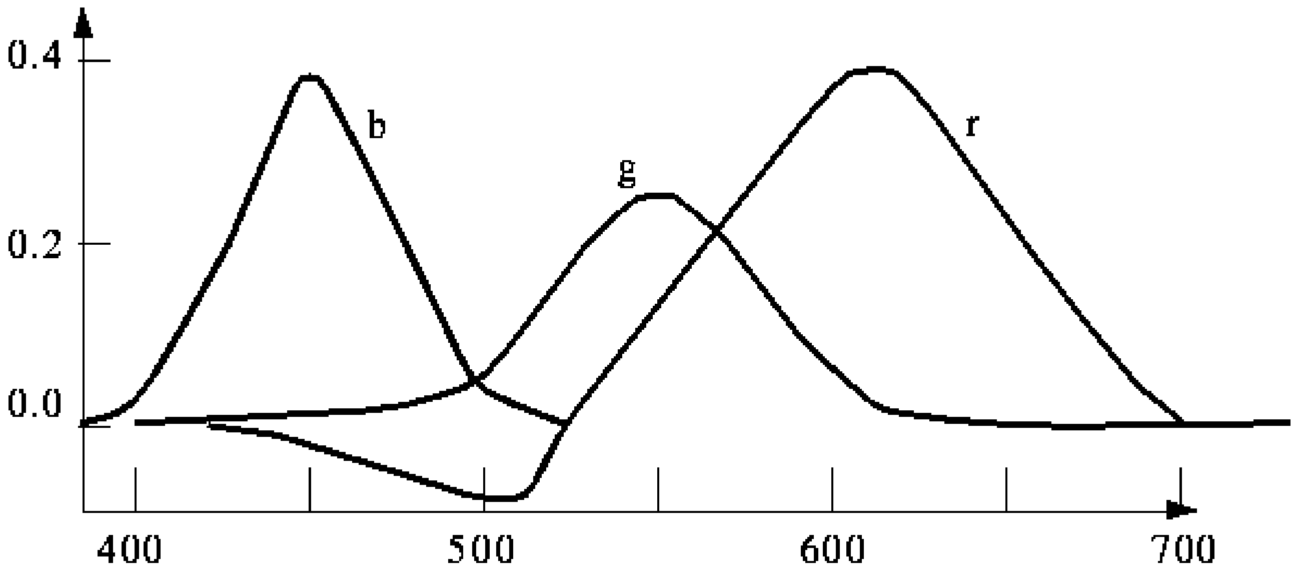
$$R = \int E(\lambda)\, S_R(\lambda)\, d\lambda$$



where *E* is the light and *S* are the sensitivity functions.

- A colour can be specified as the sum of three colours. So colours form a 3 dimensional vector space.

- The following figure shows the amounts of three primaries needed to match all the wavelengths of the visible spectrum (Fig. refspectrum).



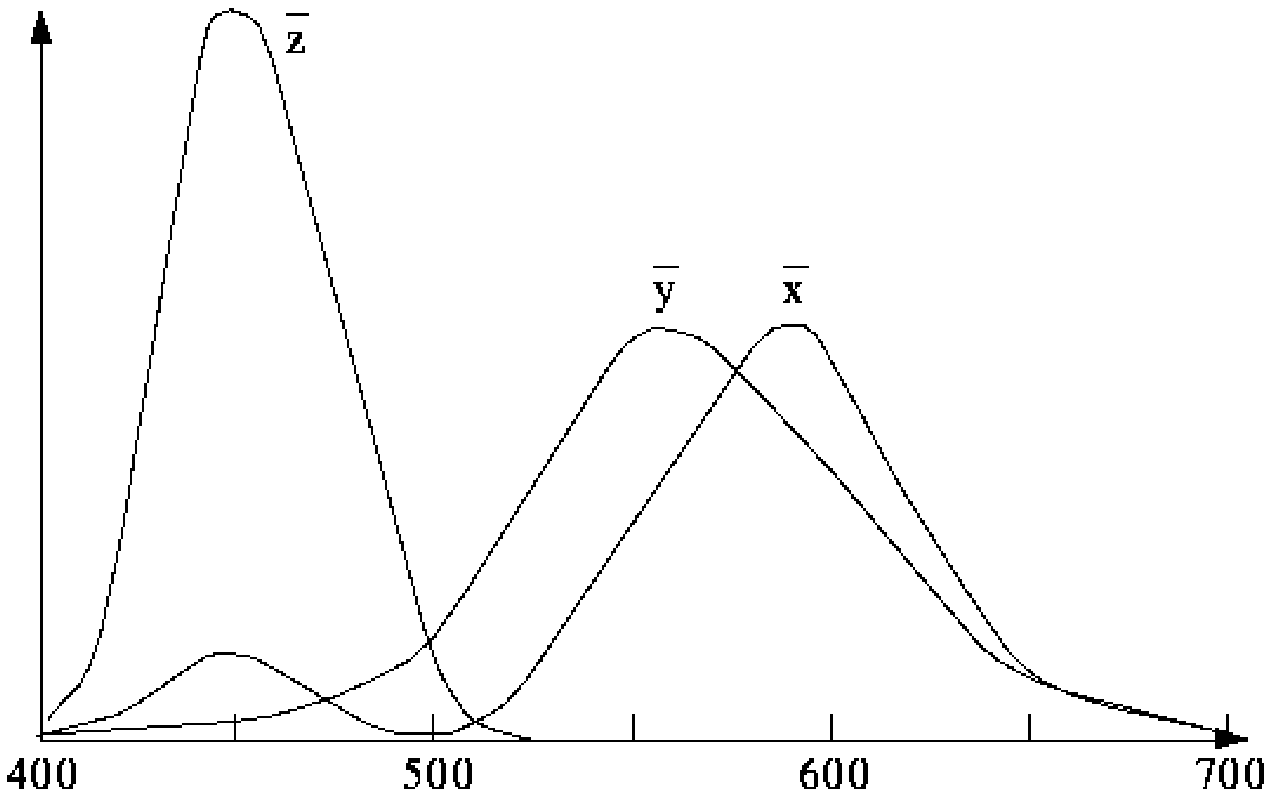
Wavelengths of the Visible Spectrum

- The negative value indicates that some colours cannot be exactly produced by adding up the primaries.

CIE Chromaticity Diagram

Does a set of primaries exist that span the space with only positive coefficients?

- Yes, but no pure colours.
- In 1931, the CIE defined three standard primaries **(X, Y, Z)** . The **Y** primary was intentionally chosen to be identical to the luminous-efficiency function of human eyes.



Reproducing Visible Colour

- Figure 6.19 shows the amounts of X, Y, Z needed to exactly reproduce any visible colour via the formulae:

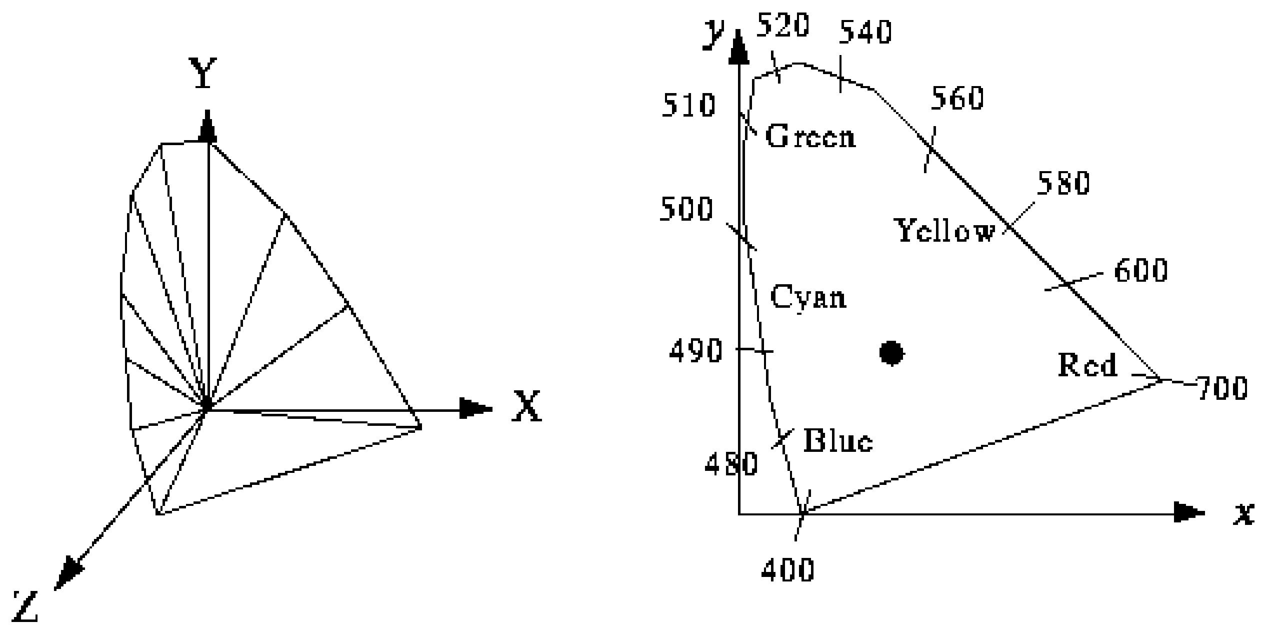
$$X = \int E(\lambda)\,\bar{x}(\lambda)\,d\lambda$$

$$Y = \int E(\lambda)\,\bar{y}(\lambda)\,d\lambda$$

$$Z = \int E(\lambda)\,\bar{z}(\lambda)\,d\lambda$$

- All visible colours are in a ***horseshoe*** shaped cone in the X-Y-Z space. Consider the plane

X+Y+Z=1 and project it onto the X-Y plane, we get the *CIE chromaticity diagram* as shown in Fig. 6.20.



CIE Chromaticity Diagram

- The edges represent the ***pure*** colours (sine waves at the appropriate frequency)
- White (a blackbody radiating at 6447 kelvin) is at the ***dot***
- When added, any two colours (points on the CIE diagram) produce a point on the line between them.

L*a*b (Lab) Colour Model

- A refined CIE model, named CIE L*a*b in 1976
- Luminance: L Chrominance: a - ranges from green to red, b - ranges from blue to yellow (Fig, 6.21)

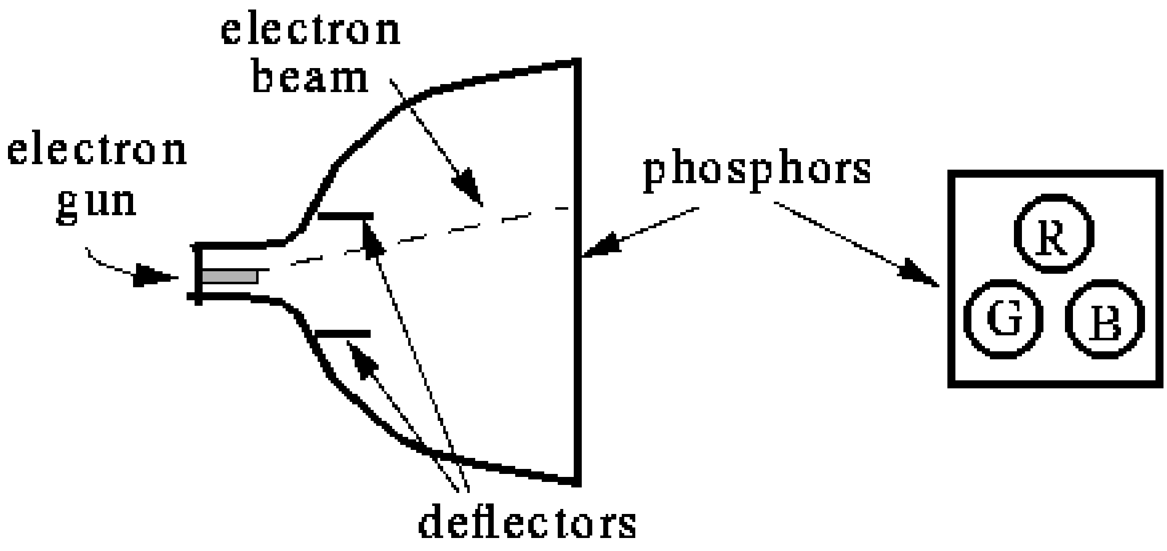
LAB Colour Model

- Used by *Photoshop*

- CRT Displays

CRT Displays

- CRT displays have three phosphors (RGB) which produce a combination of wavelengths when excited with electrons (Fig. 6.22).



RGB Colour Display

- The *gamut* of colours is all colours that can be reproduced using the three primaries
- The gamut of an colour monitor is smaller than the CIE (LAB) colour gamut on the CIE diagram.

Colour Image and Video Representations

- A black and white image is a 2-D array of integers.
- A colour image is a 2-D array of (R,G,B) integer triplets. These triplets encode how much the corresponding phosphor should be excited in devices such as a monitor.
- Example is shown in Fig 6.23.

  Display of a Colour Cube

Beside the RGB representation, YIQ and YUV are the two commonly used in video.

YIQ Colour Space

- YIQ is used in colour TV broadcasting, it is downward compatible with B/W TV.
- Y (luminance) is the CIE Y primary.

  $Y = 0.299R + 0.587G + 0.114B$

- the other two vectors:

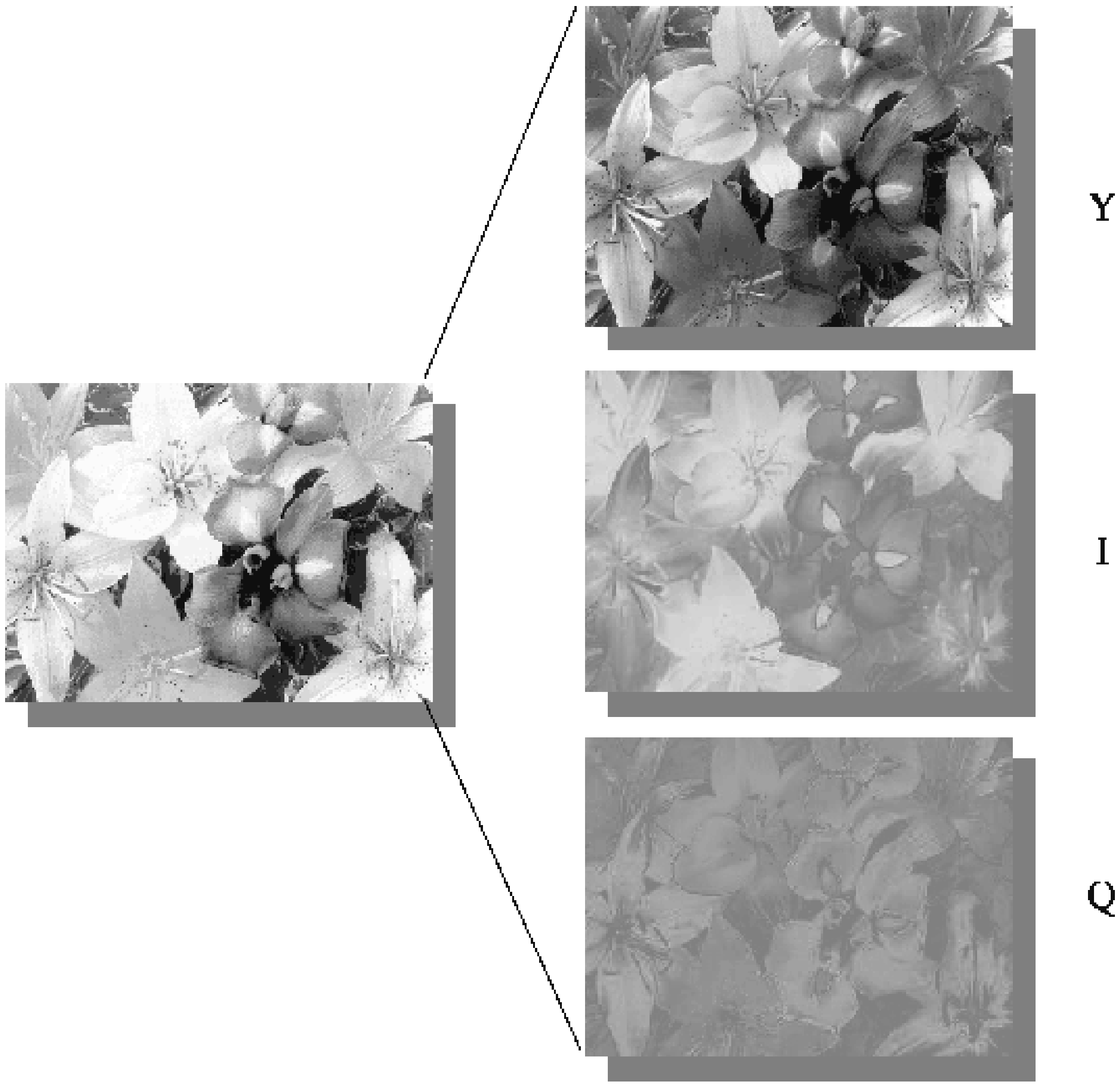
  $I = 0.596R - 0.275G - 0.321B$ $Q = 0.212R - 0.528G + 0.311B$

- The YIQ transform:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  YIQ Transform of a colour image

- I is red-orange axis, Q is roughly orthogonal to I.
- Eye is most sensitive to Y, next to I, next to Q. In NTSC, 4 MHz is allocated to Y, 1.5 MHz to I, 0.6 MHz to Q.
- An Example YIQ Decomposition is shown in Fig. 6.24.



Example YIQ Decomposition

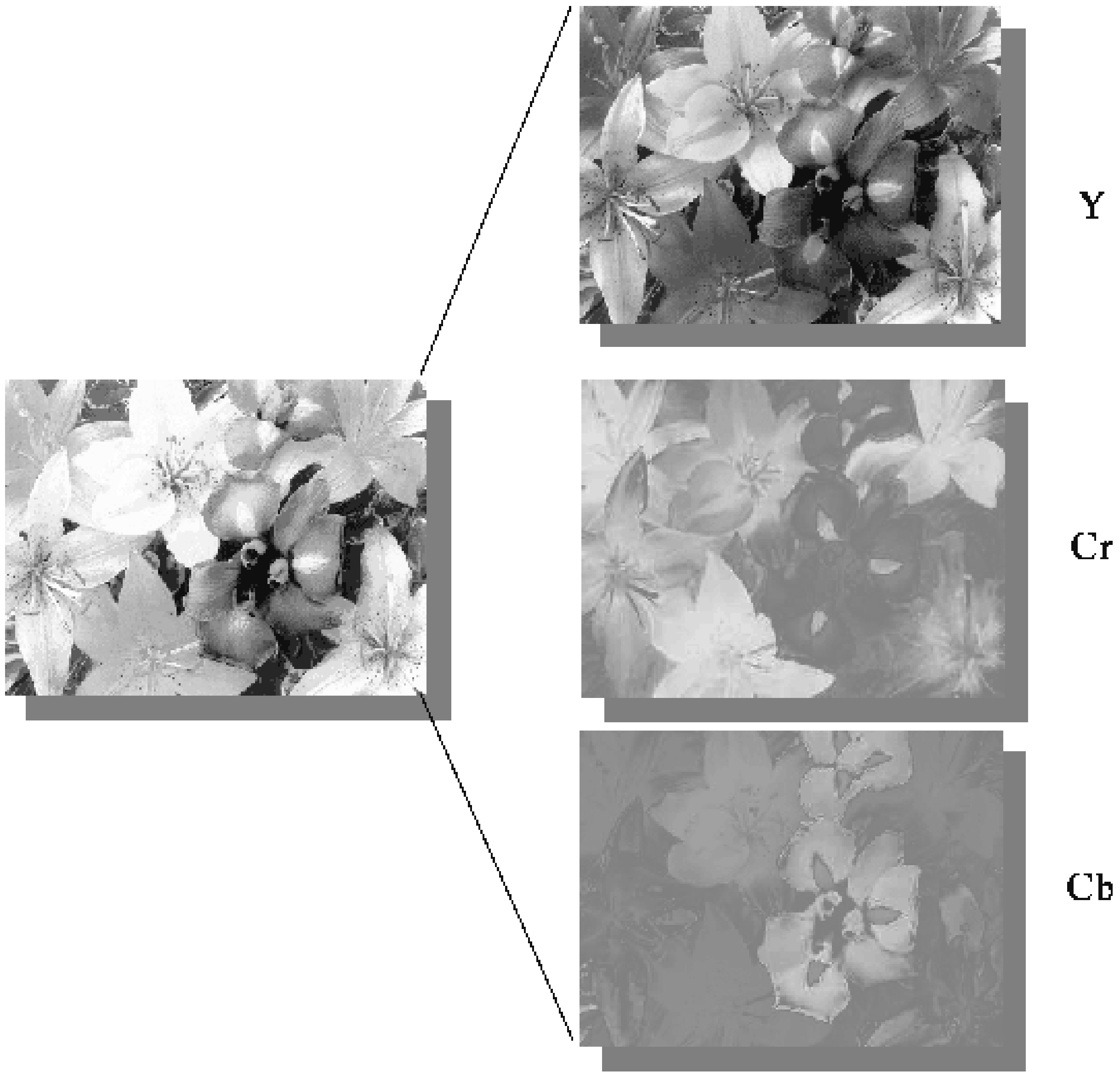
CCIR 601 (YUV)

- Established in 1982 to build digital video standard
- Video is represented by a sequence of fields (odd and even lines). Two fields make a frame.
- Works in PAL (50 fields/sec) or NTSC (60 fields/sec)
- Uses the Y, Cr, Cb colour space (also called YUV) *Y = 0.299R + 0.587G + 0.114B Cr = R - Y Cb = B - Y*
- The YCrCb (YUV) Transform:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- CCIR 601 also defines other image parameters, e.g. for NTSC, Luminance (Y) image size = 720 x 243 at 60 fields per second Chrominance image size = 360 x 243 at 60 fields per second

- An example YCrCb Decomposition is shown in Fig. 6.25

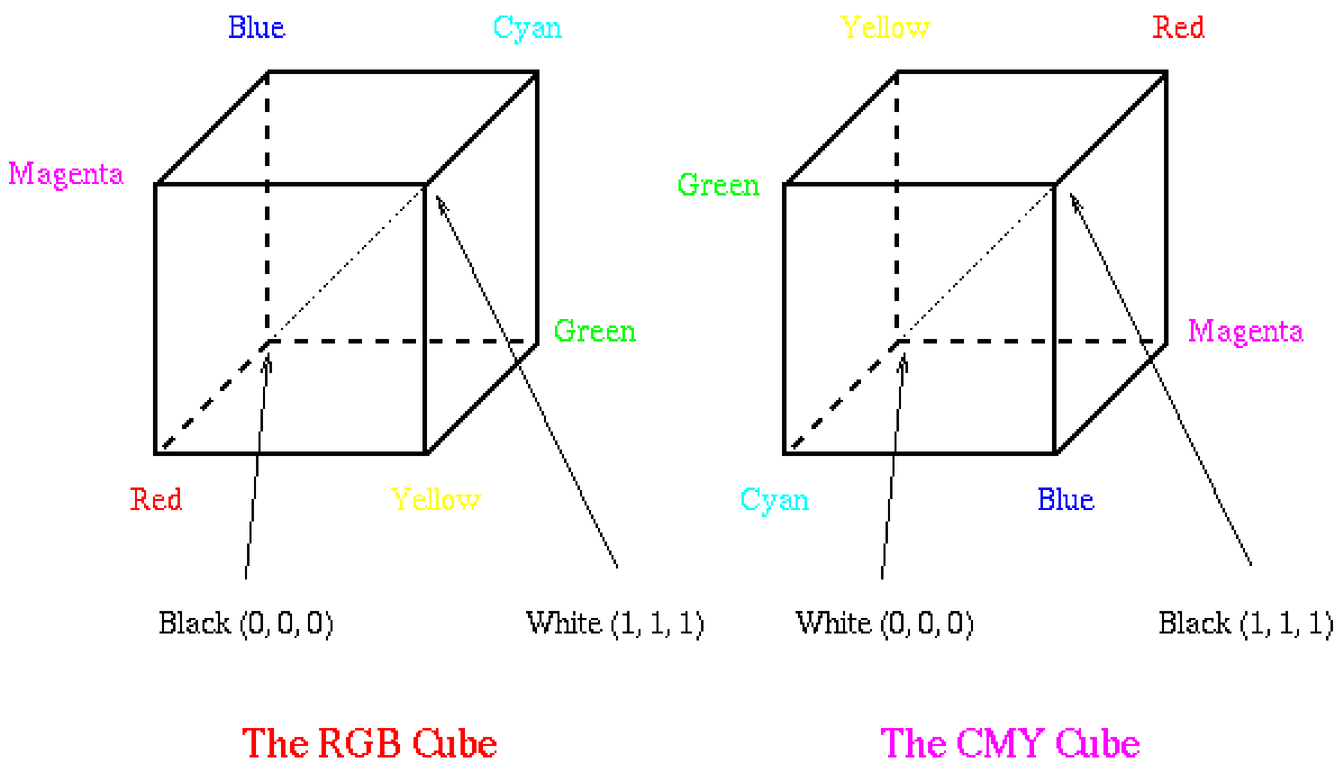


YCrCb Decomposition of a colour image

The CMY Colour Model

- Cyan, Magenta, and Yellow (CMY) are complementary colours of RGB (Fig. 6.26). They can be used as *Subtractive Primaries*.

- CMY model is mostly used in printing devices where the colour pigments on the paper absorb certain colours (e.g., no red light reflected from cyan ink).



The RGB and CMY Cubes

- Conversion between RGB and CMY:

Conversion between RGB and CMY:

- e.g., convert **White** from (1, 1, 1) in RGB to (0, 0, 0) in CMY.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

- Sometimes, an alternative CMYK model (K stands for *Black*) is used in colour printing (e.g., to produce darker black than simply mixing CMY). where

Summary of Colour

- Colour images are encoded as triplets of values.
- Three common systems of encoding in video are RGB, YIQ, and YCrCb.
- Besides the hardware-oriented colour models (i.e., RGB, CMY, YIQ, YUV), HSB (Hue, Saturation, and Brightness, e.g., used in Photoshop) and HLS (Hue, Lightness, and Saturation) are also commonly used.
- YIQ uses properties of the human eye to prioritize information. Y is the black and white (luminance) image, I and Q are the colour (chrominance) images. YUV uses similar idea.
- CCIR 601 is a standard for digital video that specifies image size, and decimates the chrominance images (for 4:2:2 video).

Basics of Video

Types of Colour Video Signals

- **Component video** - each primary is sent as a separate video signal.
    - The primaries can either be RGB or a luminance-chrominance transformation of them (e.g., YIQ, YUV).
    - Best colour reproduction
    - Requires more bandwidth and good synchronization of the three components
- **Composite video** - colour (chrominance) and luminance signals are mixed into a single carrier wave. Some interference between the two signals is inevitable.
- **S-Video** (Separated video, e.g., in S-VHS) - a compromise between component analog video and the composite video. It uses two lines, one for luminance and another for composite chrominance signal.

Analog Video

The following figures (Fig. 6.27 and 6.28) are from A.M. Tekalp, ***Digital video processing***, Prentice Hall PTR, 1995.

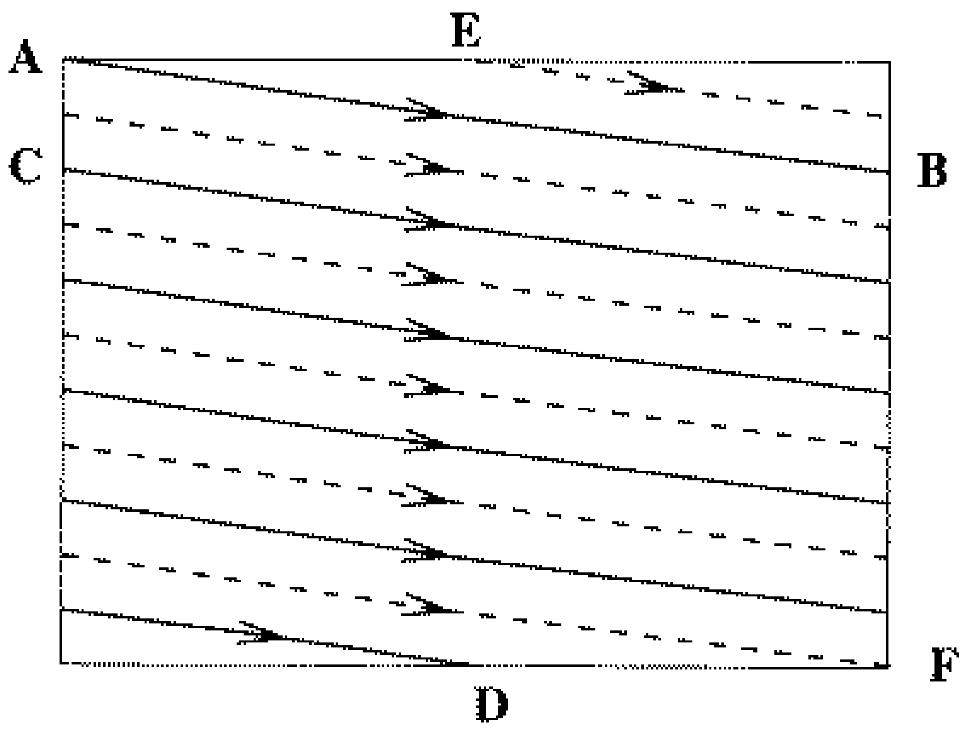
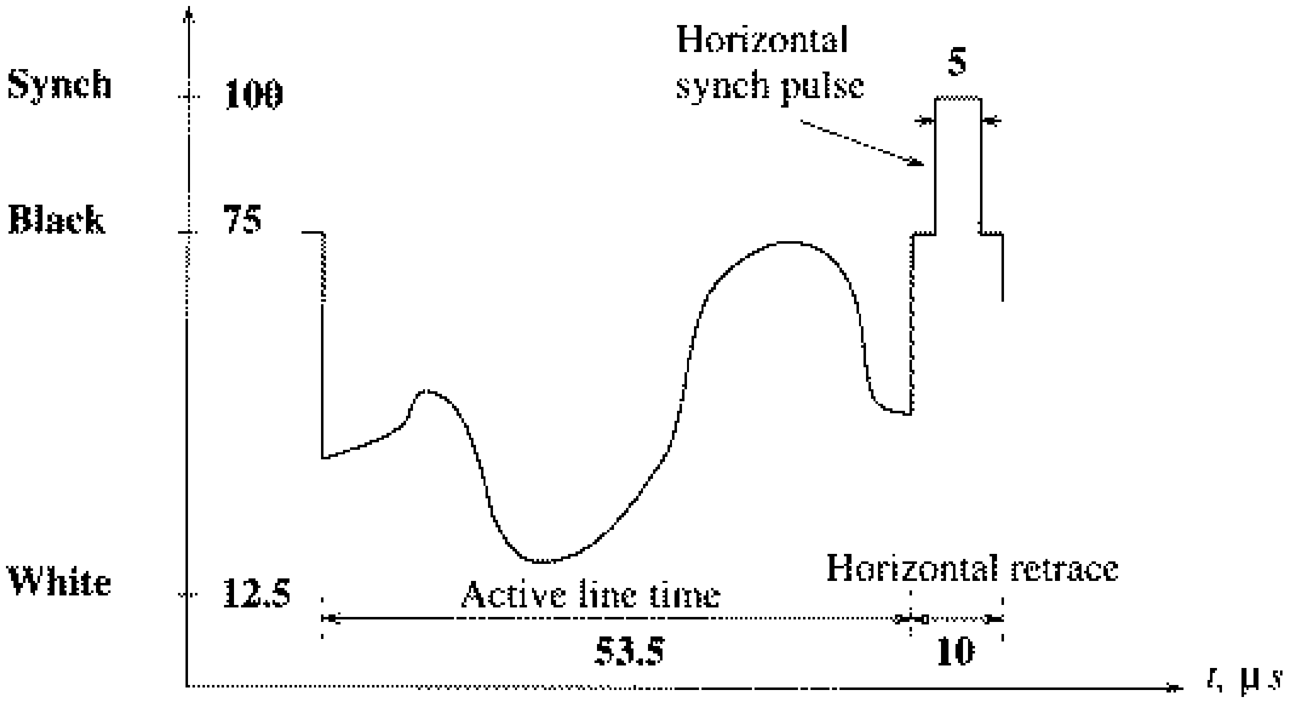


Figure 1.1: Scanning raster.

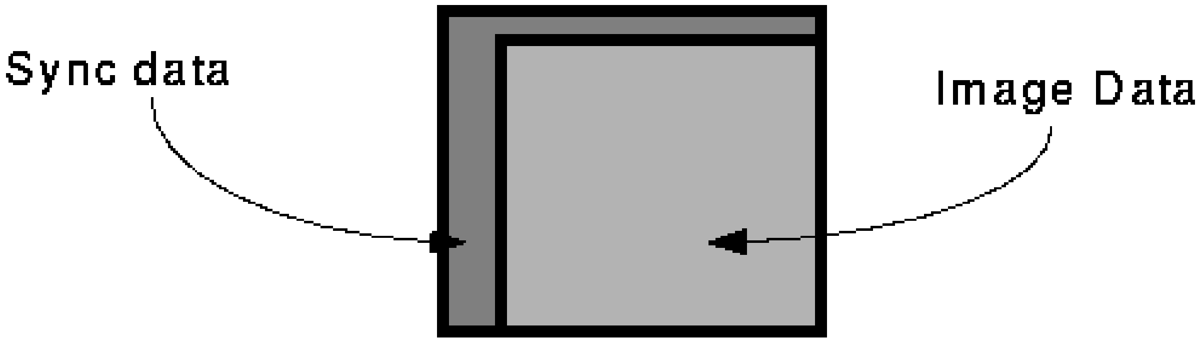
Raster Scanning



NTSC Signal

NTSC Video

- 525 scan lines per frame, 30 frames per second (or be exact, 29.97 fps, 33.37 msec/frame)
- Aspect ratio 4:3
- Interlaced, each frame is divided into 2 fields, 262.5 lines/field
- 20 lines reserved for control information at the beginning of each field (Fig. 6.29)
    - So a maximum of 485 lines of visible data
    - Laserdisc and S-VHS have actual resolution of 420 lines
    - Ordinary TV - 320 lines
- Each line takes 63.5 microseconds to scan. Horizontal retrace takes 10 microseconds (with 5 microseconds horizontal synch pulse embedded), so the active line time is 53.5 microseconds.



Digital Video Rasters

- Colour representation:

    - NTSC uses YIQ colour model.
    - composite = Y + I *cos*(Fsc t) + Q *sin*(Fsc t), where Fsc is the frequency of colour subcarrier
    - Eye is most sensitive to Y, next to I, next to Q. In NTSC, 4 MHz is allocated to Y, 1.5 MHz to I, 0.6 MHz to Q.

PAL Video

- 625 scan lines per frame, 25 frames per second (40 msec/frame)

- Aspect ratio 4:3

- Interlaced, each frame is divided into 2 fields, 312.5 lines/field

- Colour representation:
    - PAL uses YUV (YCbCr) colour model
    - composite = Y + 0.492 x U *sin*(Fsc t) + 0.877 x V *cos*(Fsc t)
    - In component analog video, U and V signals are lowpass filtered to about half the bandwidth of Y.

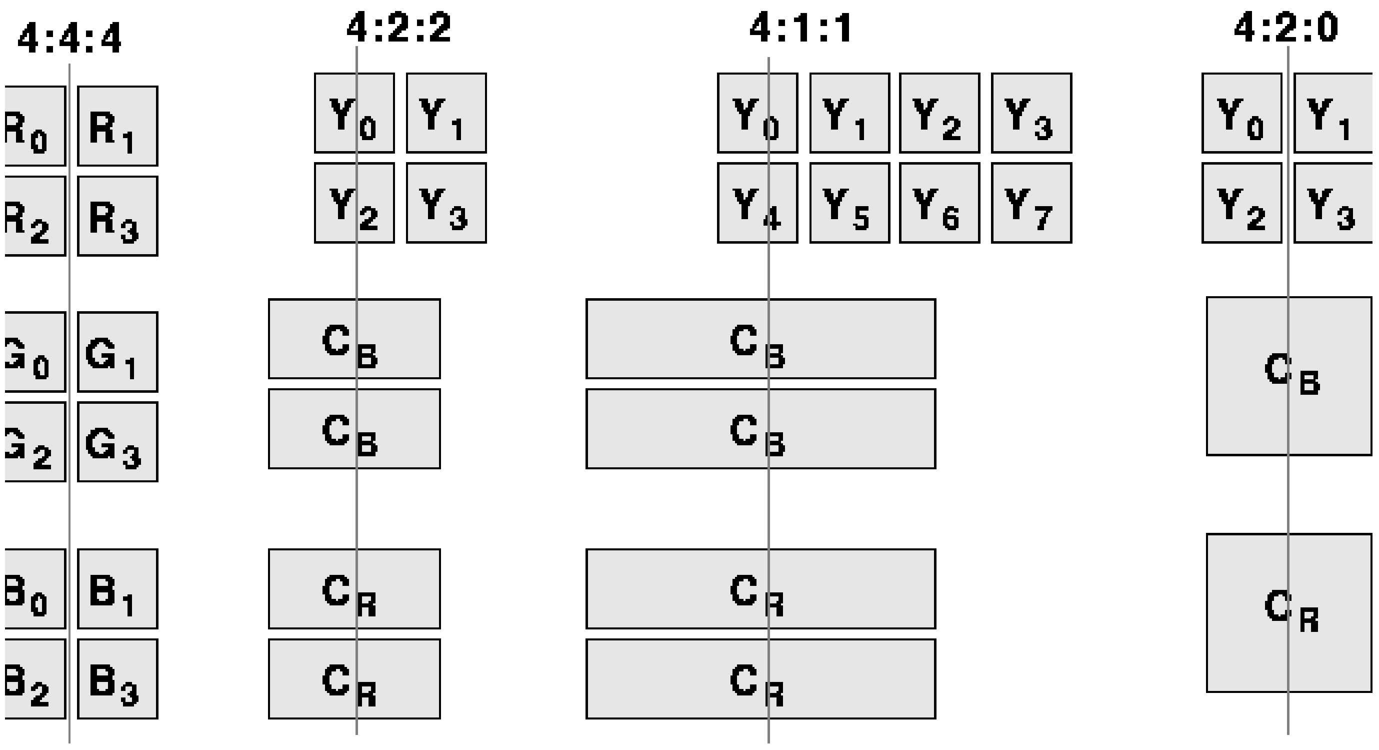
Next Up Previous
Next: Chroma Subsampling **Up:** Basics of Video **Previous:** Analog Video

Digital Video

- **Advantages:**
  - Direct random access -> good for nonlinear video editing
  - No problem for repeated recording
  - No need for blanking and sync pulse
- Almost all digital video uses component video

Chroma Subsampling

- How to decimate for chrominance (Fig. 6.30)?



Chroma Subsampling

- 4:2:2 -> Horizontally subsampled colour signals by a factor of 2. Each pixel is two bytes, e.g., (Cb0, Y0)(Cr0, Y1)(Cb2, Y2)(Cr2, Y3)(Cb4, Y4) ...
- 4:1:1 -> Horizontally subsampled by a factor of 4
- 4:2:0 -> Subsampled in both the horizontal and vertical axes by a factor of 2 between pixels as shown in the Fig. 6.30.
- 4:1:1 and 4:2:0 are mostly used in JPEG and MPEG (see Chapter 4).

Next Up Previous
Next: ATSC Digital Television Standard **Up:** Chroma Subsampling **Previous:** Chroma Subsampling

CCIR Standards for Digital Video

(CCIR - Consultative Committee for International Radio)

	CCIR 601	CCIR 601	CIF	QCIF
	525/60	625/50		
		NTSC	PAL/SECAM	NTSC
Luminance resolution	720 x 485	720 x 576	352 x 240	176 x 120
Chrominance resolut.	360 x 485	360 x 576	176 x 120	88 x 60
Colour Subsampling	4:2:2	4:2:2		
Fields/sec	60	50	30	30
Interlacing	Yes	Yes	No	No

- CCIR 601 uses interlaced scan, so each field only has half as much vertical resolution (e.g., 243 lines in NTSC). The CCIR 601 (NTSC) data rate is 165 Mbps.
- CIF (Common Intermediate Format) is introduced to as an acceptable temporary standard. It delivers about the VHS quality. CIF uses progressive (non-interlaced) scan.

ATSC Digital Television Standard

(ATSC - Advanced Television Systems Committee) The **ATSC Digital Television Standard** was recommended to be adopted as the Advanced TV broadcasting standard by the FCC Advisory Committee on Advanced Television Service on November 28, 1995. It covers the standard for **HDTV** (High Definition TV).

Video Format

The video scanning formats supported by the ATSC Digital Television Standard are shown in the following table.

Vertical Lines	Horizontal Pixels	Aspect Ratio	Picture Rate
1080	1920	16:9	60I 30P 24P
720	1280	16:9	60P 30P 24P
480	704	16:9 & 4:3	60I 60P 30P 24P
480	640	4:3	60I 60P 30P 24P

- The aspect ratio for HDTV is 16:9 as opposed to 4:3 in NTSC, PAL, and SECAM. (A 33% increase in horizontal dimension.)
- In the picture rate column, the &quot;I&quot; means interlaced scan, and the &quot;P&quot; means progressive (non-interlaced) scan.
- Both NTSC rates and integer rates are supported (i.e., 60.00, 59.94, 30.00, 29.97, 24.00, and 23.98).

Further Reading/Information

A good text on this area is:

Digital video processing, A.M. Tekalp, Prentice Hall PTR, 1995.

Interesting Web sites include:

- Homepage of the Advanced Television Systems Committee (ATSC)
- MHEG Home Page
- HDTV Web Page.

Video and Audio Compression

Video and Audio files are very large beasts. Unless we develop and maintain very high bandwidth networks (Gigabytes per second or more) we have to compress to data.

Relying on higher bandwidths is not a good option -- M25 Syndrome: Traffic needs ever increases and will adapt to swamp current limit whatever this is.

As we will compression becomes part of the representation or ***coding*** scheme which have become popular audio, image and video formats.

We will first study basic compression algorithms and then go on to study some actual coding formats.

Classifying Compression Algorithms

We can classify compression by the why it employs redundancy or by the method it compresses the data.

What is Compression?

Compression basically employs redundancy in the data:

- Temporal -- in 1D data, 1D signals, Audio etc.
- Spatial -- correlation between neighbouring pixels or data items
- Spectral -- correlation between colour or luminescence components. This uses the frequency domain to exploit relationships between frequency of change in data.
- psycho-visual -- exploit perceptual properties of the human visual system.

Compression can be categorised in two broad ways:

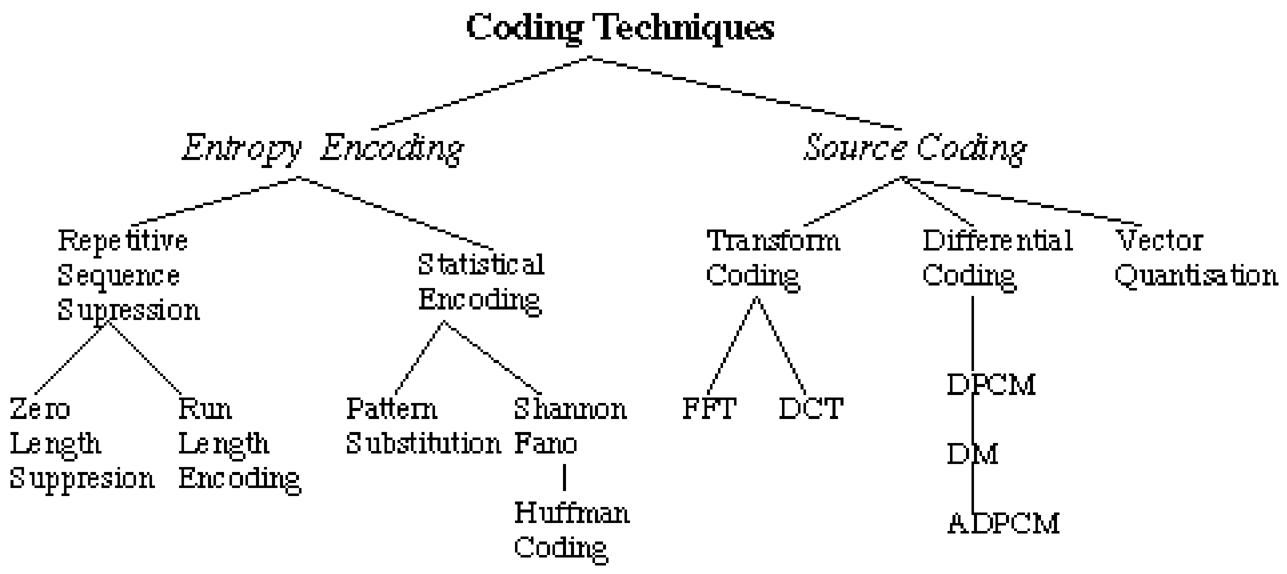
Lossless Compression
-- where data is compressed and can be reconstituted (uncompressed) without loss of detail or information. These are referred to as bit-preserving or reversible compression systems also.

Lossy Compression
-- where the aim is to obtain the best possible ***fidelity*** for a given bit-rate or minimizing the bit-rate to achieve a given fidelity measure. Video and audio compression techniques are most suited to this form of compression.

If an image is compressed it clearly needs to uncompressed (decoded) before it can viewed/listened to. Some processing of data may be possible in encoded form however.

Lossless compression frequently involves some form of ***entropy encoding*** and are based in information theoretic techniques (Fig. 7.1)

Lossy compression use source encoding techniques that may involve transform encoding, differential encoding or vector quantisation (Fig. 7.1).



Classification of Coding Techniques

We now address common coding methods of each type in turn:

Lossless Compression Algorithms (Repetitive Sequence Suppression)

These methods are fairly straight forward to understand and implement. Their simplicity is their downfall in terms of attaining the best compression ratios. However, the methods have their applications, as mentioned below:

Simple Repetition Suppresion

If in a sequence a series on *n* successive tokens appears we can replace these with a token and a count number of occurences. We usually need to have a special ***flag*** to denote when the repated token appears

For Example

```
89400000000000000000000000000000000
```

we can replace with

```
894f32
```

where `f` is the flag for zero.

Compression savings depend on the content of the data.

Applications of this simple compression technique include:

- Suppression of zero's in a file (***Zero Length Supression***)
    - Silence in audio data, Pauses in conversation ***etc.***
    - Bitmaps
    - Blanks in text or program source files
    - Backgrounds in images
- other regular image or data tokens

Run-length Encoding

This encoding method is frequently applied to images (or pixels in a scan line). It is a small compression component used in JPEG compression (Section 7.6).

In this instance, sequences of image elements $X_1, X_2, \dots, X_n$ are mapped to pairs $(c_1, l_1), (c_2, l_2), \dots, (c_n, l_n)$ where c_i represent image intensity or colour and l_i the length of the *i*th run of pixels (Not dissimilar to zero length supression above).

For example:

Original Sequence:

```
111122233333311112222
```

can be encoded as:

```
(1,4),(2,3),(3,6),(1,4),(2,4)
```

The savings are dependent on the data. In the worst case (Random Noise) encoding is more heavy than original file: 2*integer rather 1* integer if data is represented as integers.

Lossless Compression Algorithms (Pattern Substitution)

This is a simple form of statistical encoding.

Here we substitue a frequently repeating pattern(s) with a code. The code is shorter than than pattern giving us compression.

A simple Pattern Substitution scheme could employ predefined code (for example replace all occurrences of `The' with the code '&').

More typically tokens are assigned to according to frequency of occurrenc of patterns:

- Count occurrence of tokens
- Sort in Descending order
- Assign some symbols to highest count tokens

A predefined symbol table may used ie assign code *i* to token *i*.

However, it is more usual to dynamically assign codes to tokens. The entropy encoding schemes below basically attempt to decide the optimum assignment of codes to achieve the best compression.

Lossless Compression Algorithms (Entropy Encoding)

Lossless compression frequently involves some form of ***entropy encoding*** and are based in information theoretic techniques, Shannon is father of information theory and we briefly summarise information theory below before looking at specific entropy encoding methods.

Next Up Previous
Next: The Shannon-Fano Algorithm **Up:** Lossless Compression Algorithms (Entropy
Previous: Lossless Compression Algorithms (Entropy

Basics of Information Theory

According to Shannon, the entropy of an information source *S* is defined as:

$$H(S) = \eta = \sum_i \; p_i \log_2 \frac{1}{p_i}$$

where p_i is the probability that symbol S_i in *S* will occur.

- $\log_2 \frac{1}{p_i}$ indicates the amount of information contained in S_i, i.e., the number of bits needed to code S_i.
- For example, in an image with uniform distribution of gray-level intensity, i.e. p_i = 1/256, then the number of bits needed to code each gray level is 8 bits. The entropy of this image is 8.
- Q: How about an image in which half of the pixels are white (I = 220) and half are black (I = 10)?

The Shannon-Fano Algorithm

This is a basic information theoretic algorithm. A simple example will be used to illustrate the algorithm:

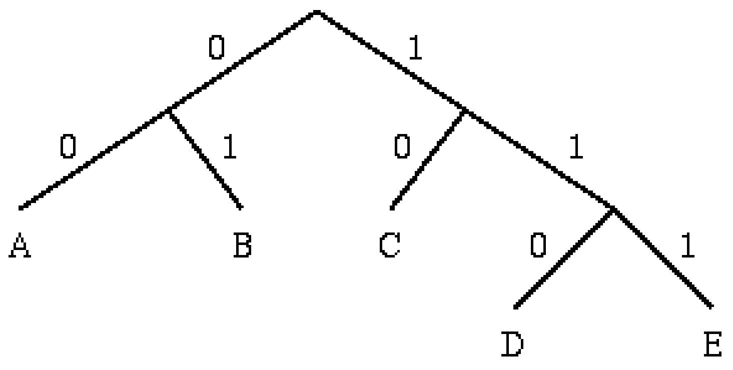
Symbol	A	B	C	D	E
Count	15	7	6	6	5

Encoding for the Shannon-Fano Algorithm:

- A top-down approach

1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.

2. Recursively divide into two parts, each with approx. same number of counts.



Symbol	Count	log(1/p)	Code	Subtotal (# of bits)
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15
			TOTAL (# of bits):	89

Huffman Coding

Huffman coding is based on the frequency of occurance of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently. Codes are stored in a ***Code Book*** which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding.

The Huffman algorithm is now briefly summarised:

- A bottom-up approach

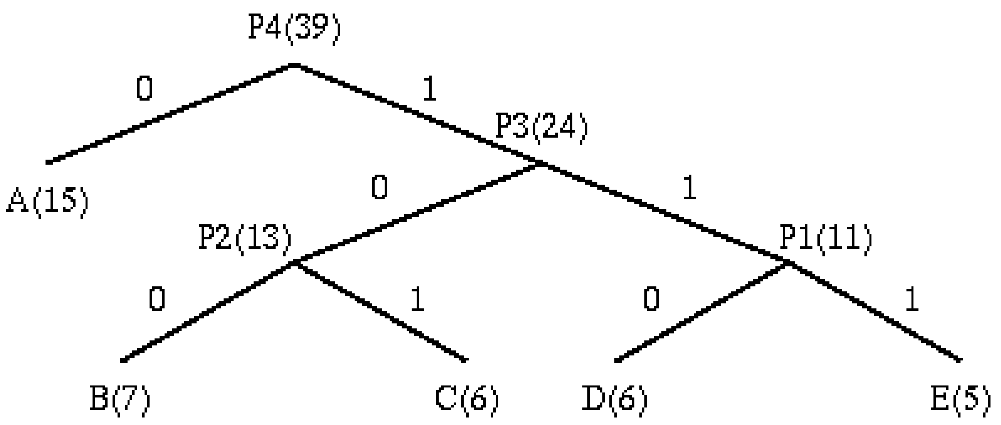
1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).

2. Repeat until the OPEN list has only one node left:

(a) From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.

(b) Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.

(c) Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.



Symbol	Count	log(1/p)	Code	Subtotal (# of bits)
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15

```
                              TOTAL (# of bits): 87
```

The following points are worth noting about the above algorithm:

- Decoding for the above two algorithms is trivial as long as the coding table (the statistics) is sent before the data. (There is a bit overhead for sending this, negligible if the data file is big.)
- **Unique Prefix Property**: no code is a prefix to any other code (all symbols are at the leaf nodes) -> great for decoder, unambiguous.
- If prior statistics are available and accurate, then Huffman coding is very good.

In the above example:

Number of bits needed for Huffman Coding is: 87 / 39 = 2.23

Huffman Coding of Images

In order to encode images:

- Divide image up into 8x8 blocks
- Each block is a symbol to be coded
- compute Huffman codes for set of block
- Encode blocks accordingly

Adaptive Huffman Coding

The basic Huffman algorithm has been extended, for the following reasons:

(a) The previous algorithms require the statistical knowledge which is often not available (e.g., live audio, video).

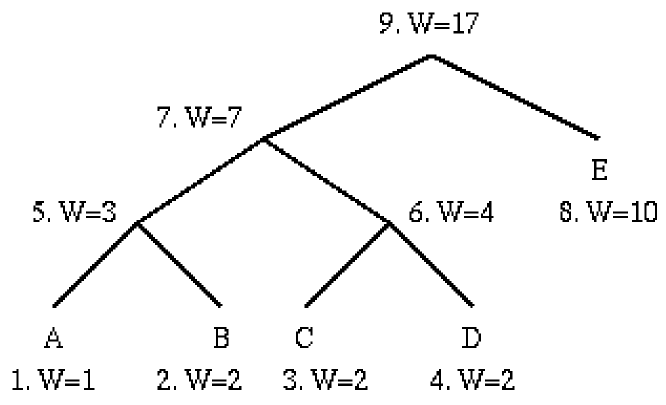
(b) Even when it is available, it could be a heavy overhead especially when many tables had to be sent when a non-order0 model is used, i.e. taking into account the impact of the previous symbol to the probability of the current symbol (e.g., "qu" often come together, ...).

The solution is to use adaptive algorithms. As an example, the Adaptive Huffman Coding is examined below. The idea is however applicable to other adaptive compression algorithms.

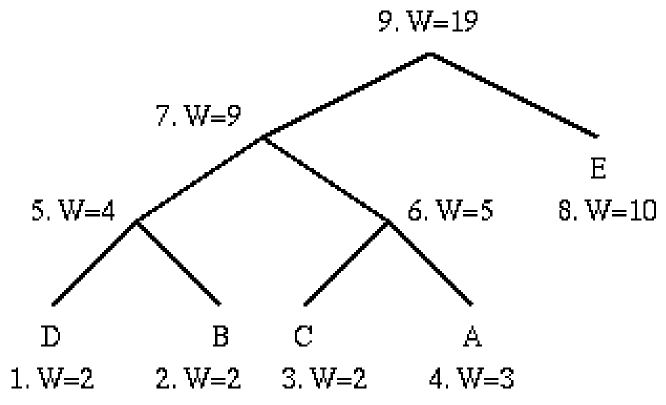
```
ENCODER                                   DECODER
-------                                   -------

Initialize_model();                       Initialize_model();
while ((c = getc (input)) != eof)         while ((c = decode (input)) !=
eof)
  {                                         {
    encode (c, output);                       putc (c, output);
    update_model (c);                         update_model (c);
  }                                         }
```

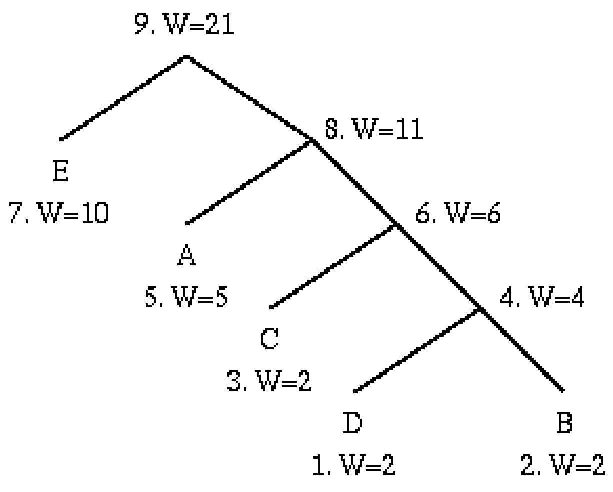
}

- The key is to have both encoder and decoder to use exactly the same *initialization* and *update_model* routines.
- *update_model* does two things: (a) increment the count, (b) update the Huffman tree (Fig 7.2).
    - During the updates, the Huffman tree will be maintained its *sibling property*, i.e. the nodes (internal and leaf) are arranged in order of increasing weights (see figure).
    - When *swapping* is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1. **Note:** If the node with weight W has a subtree beneath it, then the subtree will go with it.
    - The Huffman tree could look very different after node swapping (Fig 7.2), e.g., in the third tree, node A is again swapped and becomes the #5 node. It is now encoded using only 2 bits.



A Huffman Tree





After a node switch (A was incremented twice)

After A was incremented two more times

Note: Code for a particular symbol changes during the adaptive coding process.

Arithmetic Coding

Huffman coding and the like use an integer number (k) of bits for each symbol, hence k is never less than 1. Sometimes, e.g., when sending a 1-bit image, compression becomes impossible.

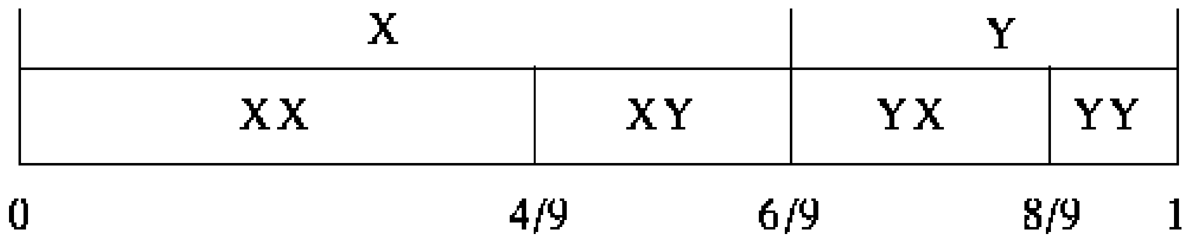
- Idea: Suppose alphabet was

  $$X, Y$$

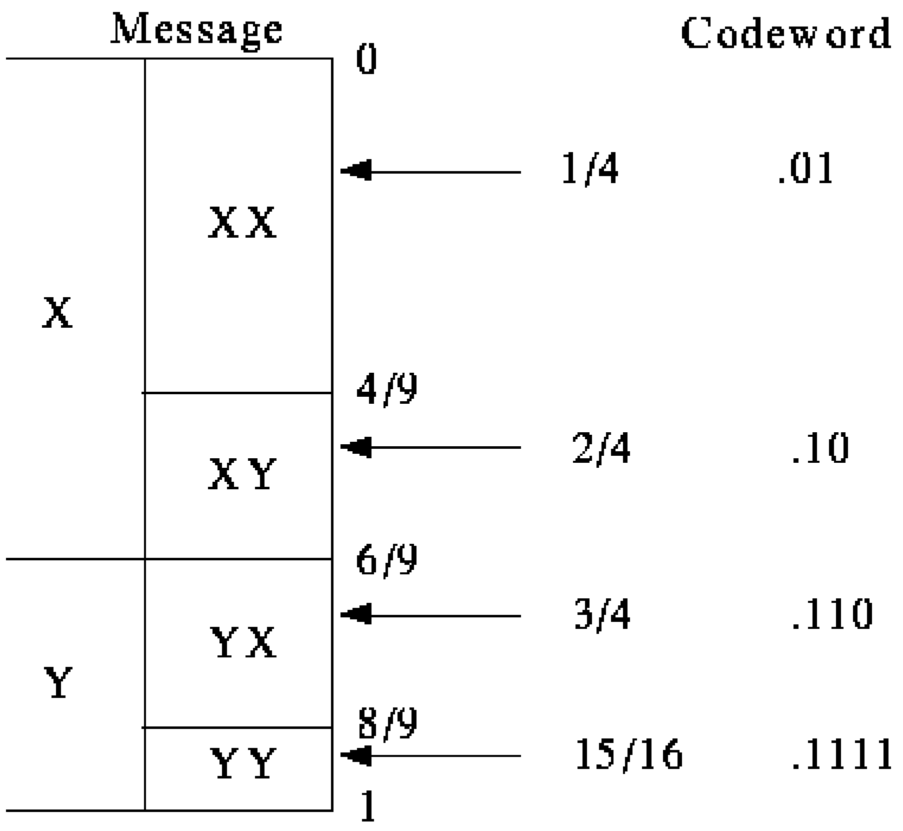
  and

  ```
  prob(X) = 2/3
  prob(Y) = 1/3
  ```

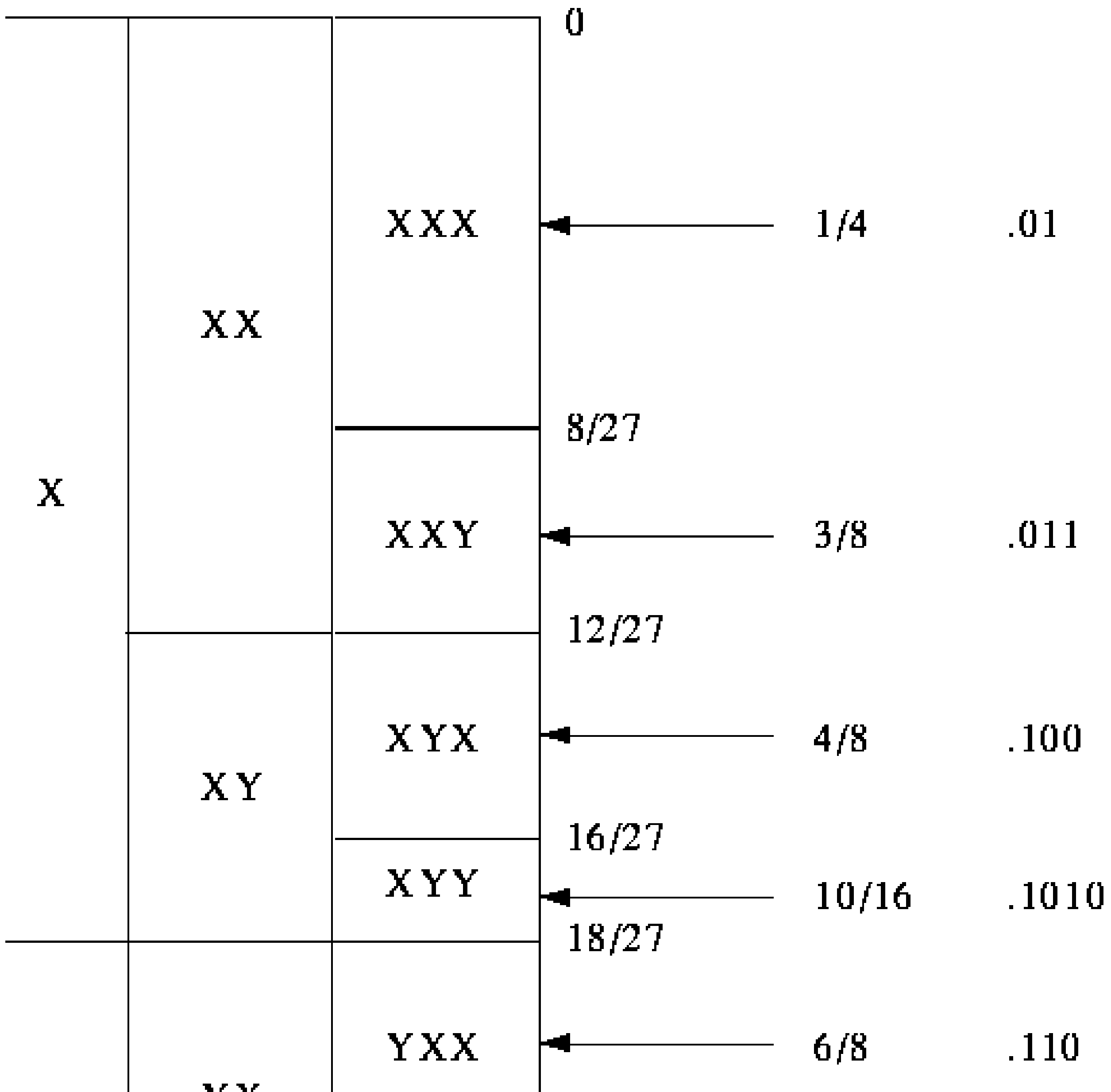
- If we are only concerned with encoding length 2 messages, then we can map all possible messages to intervals in the range [0..1]:

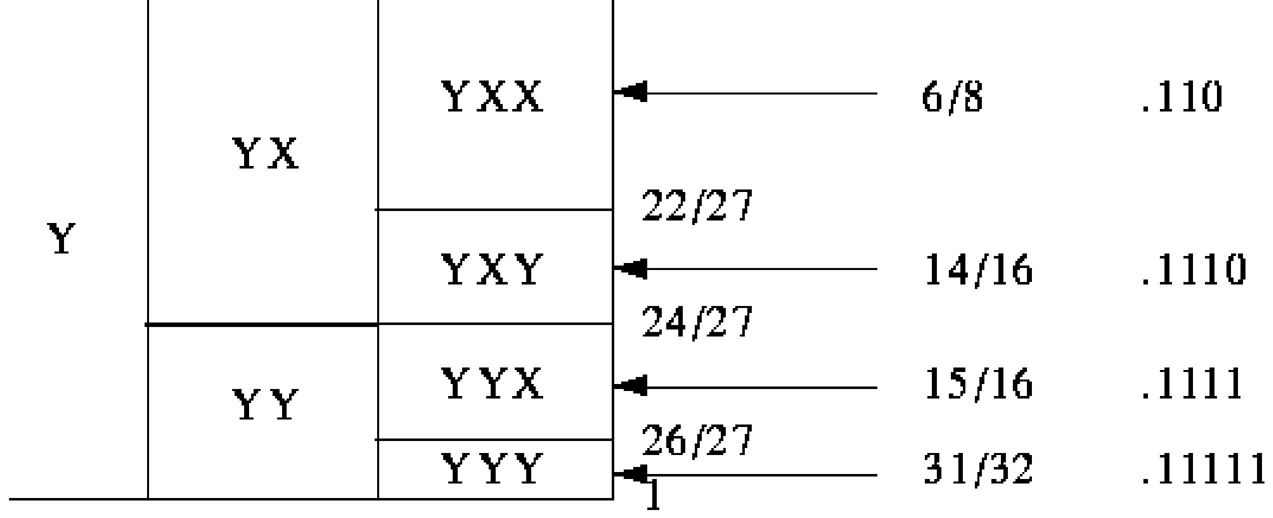


- To encode message, just send enough bits of a binary fraction that uniquely specifies the interval.



- Similarly, we can map all possible length 3 messages to intervals in the range [0..1]:





- Q: How to encode X Y X X Y X ?

  Q: What about an alphabet with 26 symbols, or 256 symbols, ...?

- In general, number of bits is determined by the size of the interval.

  Examples:

  - first interval is 8/27, needs 2 bits -> 2/3 bit per symbol (X)
  - last interval is 1/27, need 5 bits

- In general, need $-\log p$ bits to represent interval of size *p*. Approaches optimal encoding as message length got to infinity.

- Problem: how to determine probabilities?

  - Simple idea is to use adaptive model: Start with guess of symbol frequencies. Update frequency with each new symbol.
  - Another idea is to take account of intersymbol probabilities, e.g., Prediction by Partial Matching.

- Implementation Notes: Can be CPU and memory intensive; patented.

Lempel-Ziv-Welch (LZW) Algorithm

The LZW algorithm is a very common compression technique.

Suppose we want to encode the Oxford Concise English dictionary which contains about 159,000 entries. Why not just transmit each word as an 18 bit number?

Problems:

- Too many bits,
- everyone needs a dictionary,
- only works for English text.
- **Solution**: Find a way to build the dictionary adaptively.

- Original methods due to Ziv and Lempel in 1977 and 1978. Terry Welch improved the scheme in 1984 (called LZW compression).
- It is used in UNIX *compress* -- 1D token stream (similar to below)
- It used in GIF comprerssion -- 2D window tokens (treat image as with Huffman Coding Above).

Reference: Terry A. Welch, "A Technique for High Performance Data Compression", IEEE Computer, Vol. 17, No. 6, 1984, pp. 8-19.

The LZW Compression Algorithm can summarised as follows:

```
w = NIL;
while ( read a character k )
    {
      if wk exists in the dictionary
       w = wk;
      else
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
```

- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.

Example:

```
Input string is "^WED^WE^WEE^WEB^WET".
```

w	k	output	index	symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

- A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits.

- Usually, compression doesn't start until a large number of bytes (e.g., > 100) are read in.

The LZW Decompression Algorithm is as follows:

```
 read a character k;
   output k;
   w = k;
   while ( read a character k )
  /* k could be a character or a code. */
       {
         entry = dictionary entry for k;
         output entry;
         add w + entry[0] to dictionary;
```

```
        w = entry;
      }
```

Example (continued):

Input string is "^WED<256>E<260><261><257>B<260>T".

w	k	output	index	symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

- Problem: What if we run out of dictionary space?
  - Solution 1: Keep track of unused entries and use LRU
  - Solution 2: Monitor compression performance and flush dictionary when performance is poor.
- Implementation Note: LZW can be made *really* fast; it grabs a fixed number of bits from input stream, so bit parsing is very easy. Table lookup is automatic.

Entropy Encoding Summary

- Huffman maps fixed length symbols to variable length codes. Optimal only when symbol probabilities are powers of 2.
- Arithmetic maps entire message to real number range based on statistics. Theoretically optimal for long messages, but optimality depends on data model. Also can be CPU/memory intensive.
- Lempel-Ziv-Welch is a dictionary-based compression method. It maps a variable number of symbols to a fixed length code.
- Adaptive algorithms do not need a priori estimation of probabilities, they are more useful in real applications.

Further Reading/Information

Two good text books:

- ***The Data Compression Book***, Mark Nelson,M&T Books, 1995.
- ***Introduction to Data Compression***, Khalid Sayood, Morgan Kaufmann, 1996.

Source Coding Techniques

Source coding is based on the content of the original signal is also called ***semantic-based coding***

High compression rates may be high but a price of loss of information. Good compression rates make be achieved with source encoding with ***lossless*** or little loss of information.

There are three broad methods that exist:

Transform Coding

A simple transform coding example

A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. Take top left pixel as the base value for the block, pixel A.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. B-A, C-A, D-A.
3. Store the base pixel and the differences as the values of the transform.

Given the above we can easily for the forward transform:

and the inverse transform is:

The above transform scheme may be used to compress data by exploiting redundancy in the data:

Any Redundancy in the data has been transformed to values, X_i. So We can compress the data by using fewer bits to represent the differences. I.e if we use 8 bits per pixel then the 2x2 block uses 32 bits/ If we keep 8 bits for the base pixel, X0, and assign 4 bits for each difference then we only use 20 bits. Which is better than an average 5 bits/pixel

Example

Consider the following 4x4 image block:

120	130
125	120

then we get:

We can then compress these values by taking less bits to represent the data.

However for practical purposes such a simple scheme as outlined above is not sufficient for compression:

- It is **Too Simple**
- Needs to operate on larger blocks (typically 8x8 min)
- Calculation is also too simple and from above we see that simple encoding of differences for large values will result in loss of information -- v poor losses possible here 4 bits per pixel = values 0-15 unsigned, -7 - 7 signed so either quantise in multiples of 255/max value or massive overflow!!

However, More advance transform encoding techniques are very common (See JPEG/MPEG below). Frequncy Domain methods such as Fourier Transform and (more commonly) Discrete Cosine Transforms (DCT) compression techniques fall into this category. We no consider these methods in general and then specifically.

Frequency Domain Methods

Frequency domains can be obtained through the transformation from one (Time or Spatial) domain to the other (Frequency) via

- Discrete Cosine Transform,
- Fourier Transform etc.

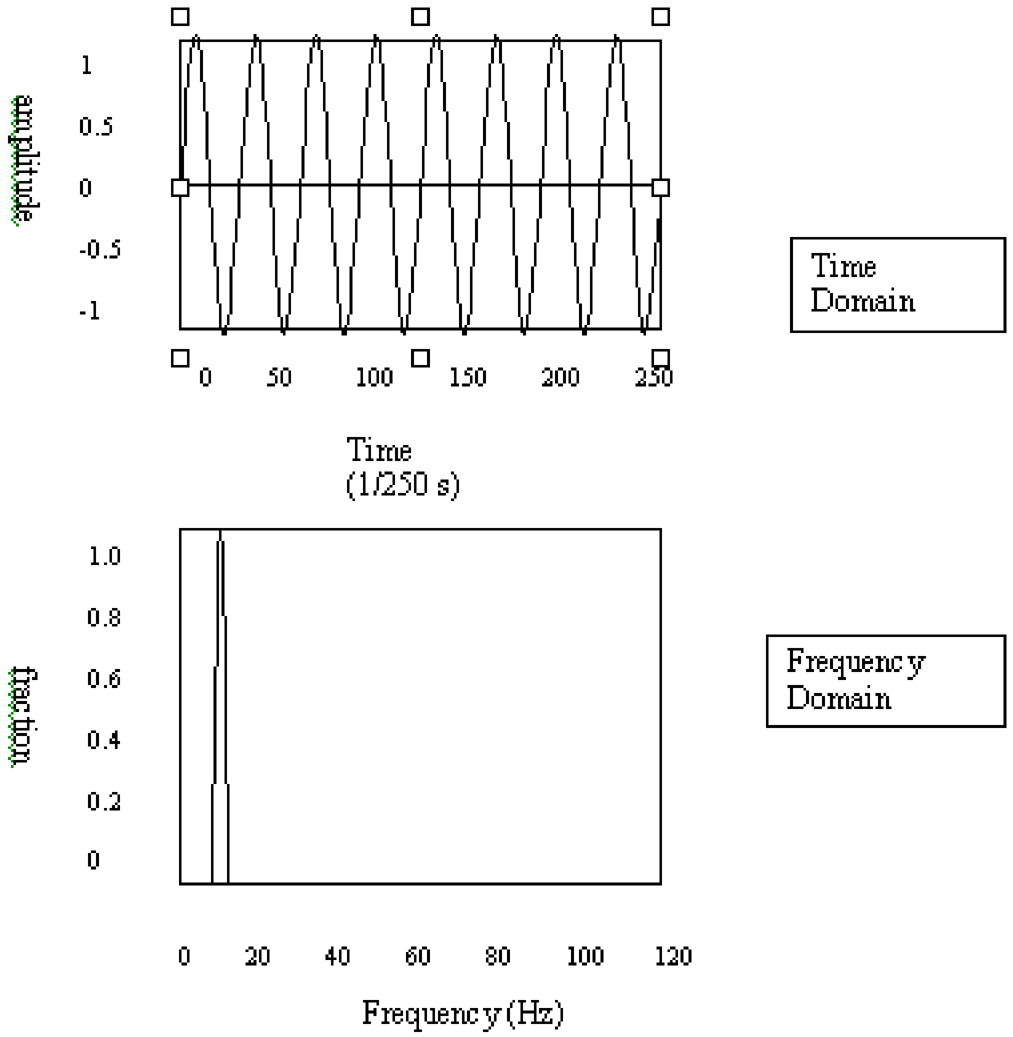
1D Example

Lets consider a 1D (e.g. Audio) example to see what the different domains mean:

Consider a complicated sound such as the noise of a car horn. We can describe this sound in two related ways:

- sample the amplitude of the sound many times a second, which gives an approximation to the sound as a function of time.
- analyse the sound in terms of the pitches of the notes, or frequencies, which make the sound up, recording the amplitude of each frequency.

In the example below (Fig) we have a signal that consists of a sinusoidal wave at 8 Hz. 8Hz means that wave is completing 8 cycles in 1 second and is the frequency of that wave. From the frequency domain we can see that the composition of our signal is one wave (one peak) occurring with a frequency of 8Hz with a magnitude/fraction of 1.0 i.e. it is the whole signal.



Relationship between Time and Frequency Domain

2D (Image) Example

Now images are no more complex really:

Similarly brightness along a line can be recorded as a set of values measured at equally spaced distances apart, or equivalently, at a set of spatial frequency values.

Each of these frequency values is referred to as a ***frequency component***.

An image is a two-dimensional array of pixel measurements on a uniform grid.

This information be described in terms of a two-dimensional grid of spatial frequencies.

A given frequency component now specifies what contribution is made by data which is changing with specified *x* and *y* direction spatial frequencies.

What do frequencies mean in an image?

If an image has large values at ***high*** frequency components then the data is changing rapidly on a short distance scale. ***e.g.*** a page of text

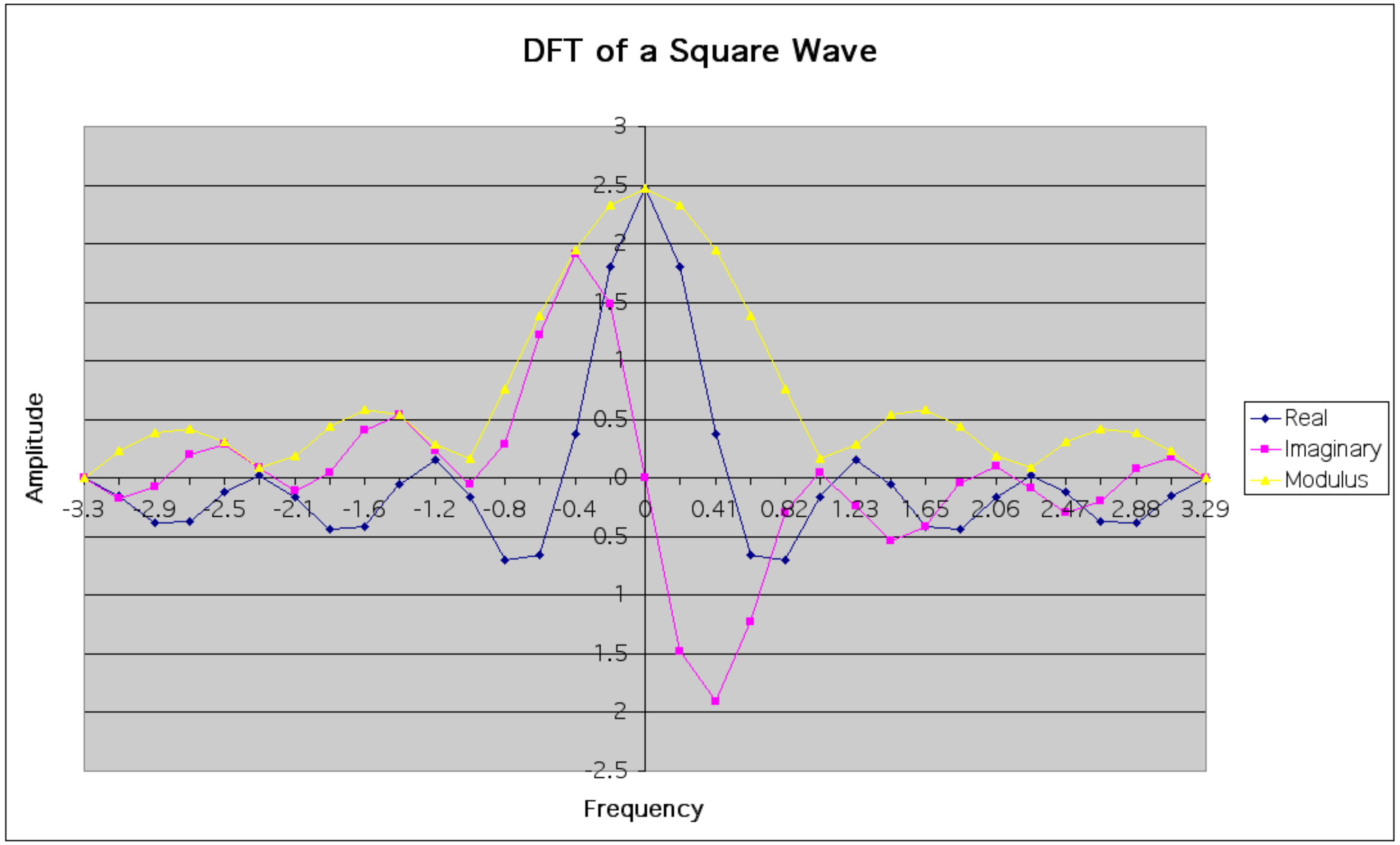
If the image has large ***low*** frequency components then the large scale features of the picture are more important. ***e.g.*** a single fairly simple object which occupies most of the image.

For colour images, The measure (now a 2D matrix) of the frequency content is with regard to colour/chrominance: this shows if values are changing rapidly or slowly. Where the fraction, or value in the frequency matrix is low, the colour is changing gradually. Now the human eye is insensitive to gradual changes in colour and sensitive to intensity. So we can ignore gradual changes in colour and throw away data without the human eye noticing, we hope.

How can transforms into the Frequecny Domain Help?

Any function (signal) can be decomposed into purely sinusoidal components (sine waves of different size/shape) which when added together make up our original signal.

In the example below (Fig 7.4) we have a square wave signal that has been decomposed by the Fourier Transform to render its sinusoidal components. Only the first few sine wave components are shown here. You can see that a the Square wave form will be roughly approximated if you add up the sinusoidal components.

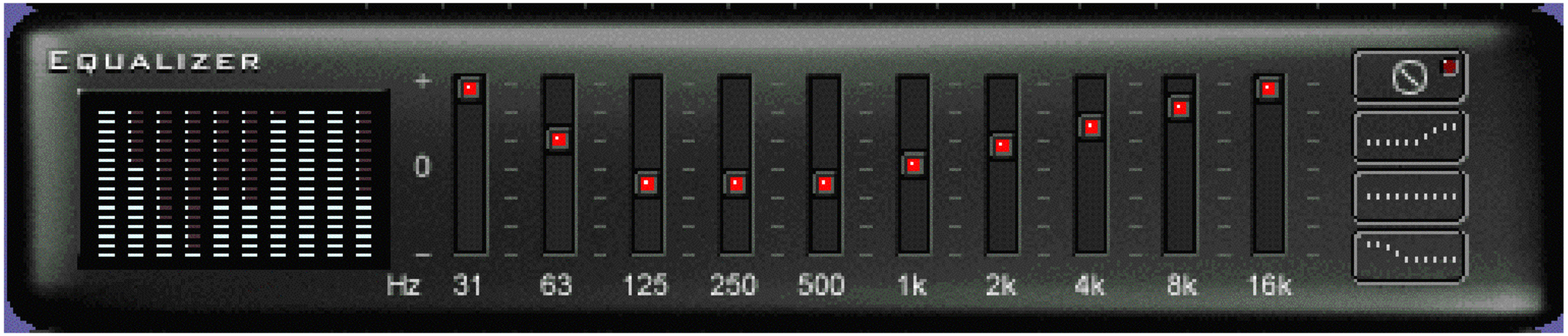


DFT of a Square Wave

Thus Transforming a signal into the frequency domain allows us to see what sine waves make up our signal e.g. One part sinusoidal wave at 50 Hz and two parts sinusoidal waves at 200 Hz.

More complex signals will give more complex graphs but the idea is exactly the same. The graph of the frequency domain is called the frequency spectrum.

An easy way to visualise what is happening is to think of a graphic equaliser on a stereo (Fig 7.5).



A Graphic Equaliser

The bars on the left are the frequency spectrum of the sound that you are listening to. The bars go up and down depending on the type of sound that you are listening to. It is pretty obvious that the accumulation of these make up the whole. The bars on the right are used to increase and decrease the sound at particular frequencies, denoted by the numbers (Hz). The lower frequencies, on the left, are for bass and the higher frequencies on the right are treble.

This is directly related to our example before. The bars show how much of the signal is made up of sinusoidal waves at that frequency. When all the waves are added together in their correct proportions that original sound is regenerated.

Next Up Previous
Next: Fourier Theory **Up:** Frequency Domain Methods **Previous:** What do frequencies mean
Dave Marshall
10/4/2001

Fourier Theory

In order to fully comprehend the DCT will do a basic study of the Fourier theory and the Fourier transform first.

Whilst the DCT is ultimately used in multimedia compression it is easier to perhaps comprehend how such compression methods work by studying Fourier theory, from which the DCT is actually derived.

The tool which converts a spatial (real space) description of an image into one in terms of its frequency components is called the **Fourier transform**

The new version is usually referred to as the **Fourier space description** of the image.

The corresponding ***inverse*** transformation which turns a Fourier space description back into a real space one is called the **inverse Fourier transform**.

1D Case

Considering a continuous function *f*(*x*) of a single variable *x* representing distance.

The Fourier transform of that function is denoted *F*(*u*), where *u* represents spatial frequency is defined by

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x u}\, dx. \tag{1}$$

Note: In general *F*(*u*) will be a complex quantity ***even though*** the original data is purely **real**.

The meaning of this is that not only is the magnitude of each frequency present important, but that its phase relationship is too.

The inverse Fourier transform for regenerating *f*(*x*) from *F*(*u*) is given by

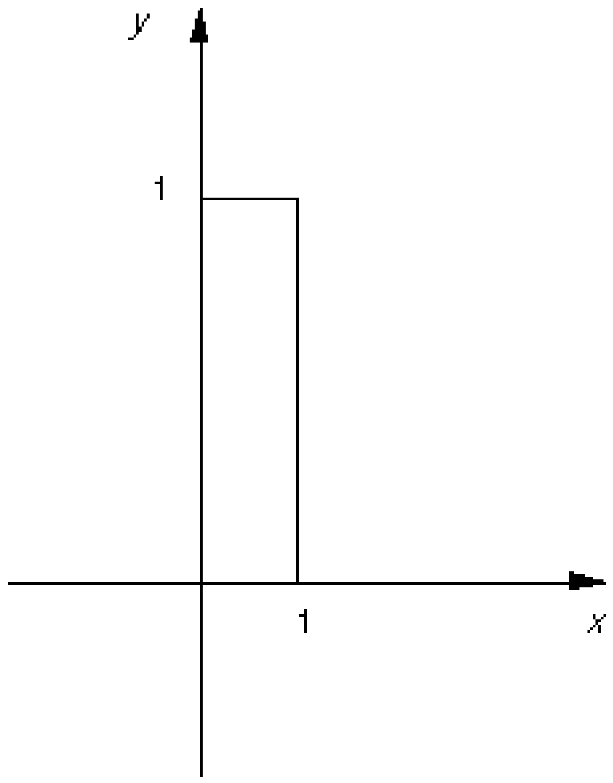
$$f(x) = \int_{-\infty}^{\infty} F(u) e^{2\pi i x u}\, du, \tag{2}$$

which is rather similar, except that the exponential term has the opposite sign.

Let's see how we compute a Fourier Transform: consider a particular function *f*(*x*) defined as

$$f(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

shown in Fig. 7.6.



A top hat function

So its Fourier transform is:

In this case *F*(*u*) is purely real, which is a consequence of the original data being symmetric in *x* and -*x*. A graph of *F*(*u*) is shown in Fig. 7.7. This function is often referred to as the Sinc function.

Fourier transform of a top hat function

2D Case

If *f*(*x*,*y*) is a function, for example the brightness in an image, its Fourier transform is given by

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-2\pi i(xu+yv)}\, dx\, dy, \tag{4}$$

and the inverse transform, as might be expected, is

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{2\pi i(xu+yv)}\, du\, dv. \tag{5}$$

The Discrete Fourier Transform (DFT)

Images and Digital Audio are digitised !!

Thus, we need a ***discrete*** formulation of the Fourier transform, which takes such regularly spaced data values, and returns the value of the Fourier transform for a set of values in frequency space which are equally spaced.

This is done quite naturally by replacing the integral by a summation, to give the ***discrete Fourier transform*** or DFT for short.

In 1D it is convenient now to assume that *x* goes up in steps of 1, and that there are *N* samples, at values of *x* from to *N*-1.

So the DFT takes the form

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-2\pi i x u/N}, \tag{6}$$

while the inverse DFT is

$$f(x) = \sum_{x=0}^{N-1} F(u) e^{2\pi i x u/N}. \tag{7}$$

NOTE: Minor changes from the continuous case are a factor of 1/*N* in the exponential terms, and also the factor 1/*N* in front of the forward transform which does not appear in the inverse transform.

The 2D DFT works is similar. So for an $N \times M$ grid in *x* and *y* we have

$$F(u,v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-2\pi i (xu/N + yv/M)}, \tag{8}$$

and

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{2\pi i (xu/N + yv/M)}. \tag{9}$$

Often *N*=*M*, and it is then it is more convenient to redefine *F*(*u*,*v*) by multiplying it by a factor of *N*, so that the forward and inverse transforms are more symmetrical:

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi i (xu+yv)/N}, \tag{10}$$

and

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{2\pi i (xu+yv)/N}. \tag{11}$$

Compression

How do we achieve compression:

- Low pass filter -- ignore high frequency noise components
- Only store lower frequency components
- High Pass Filter -- Spot Gradual Changes
- If changes to low Eye does not respond so ignore?

Where do put threshold to cut off?

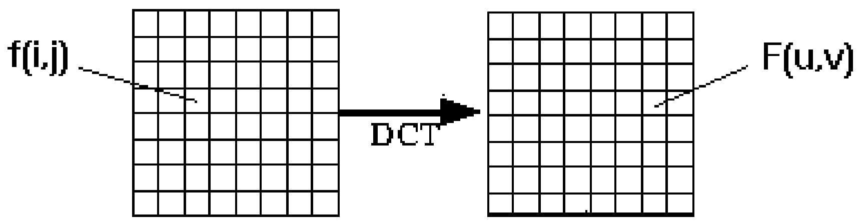
Relationship between DCT and FFT

DCT (Discrete Cosine Transform) is actually a ***cut-down*** version of the FFT:

- Only the **real** part of FFT
- Computationally simpler than FFT
- DCT -- Effective for Multimedia Compression
- DCT **MUCH** more commonly used.

The Discrete Cosine Transform (DCT)

The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain (Fig 7.8).



DCT Encoding

The general equation for a 1D (*N* data items) DCT is defined by the following equation:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(i).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] f(i)$$

and the corresponding ***inverse*** 1D DCT transform is simple $F^{-1}(u)$, i.e.:

where

$$\Lambda(i) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

The general equation for a 2D (*N* by *M* image) DCT is defined by the following equation:

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i).\Lambda(j).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] cos\left[\frac{\pi.v}{2.M}(2j+1)\right].f(i,j)$$

and the corresponding ***inverse*** 2D DCT transform is simple $F^{-1}(u,v)$, i.e.:

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

The basic operation of the DCT is as follows:

- The input image is N by M;
- f(i,j) is the intensity of the pixel in row i and column j;
- F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix.

- For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT.
- Compression is achieved since the lower right values represent higher frequencies, and are often small - small enough to be neglected with little visible distortion.
- The DCT input is an 8 by 8 array of integers. This array contains each pixel's gray scale level;
- 8 bit pixels have levels from 0 to 255.

- Therefore an 8 point DCT would be:

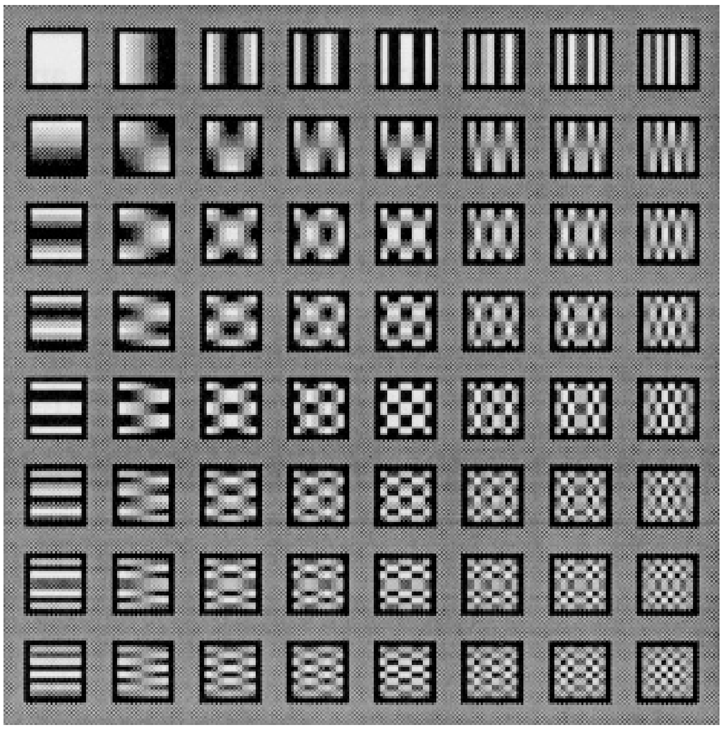
  where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

  Question: What is F[0,0]?

  answer: They define DC and AC components.
- The output array of DCT coefficients contains integers; these can range from -1024 to 1023.

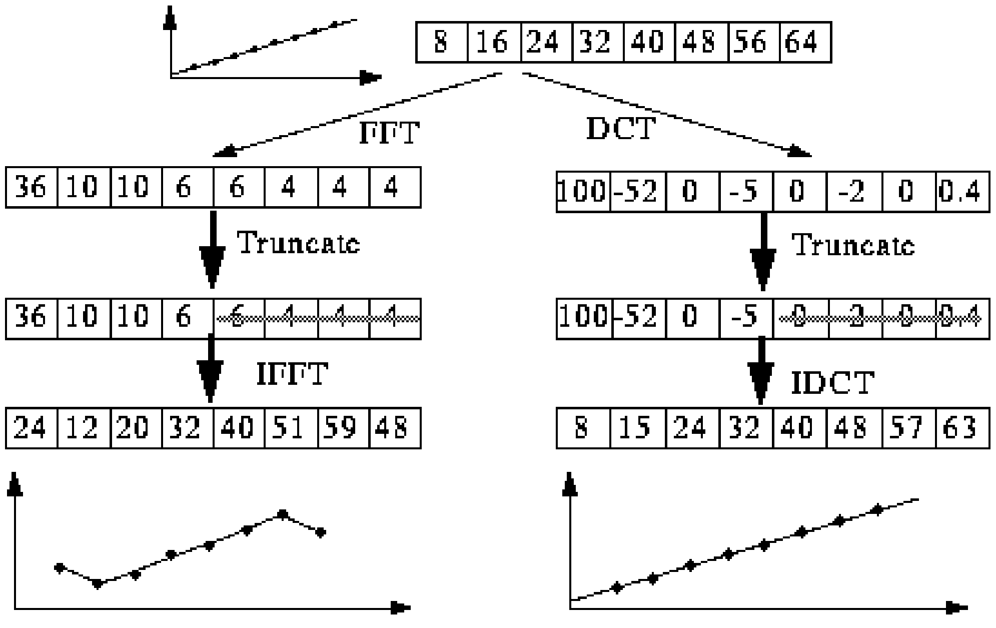
- It is computationally easier to implement and more efficient to regard the DCT as a set of **basis functions** which given a known input array size (8 x 8) can be precomputed and stored. This involves simply computing values for a convolution mask (8 x8 window) that get applied (summ values x pixelthe window overlap with image apply window accros all rows/columns of image). The values as simply calculated from the DCT formula. The 64 (8 x 8) DCT basis functions are illustrated in Fig 7.9.



DCT basis functions

- Why DCT not FFT?

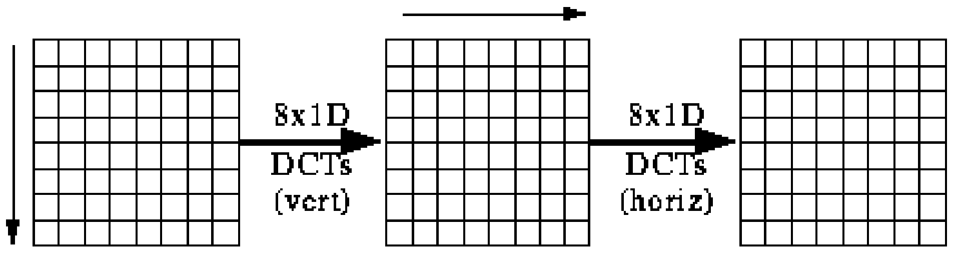
  DCT is similar to the Fast Fourier Transform (FFT), but can approximate lines well with fewer coefficients (Fig 7.10)



DCT/FFT Comparison

- Computing the 2D DCT
    - Factoring reduces problem to a series of 1D DCTs (Fig 7.11):
        - apply 1D DCT (Vertically) to Columns
        - apply 1D DCT (Horizontally) to resultant Vertical DCT above.
        - or alternatively Horizontal to Vertical.

    The equations are given by:



    - Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.
    - World record is 11 multiplies and 29 adds. (C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics, Speech, and Signal Processing 1989 (ICASSP `89), pp. 988-991)

Differential Encoding

Simple example of transform coding mentioned earlier and instance of this approach.

Here:

- The difference between the actual value of a sample and a prediction of that values is encoded.
- Also known as ***predictive encoding***.
- Example of technique include: differential pulse code modulation, delta modulation and adaptive pulse code modulation -- differ in prediction part.
- Suitable where successive signal samples do not differ much, but are not zero. ***E.g.*** Video -- difference between frames, some audio signals.
- ***Differential pulse code modulation*** (DPCM) simple prediction:

  $$f_{predict}(t_i) = f_{actual}(t_{i-1})$$

  i.e. a simple Markov model where current value is the predict next value.

  So we simply need to encode:

  $$\Delta f(t_i) = f_{actual}(t_i) - f_{actual}(t_{i-1})$$

  If successive sample are close to each other we only need to encode first sample with a large number of bits:

  Actual Data: 9 10 7 6

  Predicted Data: 0 9 10 7

  $\Delta f(t)$: +9, +1, -3, -1.
- ***Delta modulation*** is a special case of DPCM: Same predictor function, coding error is a single bit or digit that indicates the current sample should be increased or decreased by a step.

  Not Suitable for rapidly changing signals.
- ***Adaptive pulse code modulation*** -- Fuller Markov model: data is extracted from

a function of a series of previous values: ***E.g.*** Average of last *n* samples.
Characteristics of sample better preserved.

Vector Quantisation

The basic outline of this approach is:

- Data stream divided into (1D or 2D square) blocks -- ***vectors***
- A table or ***code book*** is used to find a pattern for each block.
- Code book can be dynamically constructed or predefined.
- Each pattern for block encoded as a look value in table
- Compression achieved as data is effectively subsampled and coded at this level.

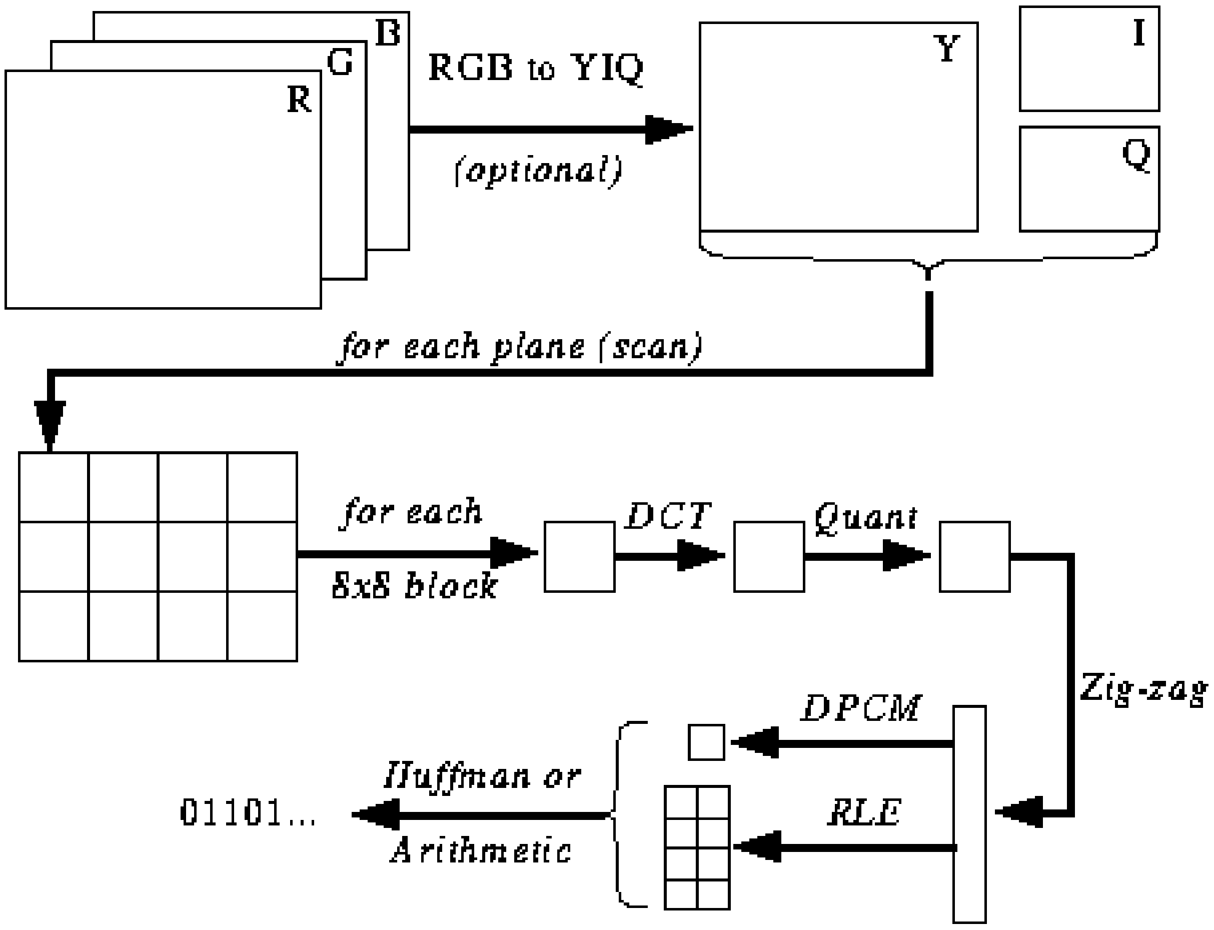
JPEG Compression

What is JPEG?

- "Joint Photographic Expert Group" -- an international standard in 1992.
- Works with colour and greyscale images, Many applications e.g., satellite, medical, ...

JPEG compression involves the following:

- Encoding (Fig 7.12)



JPEG Encoding

- Decoding - Reverse the order for encoding

The Major Steps in JPEG Coding involve:

- DCT (Discrete Cosine Transformation)
- Quantization
- Zigzag Scan
- DPCM on DC component
- RLE on AC Components
- Entropy Coding

The DCT has already been introduced (Section 7.5.4) we now summarise the other steps.

Quantization

Why do we need to quantise:

- To throw out bits
- ***Example***: 101101 = 45 (6 bits).

  Truncate to 4 bits: 1011 = 11.

  Truncate to 3 bits: 101 = 5.
- Quantization error is the main source of the Lossy Compression.

Uniform quantization

- Divide by constant *N* and round result (*N* = 4 or 8 in examples above).
- Non powers-of-two gives fine control (e.g., *N* = 6 loses 2.5 bits)

Quantization Tables

- In JPEG, each F[u,v] is divided by a constant q(u,v).
- Table of q(u,v) is called *quantization table*.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

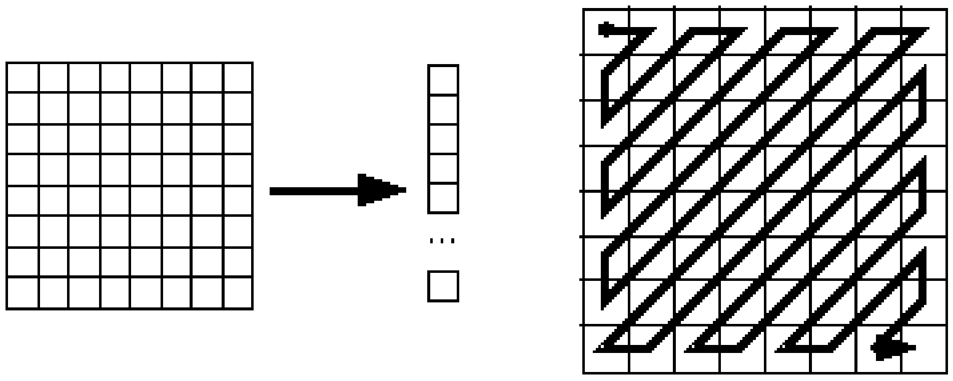
- Eye is most sensitive to low frequencies (upper left corner), less sensitive to high frequencies (lower right corner)
- Standard defines 2 default quantization tables, one for luminance (above), one for chrominance.
- Q: How would changing the numbers affect the picture (e.g., if I doubled them all)?

  Quality factor in most implementations is the scaling factor for default quantization tables.
- Custom quantization tables can be put in image/scan header.

Zig-zag Scan

What is the purpose of the Zig-zag Scan:

- to group low frequency coefficients in top of vector.
- Maps 8 x 8 to a 1 x 64 vector



Differential Pulse Code Modulation (DPCM) on DC component

Here we see that besides DCT another encoding method is employed: DPCM on the DC component at least. Why is this strategy adopted:

- DC component is large and varied, but often close to previous value (like lossless JPEG).
- Encode the difference from previous 8x8 blocks - DPCM

Run Length Encode (RLE) on AC components

Yet another simple compression technique is applied to the AC component:

- 1x64 vector has lots of zeros in it
- Encode as (*skip*, *value*) pairs, where *skip* is the number of zeros and *value* is the next non-zero component.
- Send (0,0) as end-of-block sentinel value.

Next Up Previous
Next: Summary of the JPEG **Up:** JPEG Compression **Previous:** Run Length Encode (RLE)

Entropy Coding

DC and AC components finally need to be represented by a smaller number of bits:

- Categorize DC values into SSS (number of bits needed to represent) and actual bits.

Value	SSS
0	0
-1,1	1
-3,-2,2,3	2
-7..-4,4..7	3

- *Example*: if DC value is 4, 3 bits are needed.

  Send off SSS as Huffman symbol, followed by actual 3 bits.

- For AC components (*skip*, *value*), encode the composite symbol (*skip*,*SSS*) using the Huffman coding.

- Huffman Tables can be custom (sent in header) or default.

Summary of the JPEG bitstream

Figure 7.12 and the above JPEG components have described how compression is achieved at several stages. Let us conclude by summarising the overall compression process:

- A "Frame" is a picture, a "scan" is a pass through the pixels (e.g., the red component), a "segment" is a group of blocks, a "block" is an 8x8 group of pixels.
- Frame header: sample precision (width, height) of image number of components unique ID (for each component) horizontal/vertical sampling factors (for each component) quantization table to use (for each component)
- Scan header Number of components in scan component ID (for each component) Huffman table for each component (for each component)
- Misc. (can occur between headers) Quantization tables Huffman Tables Arithmetic Coding Tables Comments Application Data

Practical JPEG Compression

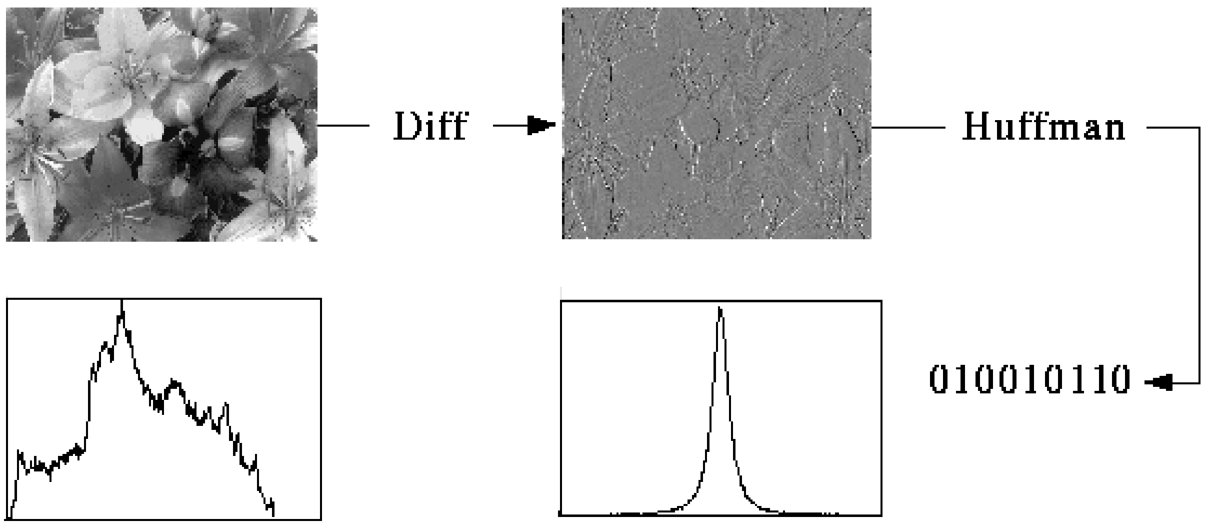
JPEG compression algorithms may fall in to one of several categories depending on how the compression is actually performed:

- Baseline/Sequential - the one that we described in detail
- Lossless
- Progressive
- Hierarchical
- "Motion JPEG" - Baseline JPEG applied to each image in a video.

Briefly, this is how each above approach is encoded:

1.
Lossless Mode

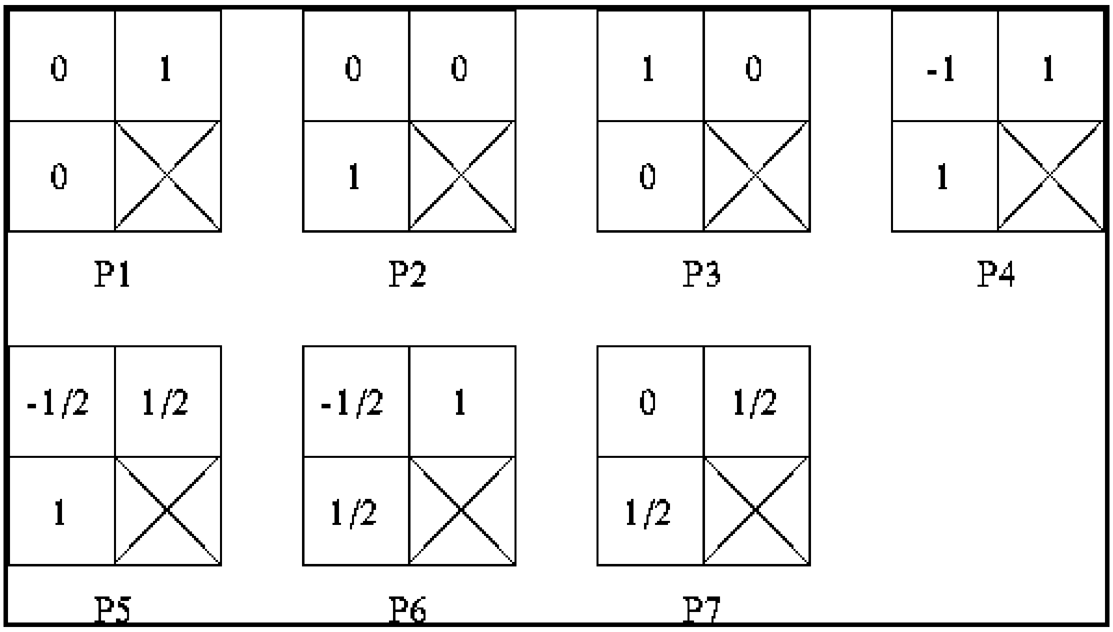
- A special case of the JPEG where indeed there is no loss



- Take difference from previous pixels (not blocks as in the Baseline mode) as a "predictor".

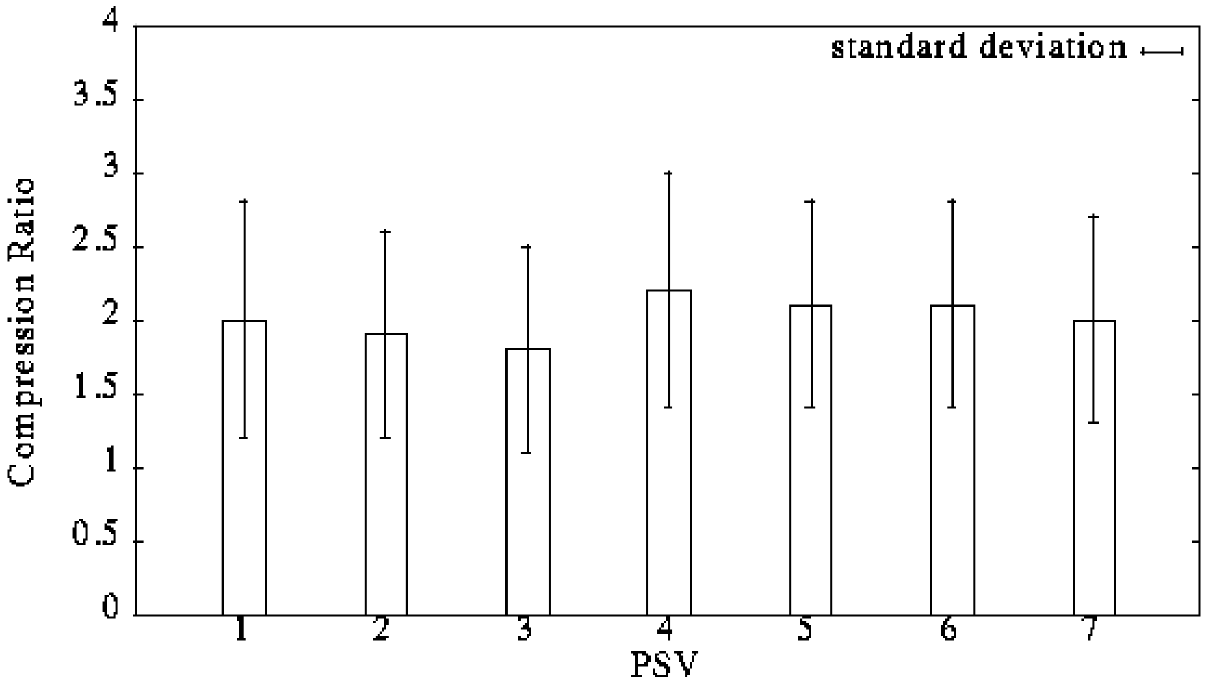
  Predictor uses linear combination of previously encoded neighbors.

  It can be one of seven different predictor based on pixels neighbors



- Since it uses only previously encoded neighbors, first row always uses P2, first column always uses P1.

- Effect of Predictor (test with 20 images)



Note: "2D" predictors (4-7) always do better than "1D" predictors.

Comparison with Other Lossless Compression Programs (compression ratio):

```
-----------------------------------------------------------------
      Compression Program              Compression Ratio
                                 Lena  football    F-18   flowers
-----------------------------------------------------------------
        lossless JPEG            1.45     1.54     2.29     1.26
    optimal lossless JPEG        1.49     1.67     2.71     1.33
       compress (LZW)            0.86     1.24     2.21     0.87
      gzip (Lempel-Ziv)          1.08     1.36     3.10     1.05
gzip -9 (optimal Lempel-Ziv)     1.08     1.36     3.13     1.05
    pack (Huffman coding)        1.02     1.12     1.19     1.00
```

- -

2.

Progressive Mode

- Goal: display low quality image and successively improve.
- Two ways to successively improve image:

  (a)

  Spectral selection: Send DC component, then first few AC, some more AC, etc.

  (b)

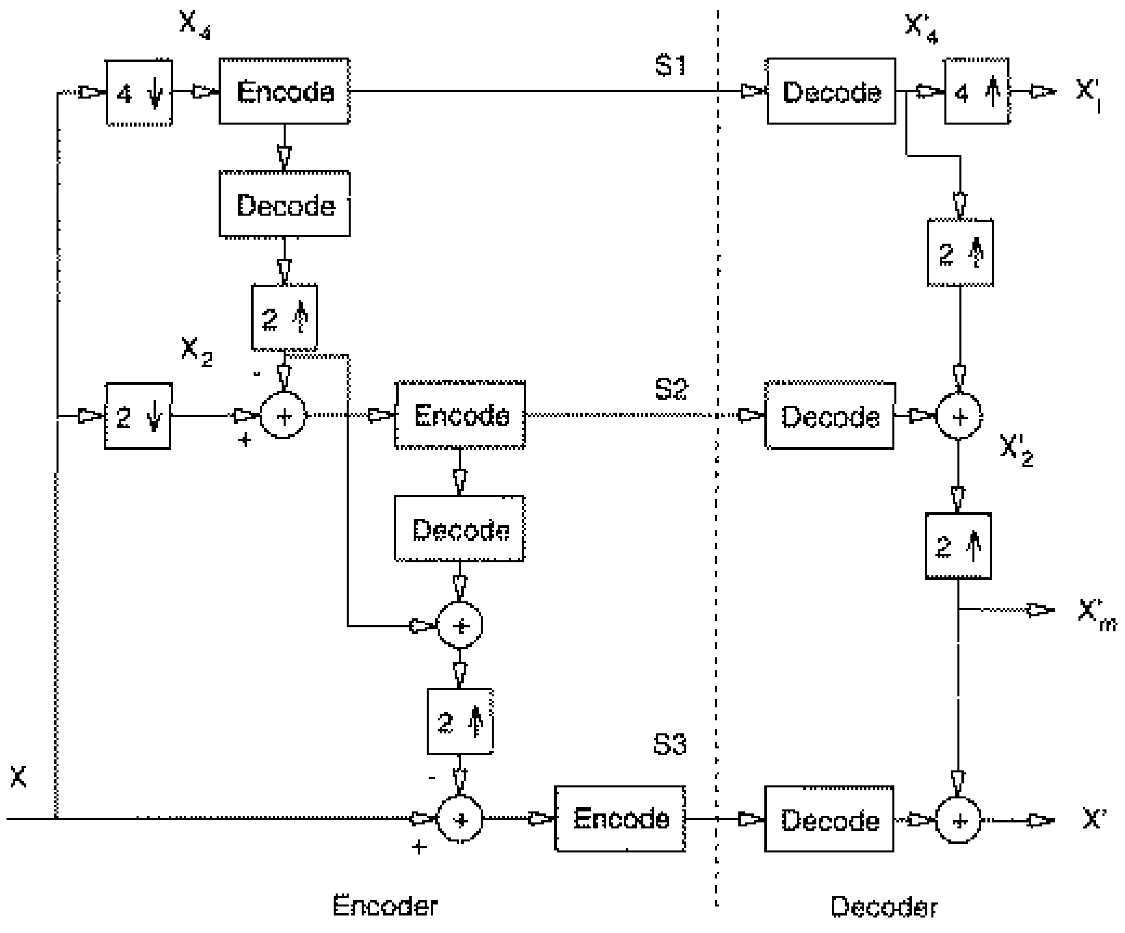
  Successive approximation: send DCT coefficients MSB (most significant bit) to LSB (least significant bit).

3.

Hierarchical Mode

A Three-level Hierarchical JPEG Encoder

(From *V. Bhaskaran and K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures", Kluwer Academic Publishers, 1995.*)



- Down-sample by factors of 2 in each direction.

  Example: map 640x480 to 320x240

- Code smaller image using another method (Progressive, Baseline, or Lossless).

- Decode and up-sample encoded image
- Encode difference between the up-sampled and the original using Progressive, Baseline, or Lossless.
- Can be repeated multiple times.
- Good for viewing high resolution image on low resolution display.

4.
JPEG-2

- Big change was to use *adaptive quantization*

Further Reading

A good tutorial on JPEG may be found at:

http://www.ece.purdue.edu/~ace/jpeg-tut/jpegtut1.html

Video Compression

We have studied the theory of encoding now let us see how this is applied in practice.

We need to compress video (and audio) in practice since:

1.
Uncompressed video (and audio) data are huge. In HDTV, the bit rate easily exceeds 1 Gbps. -- big problems for storage and network communications. For example:

One of the formats defined for HDTV broadcasting within the United States is 1920 pixels horizontally by 1080 lines vertically, at 30 frames per second. If these numbers are all multiplied together, along with 8 bits for each of the three primary colors, the total data rate required would be approximately 1.5 Gb/sec. Because of the 6 MHz. channel bandwidth allocated, each channel will only support a data rate of 19.2 Mb/sec, which is further reduced to 18 Mb/sec by the fact that the channel must also support audio, transport, and ancillary data information. As can be seen, this restriction in data rate means that the original signal must be compressed by a figure of approximately 83:1. This number seems all the more impressive when it is realized that the intent is to deliver very high quality video to the end user, with as few visible artifacts as possible.

2.
Lossy methods have to employed since the *compression ratio* of lossless methods (e.g., Huffman, Arithmetic, LZW) is not high enough for image and video compression, especially when distribution of pixel values is relatively flat.

The following compression types are commonly used in Video compression:

- Spatial Redundancy Removal - Intraframe coding (JPEG)
- Spatial and Temporal Redundancy Removal - Intraframe and Interframe coding (H.261, MPEG)

These are discussed in the following sections.

H. 261 Compression

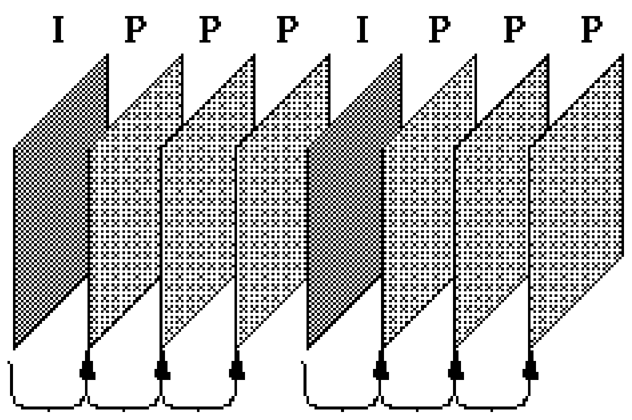
H. 261 Compression has been specifically designed for video telecommunication applications:

- Developed by CCITT in 1988-1990
- Meant for videoconferencing, videotelephone applications over ISDN telephone lines.
- Baseline ISDN is 64 kbits/sec, and integral multiples (*p*x64)

Overview of H.261

The basic approach to H. 261 Compression is summarised as follows:

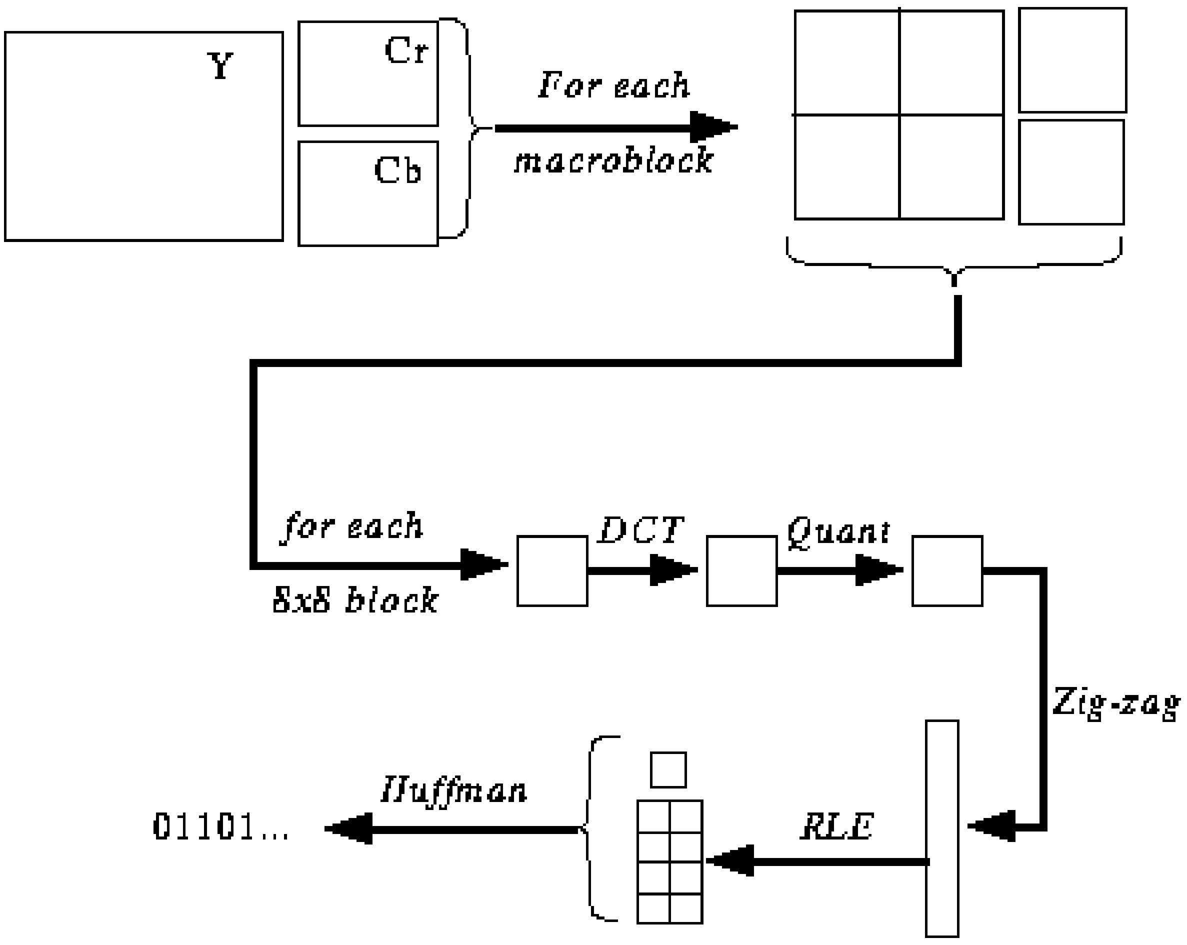
- Decoded Sequence



- Frame types are CCIR 601 CIF (352x288) and QCIF (176x144) images with 4:2:0 subsampling.
- Two frame types: Intraframes (*I-frames*) and Interframes (*P-frames*)
- I-frames use basically JPEG
- P-frames use ***pseudo-differences*** from previous frame (predicted), so frames depend on each other.
- I-frame provide us with an accessing point.

Intra Frame Coding

The term ***intra frame coding*** refers to the fact that the various lossless and lossy compression techniques are performed relative to information that is contained only within the current frame, and not relative to any other frame in the video sequence. In other words, no temporal processing is performed outside of the current picture or frame. This mode will be described first because it is simpler, and because non-intra coding techniques are extensions to these basics. Figure 1 shows a block diagram of a basic video encoder for intra frames only. It turns out that this block diagram is very similar to that of a JPEG still image video encoder, with only slight implementation detail differences.



The potential ramifications of this similarity will be discussed later. The basic processing blocks shown are the video filter, discrete cosine transform, DCT coefficient quantizer, and run-length amplitude/variable length coder. These blocks are described individually in the sections below or have already been described in JPEG Compression.

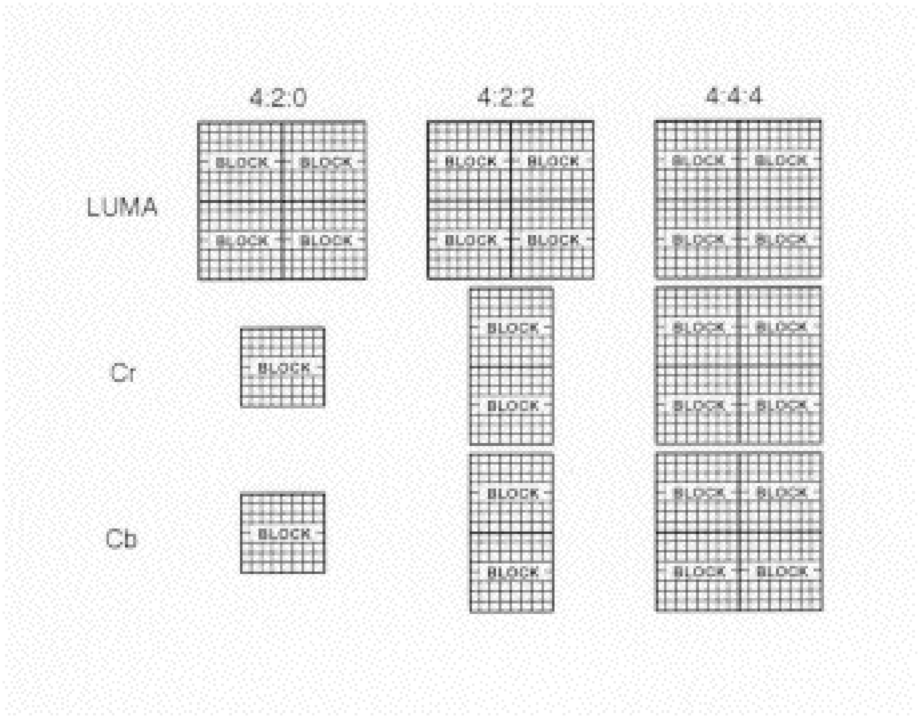
This is a basic Intra Frame Coding Scheme is as follows:

- Macroblocks are 16x16 pixel areas on Y plane of original image.

  A ***macroblock*** usually consists of 4 Y blocks, 1 Cr block, and 1 Cb block.

  In the example HDTV data rate calculation shown previously, the pixels were represented as 8-bit values for each of the primary colors - red, green, and blue. It turns out that while this may be good for high performance computer generated graphics, it is wasteful in most video compression applications. Research into the Human Visual System (HVS) has shown that the eye is most sensitive to changes in luminance, and less sensitive to variations in chrominance. Since absolute compression is the name of the game, it makes sense that MPEG should operate on a color space that can effectively take advantage of the eye¹s different sensitivity to luminance and chrominance information. As such, H/261 (and MPEG) uses the YCbCr color space to represent the data values instead of RGB, where Y is the luminance signal, Cb is the blue color difference signal, and Cr is the red color difference signal.

  A macroblock can be represented in several different manners when referring to the YCbCr color space. Figure 7.13 below shows 3 formats known as 4:4:4, 4:2:2, and 4:2:0 video. 4:4:4 is full bandwidth YCbCr video, and each macroblock consists of 4 Y blocks, 4 Cb blocks, and 4 Cr blocks. Being full bandwidth, this format contains as much information as the data would if it were in the RGB color space. 4:2:2 contains half as much chrominance information as 4:4:4, and 4:2:0 contains one quarter of the chrominance information. Although MPEG-2 has provisions to handle the higher chrominance formats for professional applications, most consumer level products will use the normal 4:2:0 mode.



Macroblock Video Formats

Because of the efficient manner of luminance and chrominance representation, the 4:2:0 representation allows an immediate data reduction from 12 blocks/macroblock to 6 blocks/macroblock, or 2:1 compared to full bandwidth representations such as 4:4:4 or RGB. To generate this format without generating color aliases or artifacts requires that the chrominance signals be filtered.

The Macroblock is coded as follows:

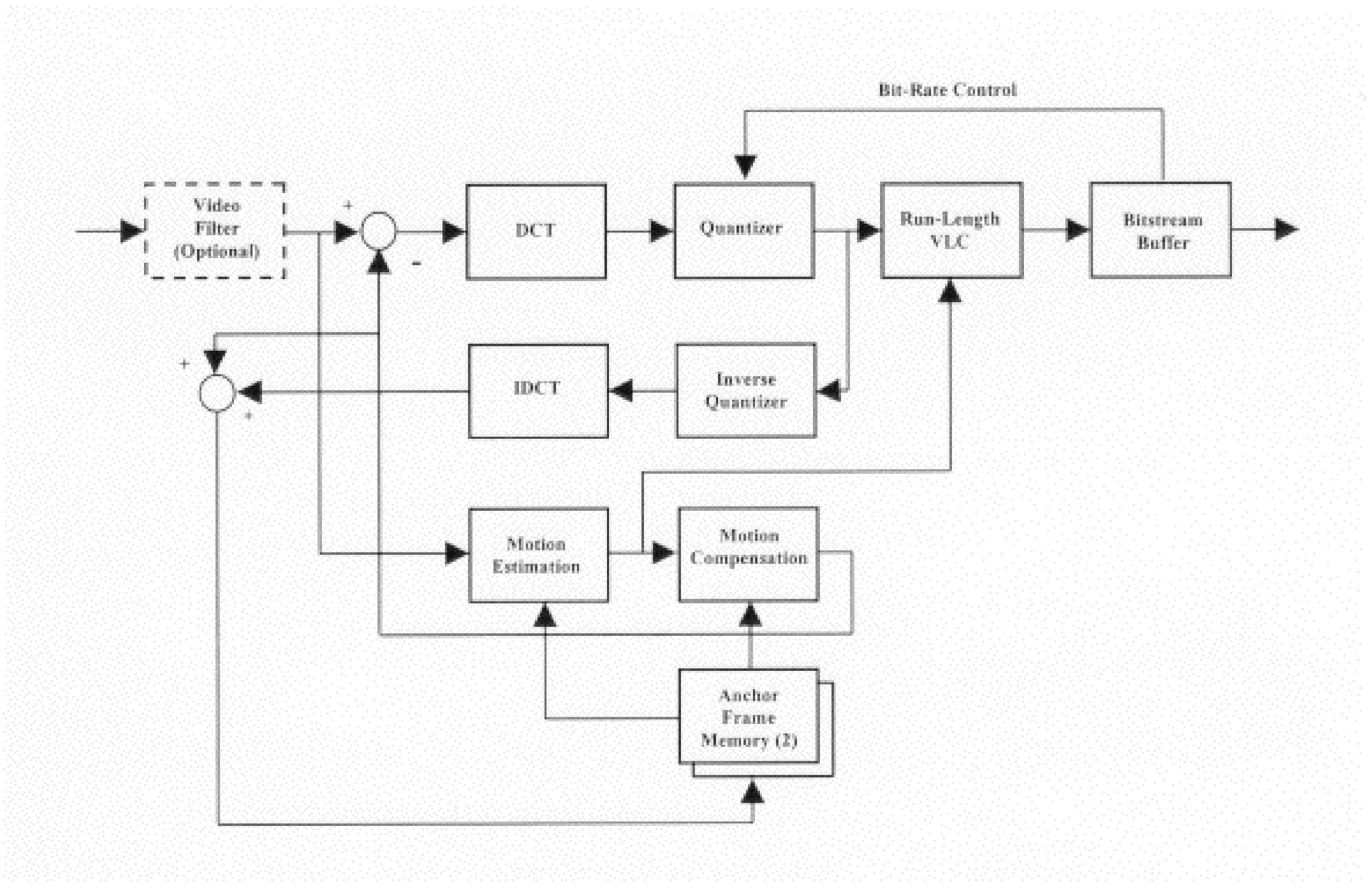
Addr	Type	Quant	Vector	CBP	b0	b1	...	b5

- Many macroblocks will be exact matches (or close enough). So send address of each block in image -> *Addr*
- Sometimes no good match can be found, so send INTRA block -> *Type*
- Will want to vary the quantization to fine tune compression, so send quantization value -> *Quant*
- Motion vector -> *vector*
- Some blocks in macroblock will match well, others match poorly. So send bitmask indicating which blocks are present (Coded Block Pattern, or *CBP*).
- Send the blocks (4 Y, 1 Cr, 1 Cb) as in JPEG.

- Quantization is by constant value for all DCT coefficients (i.e., no quantization table as in JPEG).

Inter-frame (P-frame) Coding

The previously discussed intra frame coding techniques were limited to processing the video signal on a spatial basis, relative only to information within the current video frame. Considerably more compression efficiency can be obtained however, if the inherent temporal, or time-based redundancies, are exploited as well. Anyone who has ever taken a reel of the old-style super-8 movie film and held it up to a light can certainly remember seeing that most consecutive frames within a sequence are very similar to the frames both before and after the frame of interest. Temporal processing to exploit this redundancy uses a technique known as block-based motion compensated prediction, using motion estimation. A block diagram of the basic encoder with extensions for non-intra frame coding techniques is given in Figure 7.14. Of course, this encoder can also support intra frame coding as a subset.

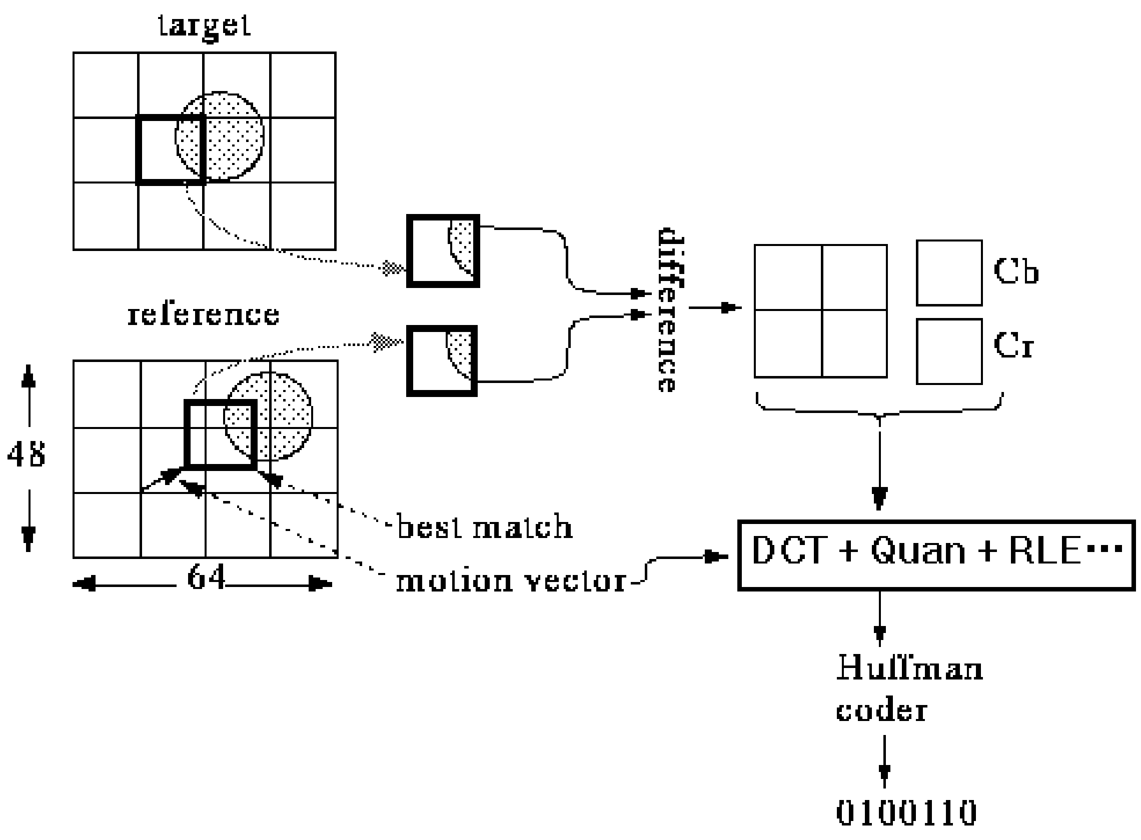


P-Frame Coding

Starting with an intra, or I frame, the encoder can forward predict a future frame. This is commonly referred to as a P frame, and it may also be predicted from other P frames, although only in a forward time manner. As an example, consider a group of pictures that lasts for 6 frames. In this case, the frame ordering is given as I,P,P,P,P,P,I,P,P,P,P,Š

Each P frame in this sequence is predicted from the frame immediately preceding it, whether it is an I frame or a P frame. As a reminder, I frames are coded spatially with no reference to any other frame in the sequence.

P-coding can be summarised as follows:



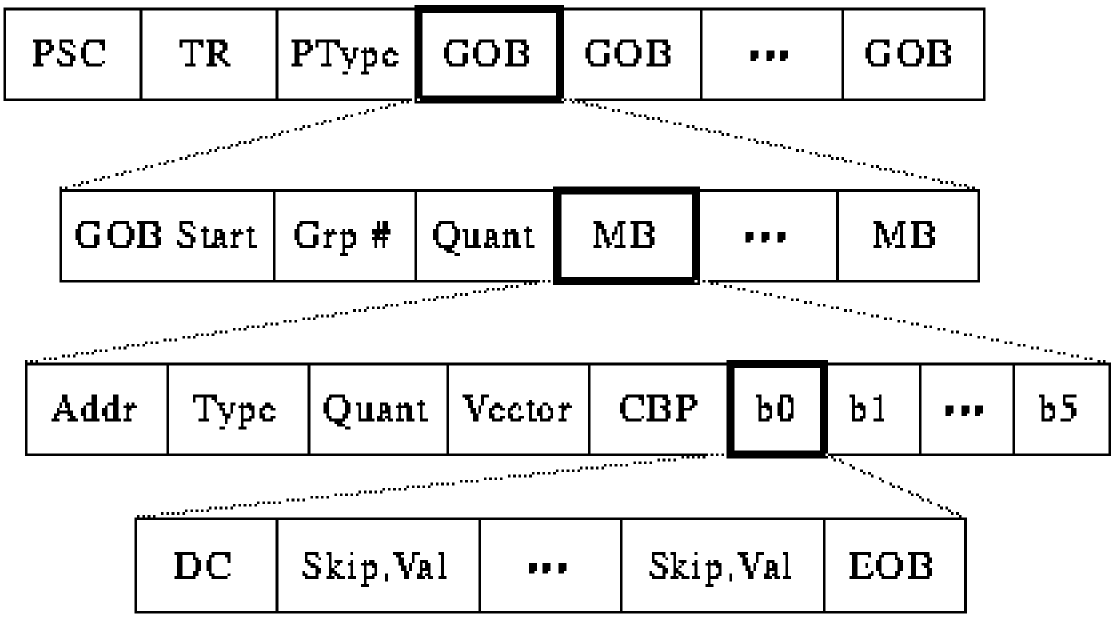
- An Coding Example (P-frame)
- Previous image is called *reference image*.
- Image to code is called *target image*.
- Actually, the difference is encoded.
- Subtle points:

1.
Need to use decoded image as reference image, *not* original. Why?

2.
We're using "Mean Absolute Difference" (MAD) to decide best block. Can also use "Mean Squared Error" (MSE) = sum(E*E)

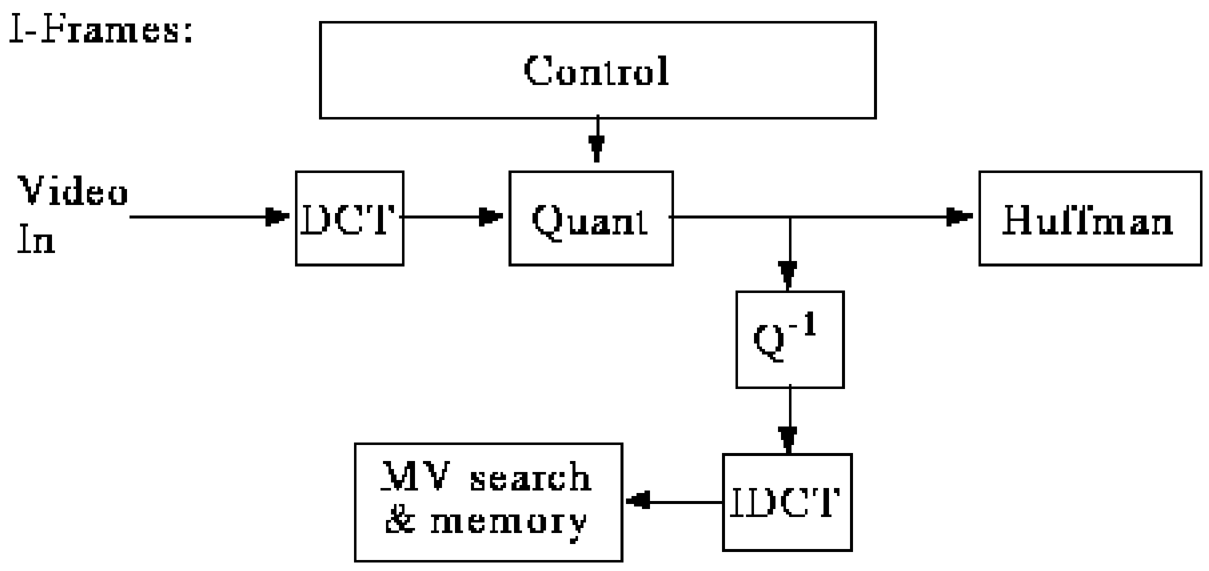
The H.261 Bitstream Structure

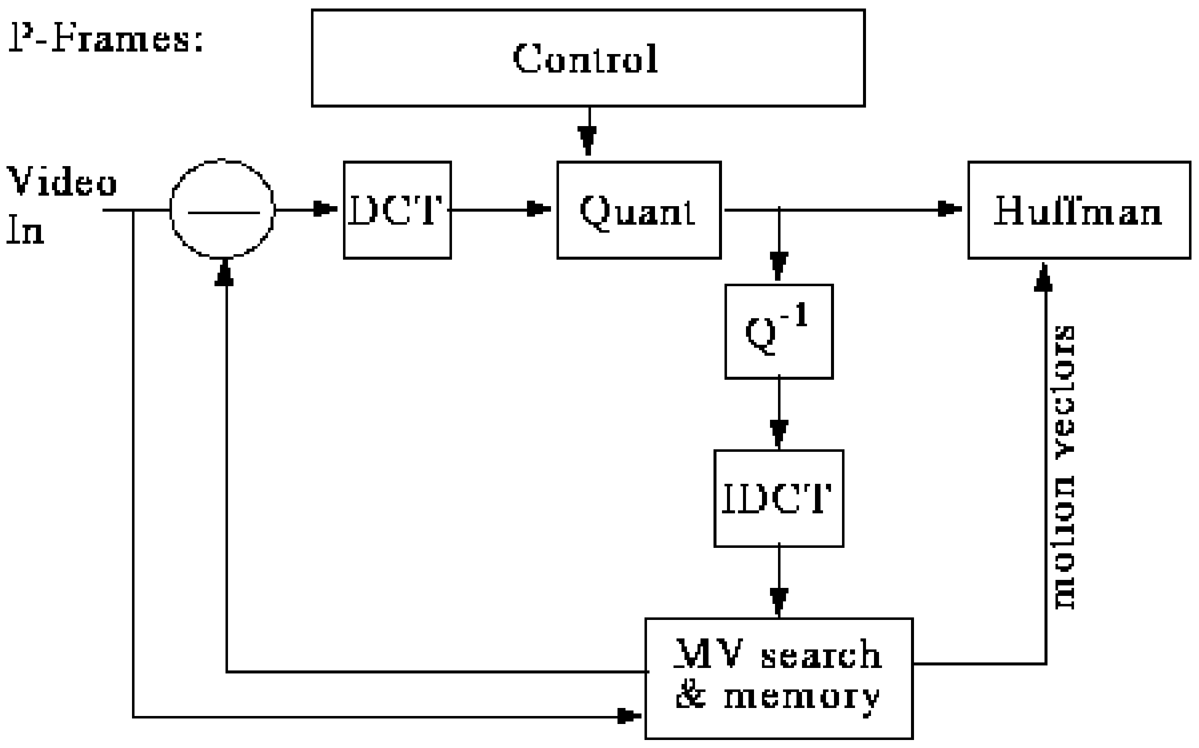
The H.261 Bitstream structure may be summarised as follows:



- Need to delineate boundaries between pictures, so send Picture Start Code -> *PSC*
- Need timestamp for picture (used later for audio synchronization), so send Temporal Reference -> *TR*
- Is this a P-frame or an I-frame? Send Picture Type -> *PType*
- Picture is divided into regions of 11x3 macroblocks called Groups of Blocks -> *GOB*
- Might want to skip whole groups, so send Group Number (*Grp #*)
- Might want to use one quantization value for whole group, so send Group Quantization Value -> *GQuant*
- Overall, bitstream is designed so we can skip data whenever possible while still unambiguous.

The overall H.261 Codec is summarised in Fig 7.8.4.



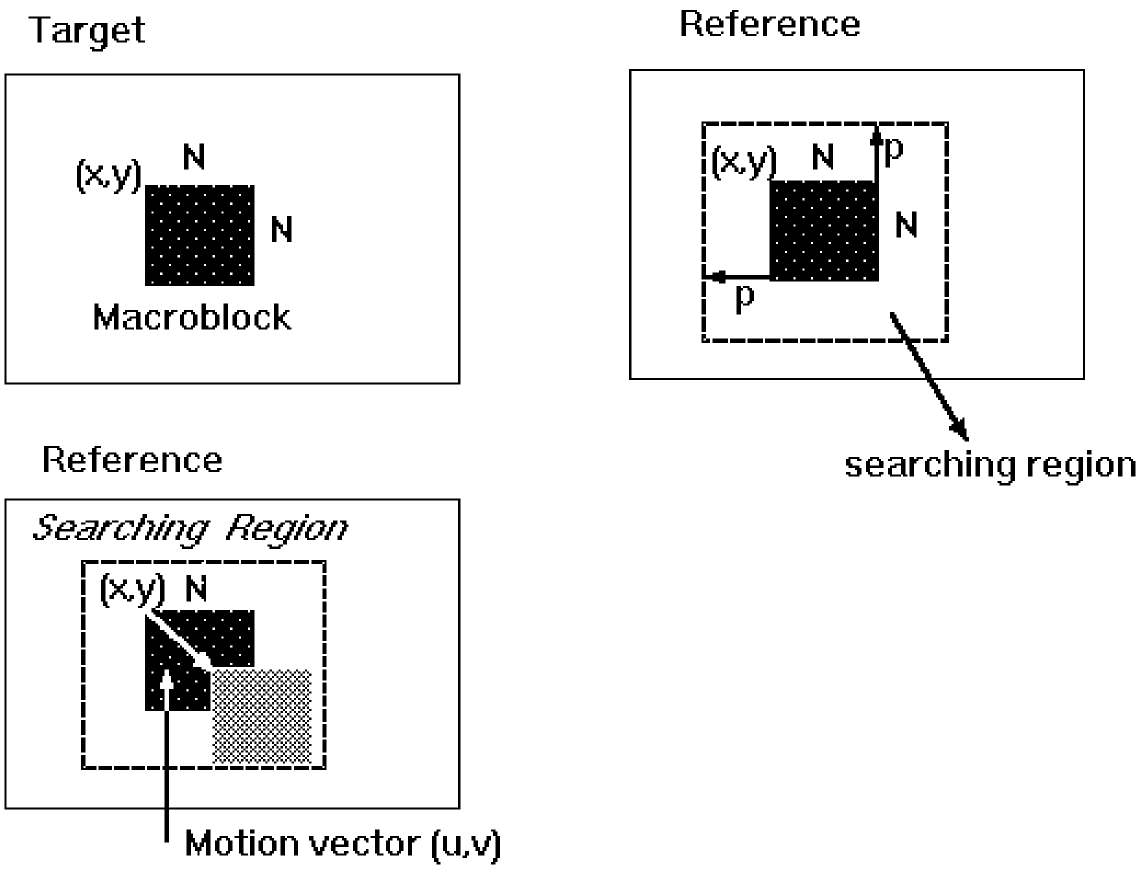


Hard Problems in H.261

There are however a few difficult problems in H.261:

- Motion vector search
- Propagation of Errors
- Bit-rate Control

Motion Vector Search



- $C(x+k,y+i)$ - pixels in the macro block with upper left corner (x,y) in the Target.

  $R(X+i+k,y+j+l)$ - pixels in the macro block with upper left corner $(x+i,y+j)$ in the Reference.

  Cost function is:

$$MAE(i,j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \left| C(x+k, y+l) - R(x+i+k, y+j+l) \right|$$

  Where MAE stands for *Mean Absolute Error*.

- Goal is to find a vector (u, v) such that MAE (u, v) is minimum

- **Full Search Method:**

  1.

  Search the whole $[-p, p]$ searching region.

  2.

  Cost is:

$$\frac{IJF}{N^2}(2p+1)^2 \times N^2 \times 3$$

operations,

assuming that each pixel comparison needs 3 operations (Subtraction, Absolute value, Addition).

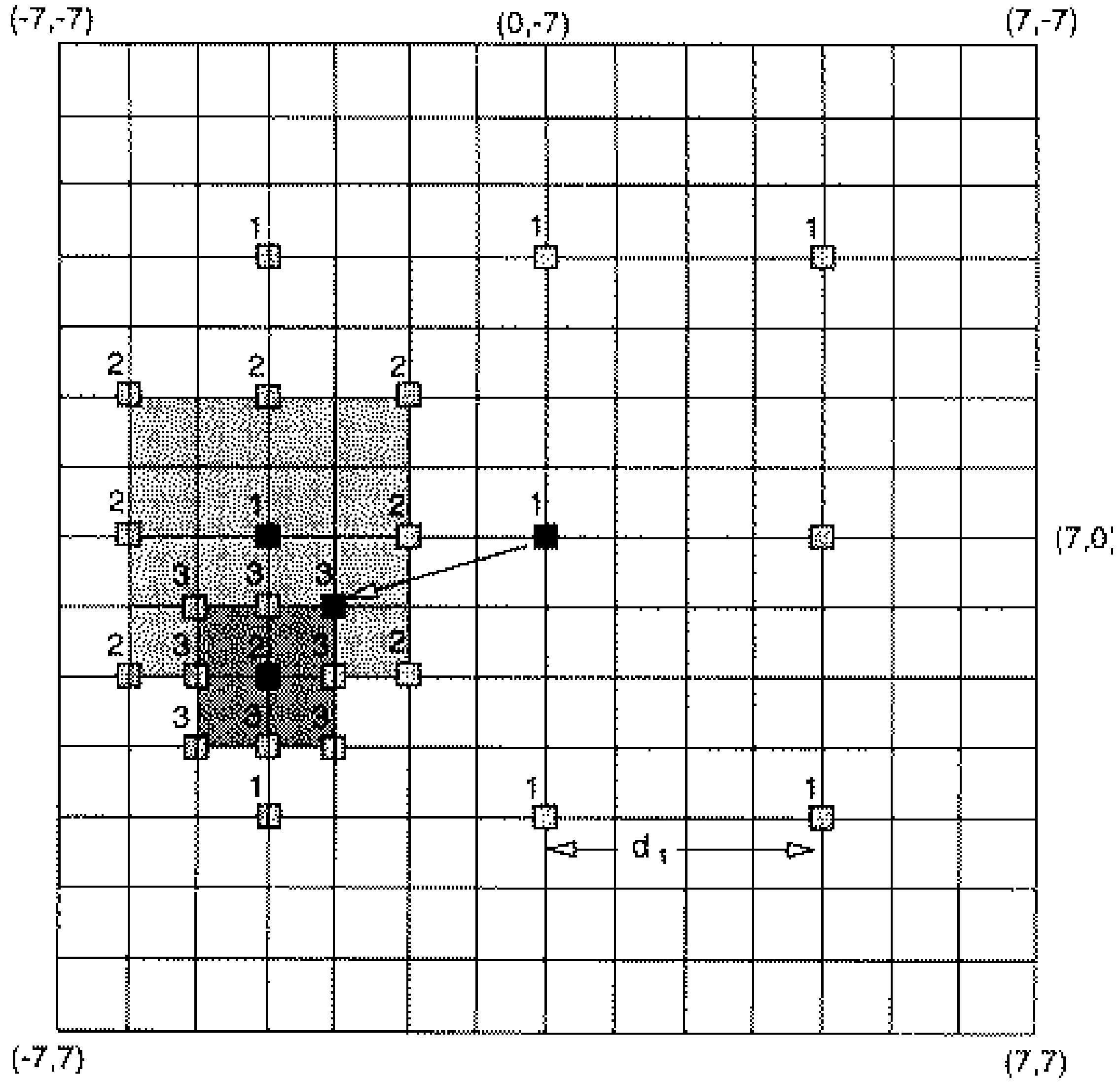
- **Two-Dimensional Logarithmic Search:**

  Similar to binary search. MAE function is initially computed within a window of $[-p/2, p/2]$ at nine locations as shown in the figure.

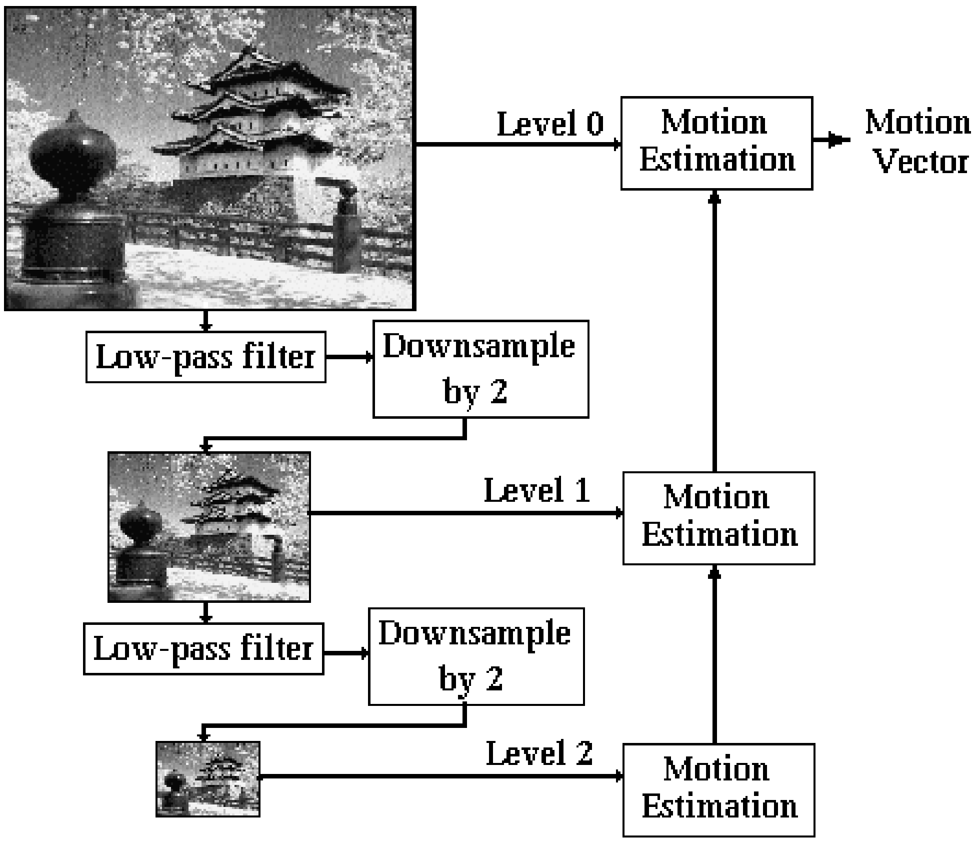
  Repeat until the size of the search region is one pixel wide:

  1. Find one of the nine locations that yields the minimum MAE.

  2. Form a new searching region with half of the previous size and centered at the location found in step 1.



- **Hierarchical Motion Estimation:**



1. Form several low resolution version of the target and reference pictures

2. Find the best match motion vector in the lowest resolution version.

3. Modify the motion vector level by level when going up

- Performance comparison:

Search Method	Operation for 720x480 at 30 fps	
	p = 15	p=7
Full Search	29.89 GOPS	6.99 GOPS
Logarithmic	1.02 GOPS	777.60 MOPS
Hierarchical	507.38 MOPS	398.52 MOPS

Propagation of Errors

- Send an I-frame every once in a while
- Make sure you use decoded frame for comparison

Bit-rate Control

- Simple feedback loop based on "buffer fullness"

  If buffer is too full, increase the quantization scale factor to reduce the data.

MPEG Compression

The acronym MPEG stands for Moving Picture Expert Group, which worked to generate the specifications under ISO, the International Organization for Standardization and IEC, the International Electrotechnical Commission. What is commonly referred to as "MPEG video" actually consists at the present time of two finalized standards, MPEG-11 and MPEG-22, with a third standard, MPEG-4, was finalized in 1998 for ***Very Low Bitrate Audio-Visual Coding***. The MPEG-1 and MPEG-2 standards are similar in basic concepts. They both are based on motion compensated block-based transform coding techniques, while MPEG-4 deviates from these more traditional approaches in its usage of software image construct descriptors, for target bit-rates in the very low range, < 64Kb/sec. Because MPEG-1 and MPEG-2 are finalized standards and are both presently being utilized in a large number of applications, this paper concentrates on compression techniques relating only to these two standards. Note that there is no reference to MPEG-3. This is because it was originally anticipated that this standard would refer to HDTV applications, but it was found that minor extensions to the MPEG-2 standard would suffice for this higher bit-rate, higher resolution application, so work on a separate MPEG-3 standard was abandoned.

The current thrust is MPEG-7 "Multimedia Content Description Interface" whose completion is scheduled for July 2001. Work on the new standard MPEG-21 "Multimedia Framework" has started in June 2000 and has already produced a Draft Technical Report and two Calls for Proposals.

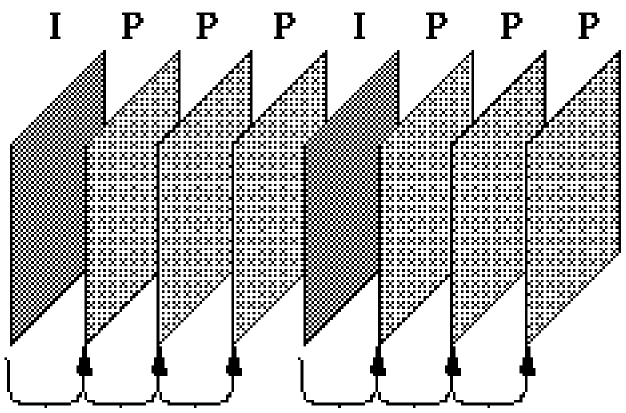
MPEG-1 was finalized in 1991, and was originally optimized to work at video resolutions of 352x240 pixels at 30 frames/sec (NTSC based) or 352x288 pixels at 25 frames/sec (PAL based), commonly referred to as Source Input Format (SIF) video. It is often mistakenly thought that the MPEG-1 resolution is limited to the above sizes, but it in fact may go as high as 4095x4095 at 60 frames/sec. The bit-rate is optimized for applications of around 1.5 Mb/sec, but again can be used at higher rates if required. MPEG-1 is defined for progressive frames only, and has no direct provision for interlaced video applications, such as in broadcast television applications.

MPEG-2 was finalized in 1994, and addressed issues directly related to digital television broadcasting, such as the efficient coding of field-interlaced video and scalability. Also, the target bit-rate was raised to between 4 and 9 Mb/sec, resulting in potentially very high quality video. MPEG-2 consists of profiles and levels. The profile defines the bitstream scalability and the colorspace resolution, while the level defines the image resolution and the maximum bit-rate per profile. Probably the most common descriptor in use currently is Main Profile, Main Level (MP@ML) which refers to 720x480 resolution video at 30 frames/sec, at bit-rates up to 15 Mb/sec for NTSC video. Another example is the HDTV resolution of 1920x1080 pixels at 30 frame/sec, at a bit-rate of up to 80 Mb/sec. This is an example of the Main Profile, High Level (MP@HL) descriptor. A complete table of the various legal combinations can be found in reference2.

MPEG Video

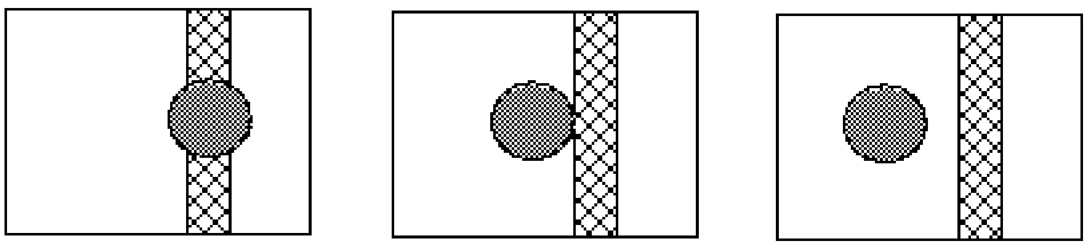
MPEG compression is essentially a attempts to over come some shortcomings of H.261 and JPEG:

- Recall H.261 dependencies:



- The Problem here is that many macroblocks need information is **not** in the reference frame.

- For example:

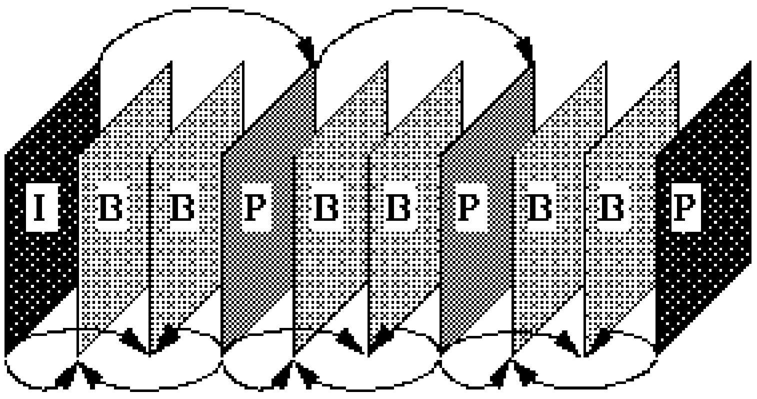


- The ***MPEG solution*** is to add a third frame type which is a bidirectional frame, or *B-frame*

- B-frames search for macroblock in *past* and *future* frames.

- Typical pattern is IBBPBBPBB IBBPBBPBB IBBPBBPBB

  Actual pattern is up to encoder, and need not be regular.

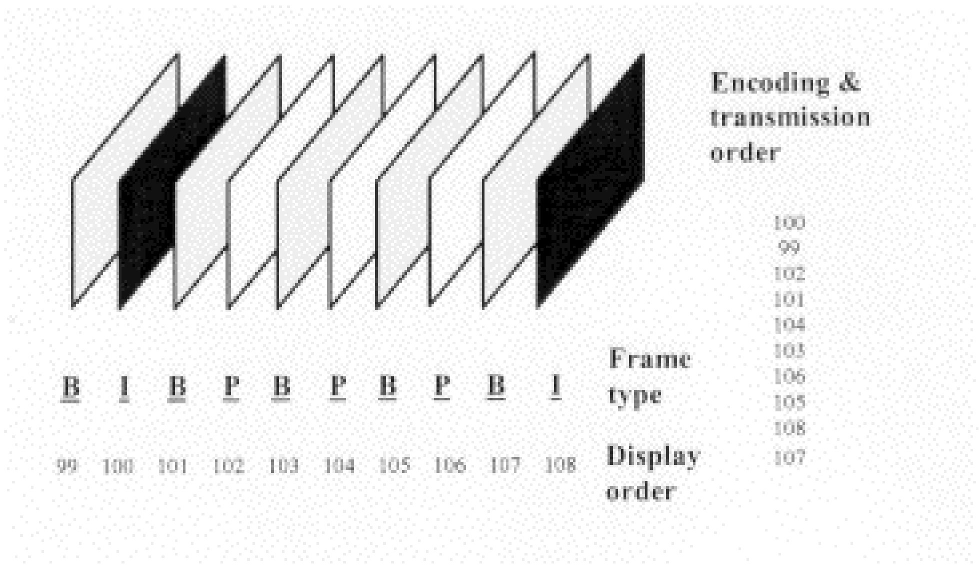

I
B
B
P
B
B
P
B
B
P

MPEG Video Layers

MPEG video is broken up into a hierarchy of layers to help with error handling, random search and editing, and synchronization, for example with an audio bitstream. From the top level, the first layer is known as the video sequence layer, and is any self-contained bitstream, for example a coded movie or advertisement. The second layer down is the group of pictures, which is composed of 1 or more groups of intra (I) frames and/or non-intra (P and/or B) pictures that will be defined later. Of course the third layer down is the picture layer itself, and the next layer beneath it is called the slice layer. Each slice is a contiguous sequence of raster ordered macroblocks, most often on a row basis in typical video applications, but not limited to this by the specification. Each slice consists of macroblocks, which are 16x16 arrays of luminance pixels, or picture data elements, with 2 8x8 arrays of associated chrominance pixels. The macroblocks can be further divided into distinct 8x8 blocks, for further processing such as transform coding. Each of these layers has its own unique 32 bit start code defined in the syntax to consist of 23 zero bits followed by a one, then followed by 8 bits for the actual start code. These start codes may have as many zero bits as desired preceding them.

B-Frames

The MPEG encoder also has the option of using forward/backward interpolated prediction. These frames are commonly referred to as bi-directional interpolated prediction frames, or B frames for short. As an example of the usage of I, P, and B frames, consider a group of pictures that lasts for 6 frames, and is given as I,B,P,B,P,B,I,B,P,B,P,B,Š As in the previous I and P only example, I frames are coded spatially only and the P frames are forward predicted based on previous I and P frames. The B frames however, are coded based on a forward prediction from a previous I or P frame, as well as a backward prediction from a succeeding I or P frame. As such, the example sequence is processed by the encoder such that the first B frame is predicted from the first I frame and first P frame, the second B frame is predicted from the second and third P frames, and the third B frame is predicted from the third P frame and the first I frame of the next group of pictures. From this example, it can be seen that backward prediction requires that the future frames that are to be used for backward prediction be encoded and transmitted first, out of order. This process is summarized in Figure 7.16. There is no defined limit to the number of consecutive B frames that may be used in a group of pictures, and of course the optimal number is application dependent. Most broadcast quality applications however, have tended to use 2 consecutive B frames (I,B,B,P,B,B,P,Š) as the ideal trade-off between compression efficiency and video quality.



B-Frame Encoding

The main advantage of the usage of B frames is coding efficiency. In most cases, B frames will result in less bits being coded overall. Quality can also be improved in the case of moving objects that reveal hidden areas within a video sequence. Backward prediction in this case allows the encoder to make more intelligent decisions on how to encode the video within these areas. Also, since B frames are not used to predict future frames, errors generated will not be propagated further within the sequence.

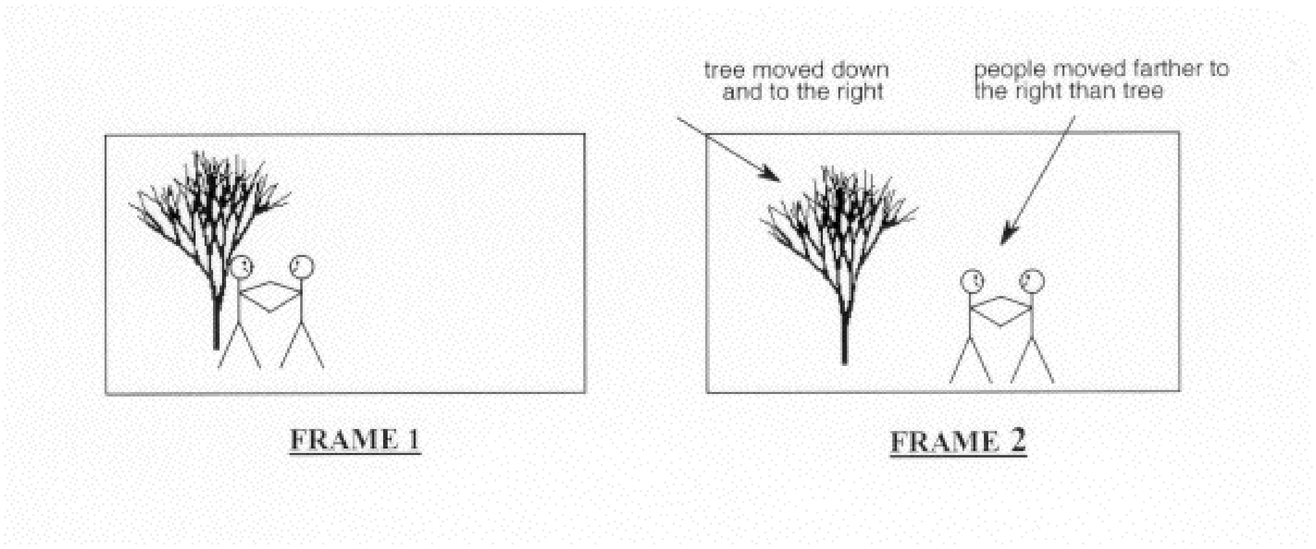
One disadvantage is that the frame reconstruction memory buffers within the encoder and decoder must be doubled in size to accommodate the 2 anchor frames. This is almost never an issue for the relatively expensive encoder, and in these days of inexpensive DRAM it has become much less of an issue for the decoder as well. Another disadvantage is that there will necessarily be a delay throughout the system as the frames are delivered out of order as was shown in Figure . Most one-way systems can tolerate these delays, as they are more objectionable in applications such as video conferencing systems.

Dave Marshall
10/4/2001

Motion Estimation

The temporal prediction technique used in MPEG video is based on motion estimation. The basic premise of motion estimation is that in most cases, consecutive video frames will be similar except for changes induced by objects moving within the frames. In the trivial case of zero motion between frames (and no other differences caused by noise, etc.), it is easy for the encoder to efficiently predict the current frame as a duplicate of the prediction frame. When this is done, the only information necessary to transmit to the decoder becomes the syntactic overhead necessary to reconstruct the picture from the original reference frame. When there is motion in the images, the situation is not as simple.

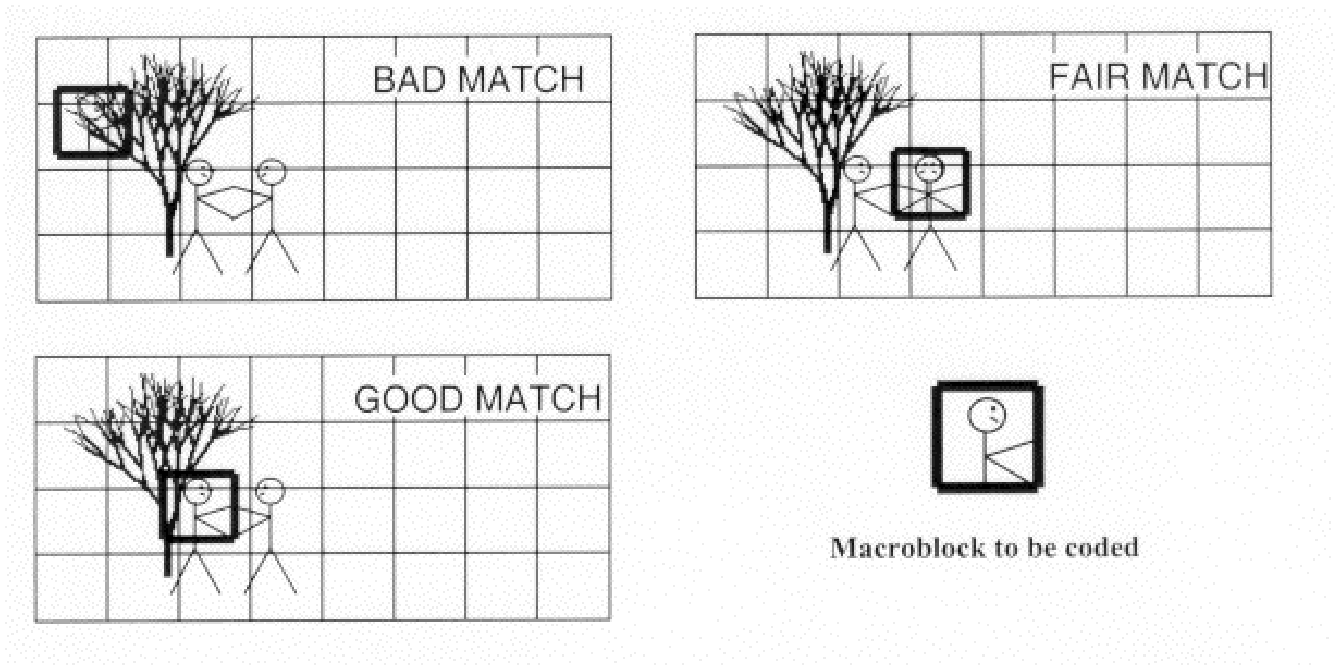
Figure 7.17 shows an example of a frame with 2 stick figures and a tree. The second half of this figure is an example of a possible next frame, where panning has resulted in the tree moving down and to the right, and the figures have moved farther to the right because of their own movement outside of the panning. The problem for motion estimation to solve is how to adequately represent the changes, or differences, between these two video frames.


tree moved down and to the right
people moved farther to the right than tree
FRAME 1
FRAME 2

Motion Estimation Example

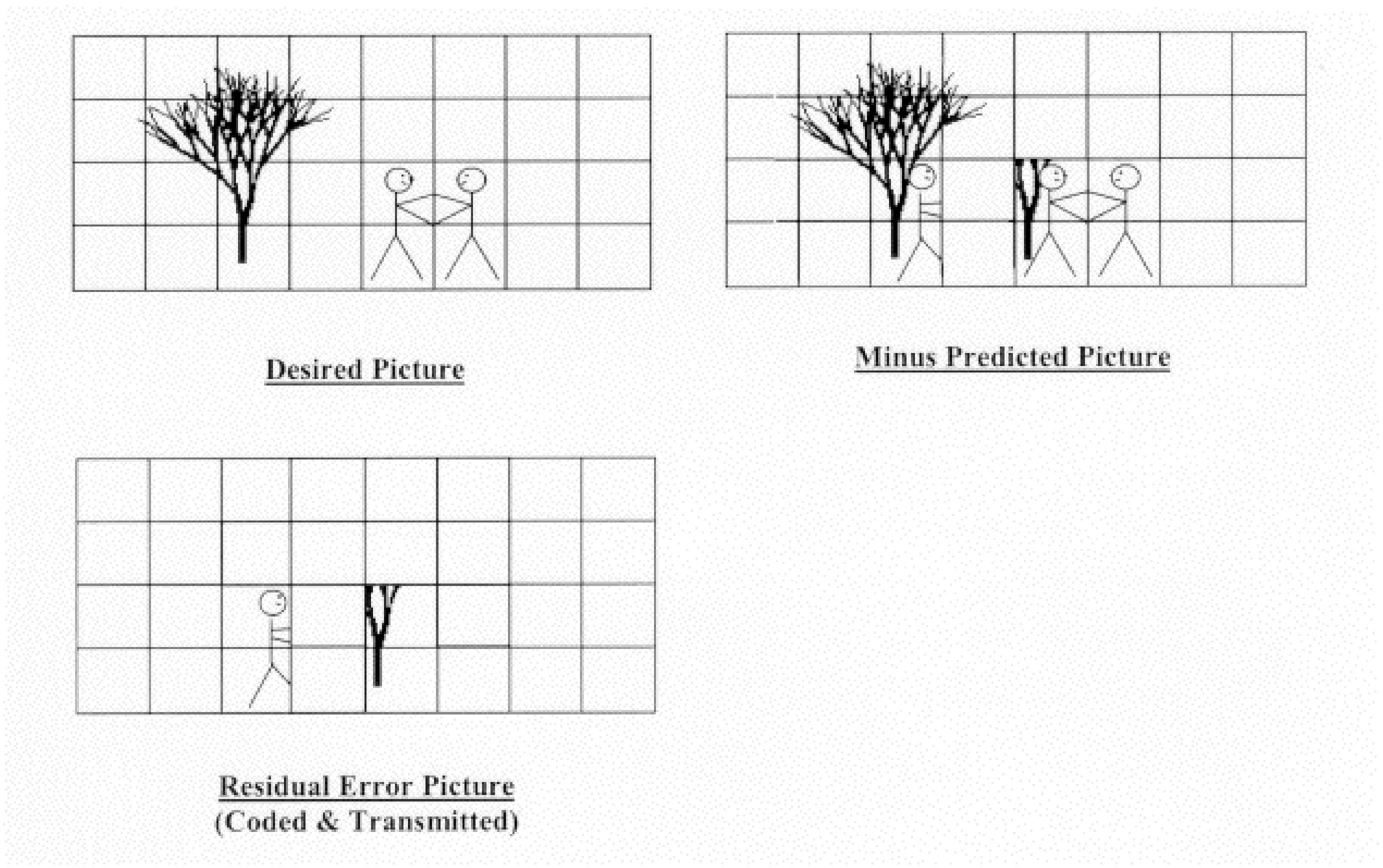
The way that motion estimation goes about solving this problem is that a comprehensive 2-dimensional spatial search is performed for each luminance macroblock. Motion estimation is not applied directly to chrominance in MPEG video, as it is assumed that the color motion can be adequately represented with the same motion information as the luminance. It should be noted at this point that MPEG does not define how this search should be performed. This is a detail that the system designer can choose to implement in one of many possible ways. This is similar to the bit-rate control algorithms discussed previously, in the respect that complexity vs. quality issues need to be addressed relative to the individual application. It is well known that a full, exhaustive search over a wide 2-dimensional area yields the best matching results in most cases, but this performance comes at an extreme computational cost to the encoder. As motion estimation usually is the most computationally expensive portion of the video encoder, some lower cost encoders might choose to limit the pixel search range, or use other techniques such as telescopic searches, usually at some cost to the video quality.

Figure 7.18 shows an example of a particular macroblock from Frame 2 of Figure 7.17, relative to various macroblocks of Frame 1. As can be seen, the top frame has a bad match with the macroblock to be coded. The middle frame has a fair match, as there is some commonality between the 2 macroblocks. The bottom frame has the best match, with only a slight error between the 2 macroblocks. Because a relatively good match has been found, the encoder assigns motion vectors to the macroblock, which indicate how far horizontally and vertically the macroblock must be moved so that a match is made. As such, each forward and backward predicted macroblock may contain 2 motion vectors, so true bidirectionally predicted macroblocks will utilize 4 motion vectors.



Motion Estimation Macroblock Example

Figure 7.19 shows how a potential predicted Frame 2 can be generated from Frame 1 by using motion estimation. In this figure, the predicted frame is subtracted from the desired frame, leaving a (hopefully) less complicated residual error frame that can then be encoded much more efficiently than before motion estimation. It can be seen that the more accurate the motion is estimated and matched, the more likely it will be that the residual error will approach zero, and the coding efficiency will be highest. Further coding efficiency is accomplished by taking advantage of the fact that motion vectors tend to be highly correlated between macroblocks. Because of this, the horizontal component is compared to the previously valid horizontal motion vector and only the difference is coded. This same difference is calculated for the vertical component before coding. These difference codes are then described with a variable length code for maximum compression efficiency.



Final Motion Estimation Prediction

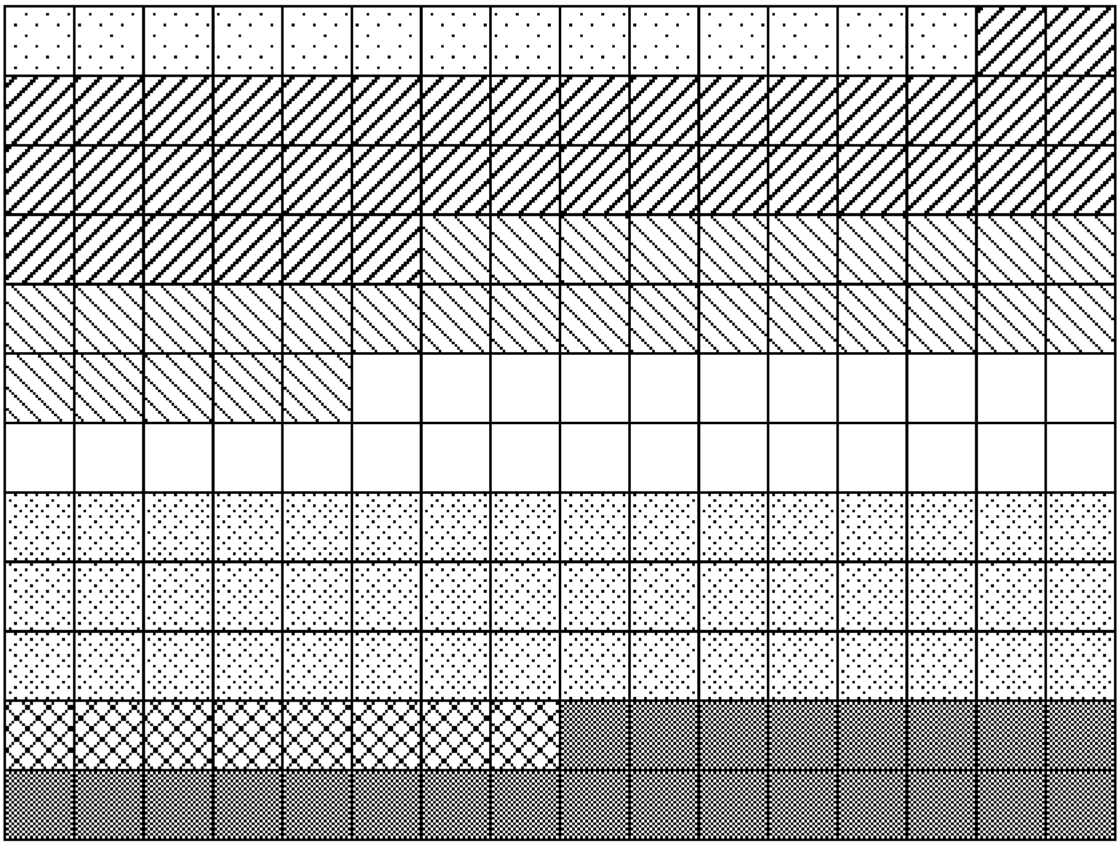
Of course not every macroblock search will result in an acceptable match. If the encoder decides that no acceptable match exists (again, the "acceptable" criterion is not MPEG defined, and is up to the system designer) then it has the option of coding that particular macroblock as an intra macroblock, even though it may be in a P or B frame. In this manner, high quality video is maintained at a slight cost to coding efficiency.

Coding of Predicted Frames:Coding Residual Errors

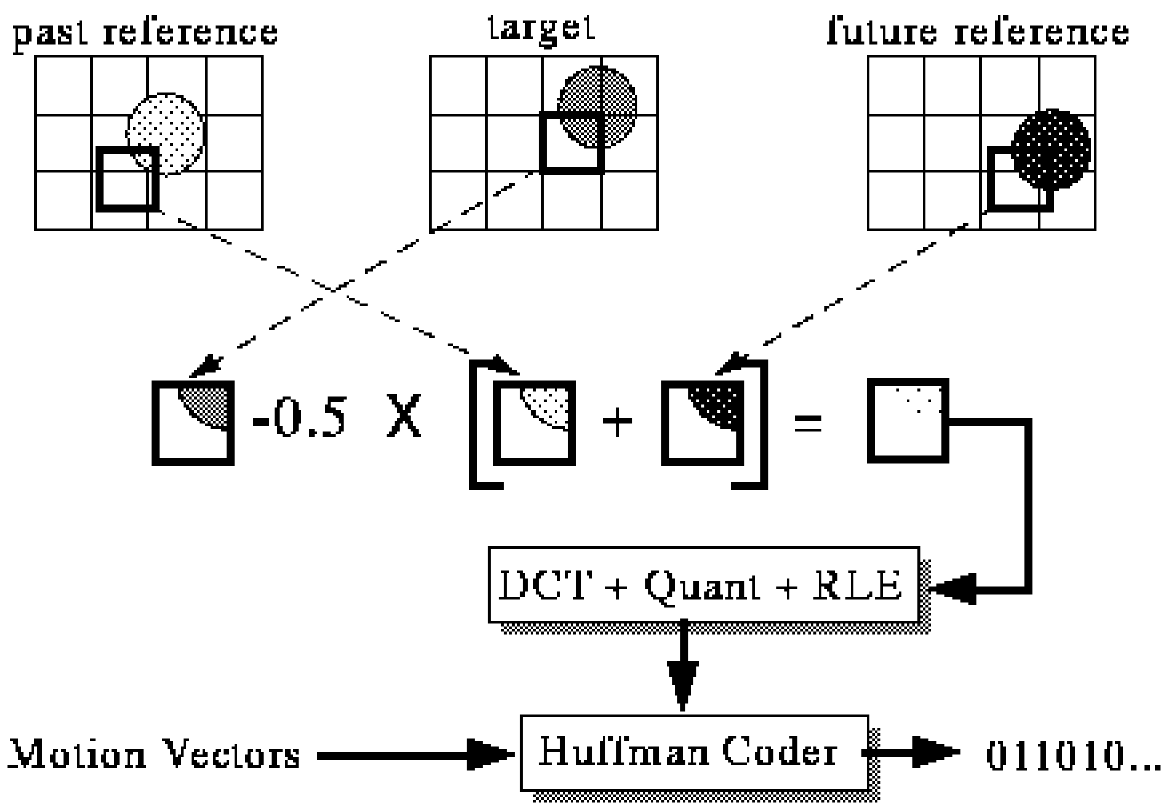
After a predicted frame is subtracted from its reference and the residual error frame is generated, this information is spatially coded as in I frames, by coding 8x8 blocks with the DCT, DCT coefficient quantization, run-length/amplitude coding, and bitstream buffering with rate control feedback. This process is basically the same with some minor differences, the main ones being in the DCT coefficient quantization. The default quantization matrix for non-intra frames is a flat matrix with a constant value of 16 for each of the 64 locations. This is very different from that of the default intra quantization matrix which is tailored for more quantization in direct proportion to higher spatial frequency content. As in the intra case, the encoder may choose to override this default, and utilize another matrix of choice during the encoding process, and download it via the encoded bitstream to the decoder on a picture basis. Also, the non-intra quantization step function contains a dead-zone around zero that is not present in the intra version. This helps eliminate any lone DCT coefficient quantization values that might reduce the run-length amplitude efficiency. Finally, the motion vectors for the residual block information are calculated as differential values and are coded with a variable length code according to their statistical likelihood of occurrence.

Differences from H.261

- Larger gaps between I and P frames, so expand motion vector search range.
- To get better encoding, allow motion vectors to be specified to fraction of a pixel (1/2 pixels).
- Bitstream syntax must allow random access, forward/backward play, etc.
- Added notion of *slice* for synchronization after loss/corrupt data. Example: picture with 7 slices:



- B frame macroblocks can specify *two* motion vectors (one to past and one to future), indicating result is to be averaged.



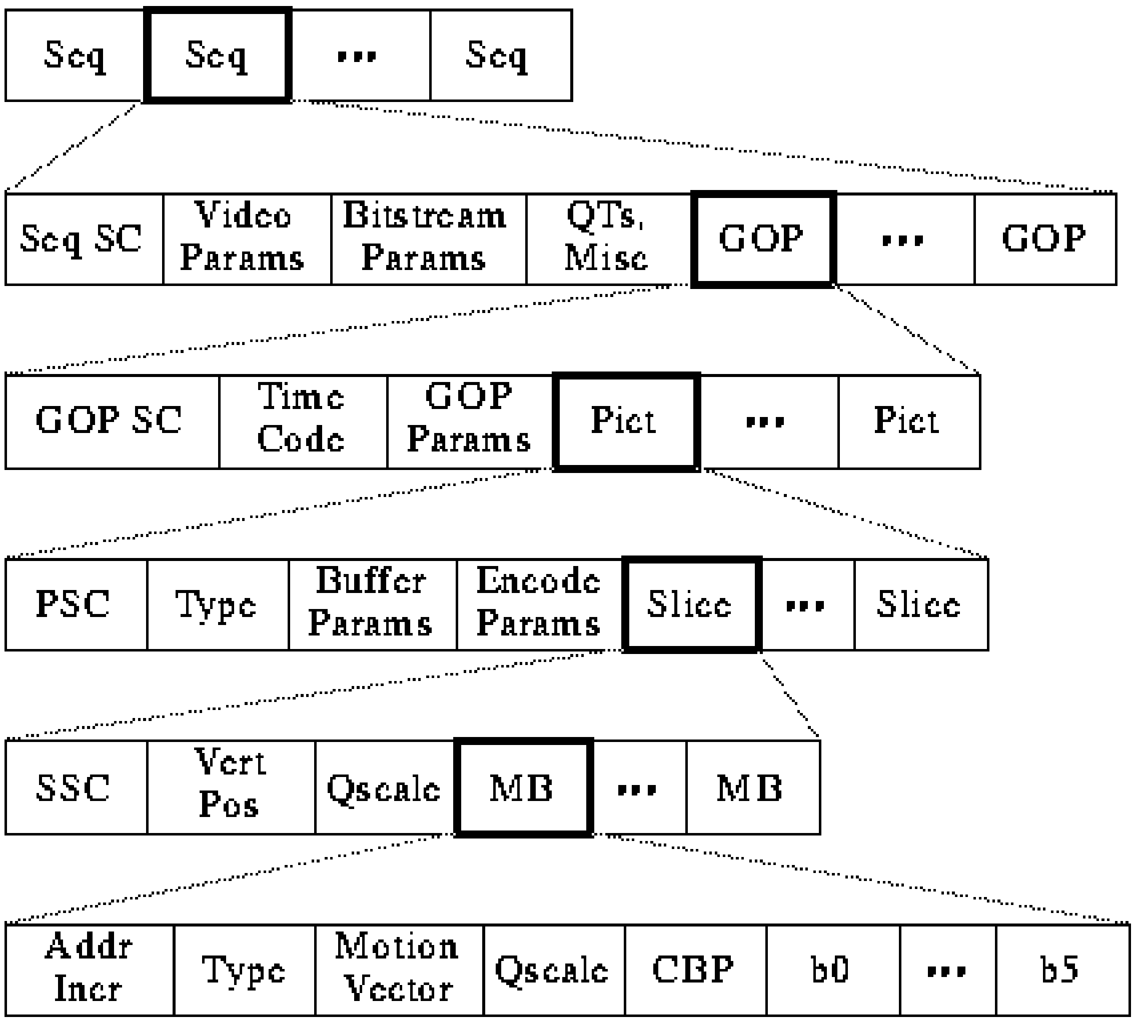
- Compression performance of MPEG 1

```
------------------------------
Type      Size      Compression
------------------------------
 I      18   KB        7:1
 P       6   KB       20:1
 B       2.5 KB       50:1
Avg      4.8 KB       27:1
------------------------------
```

The MPEG Video Bitstream

The MPEG Video Bitstream is summarised as follows:

- Public domain tool **mpeg_stat** and **mpeg_bits** will analyze a bitstream.



- Sequence Information

1.

Video Params include width, height, aspect ratio of pixels, picture rate.

2.

Bitstream Params are bit rate, buffer size, and constrained parameters flag (means bitstream can be decoded by most hardware)

3. Two types of QTs: one for intra-coded blocks (I-frames) and one for inter-coded blocks (P-frames).

- Group of Pictures (GOP) information

  1. *Time code*: bit field with SMPTE time code (hours, minutes, seconds, frame).

  2. *GOP Params* are bits describing structure of GOP. Is GOP *closed*? Does it have a dangling pointer *broken*?

- Picture Information

  1. *Type*: I, P, or B-frame?

  2. *Buffer Params* indicate how full decoder's buffer should be before starting decode.

  3. *Encode Params* indicate whether half pixel motion vectors are used.

- Slice information

  1. *Vert Pos*: what line does this slice start on?

  2. *QScale*: How is the quantization table scaled in this slice?

- Macroblock information

  1. *Addr Incr*: number of MBs to skip.

  2. *Type*: Does this MB use a motion vector? What type?

  3. *QScale*: How is the quantization table scaled in this MB?

4.
Coded Block Pattern (CBP): bitmap indicating which blocks are coded.

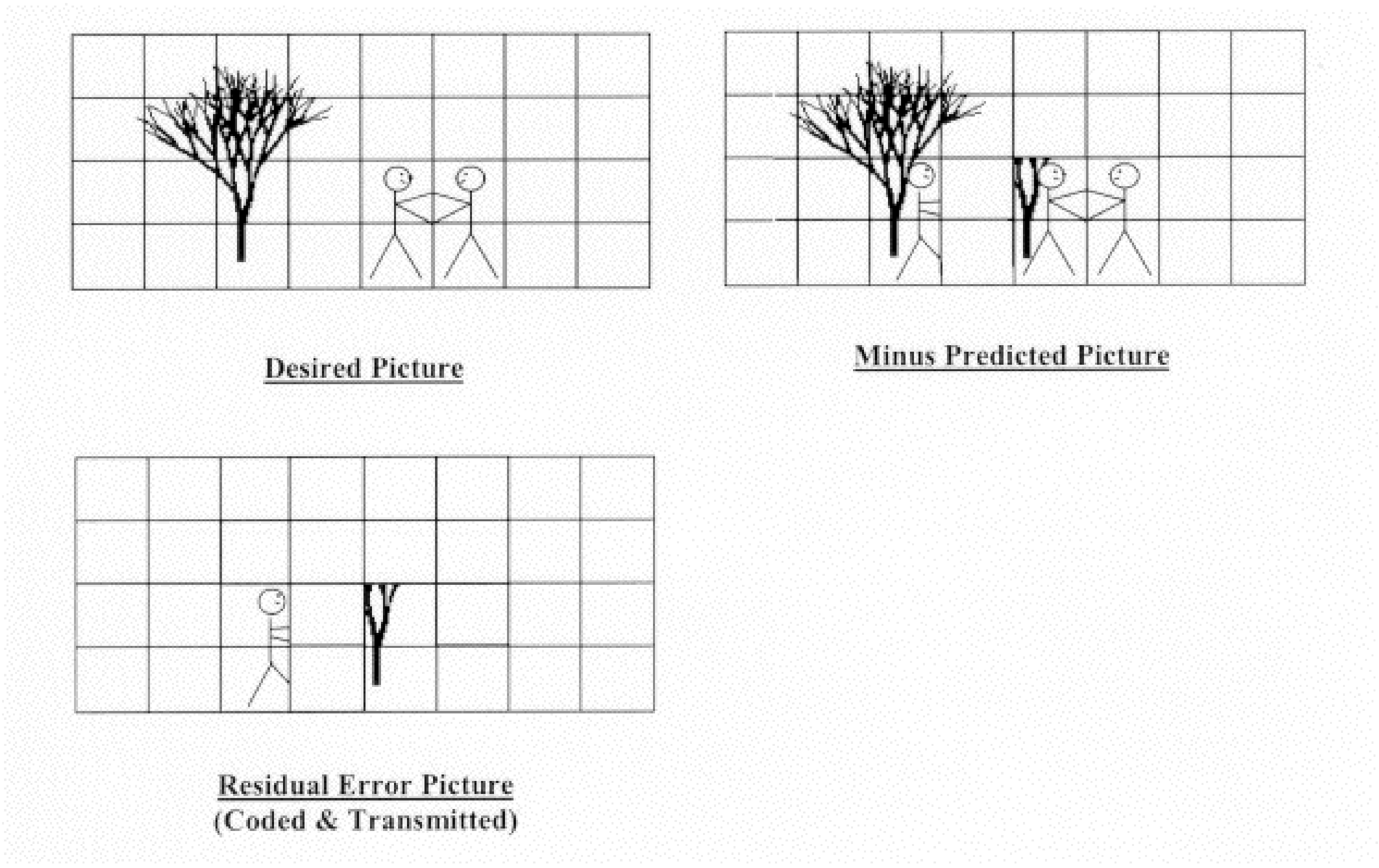
Decoding MPEG Video in Software

- Software Decoder goals: portable, multiple display types
- Breakdown of time

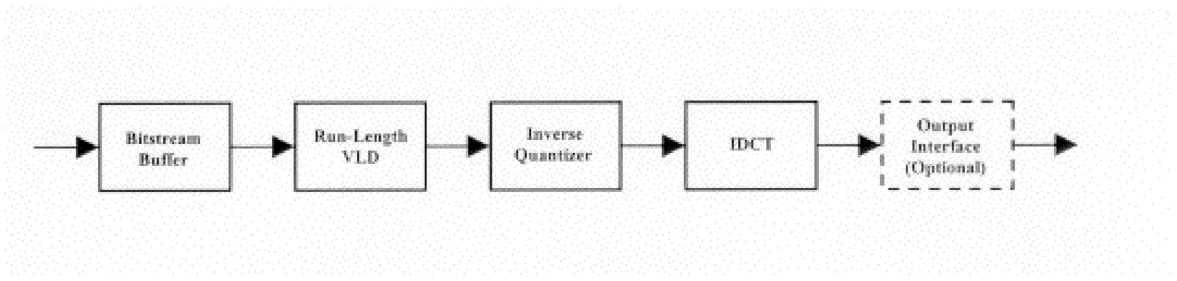
Function	% Time
Parsing Bitstream	17.4%
IDCT	14.2%
Reconstruction	31.5%
Dithering	24.5%
Misc. Arith.	9.9%
Other	2.7%

Intra Frame Decoding

To decode a bitstream generated from the encoder of Figure 7.20, it is necessary to reverse the order of the encoder processing. In this manner, an I frame decoder consists of an input bitstream buffer, a Variable Length Decoder (VLD), an inverse quantizer, an Inverse Discrete Cosine Transform (IDCT), and an output interface to the required environment (computer hard drive, video frame buffer, etc.). This decoder is shown in Figure .



Intra Frame Encoding



Intra Frame Decoding

The input bitstream buffer consists of memory that operates in the inverse fashion of the buffer in the encoder. For fixed bit-rate applications, the constant rate bitstream is buffered in the memory and read out at a variable rate depending on the coding efficiency of the macroblocks and frames to be decoded.

The VLD is probably the most computationally expensive portion of the decoder because it must operate on a bit-wise basis (VLD decoders need to look at every bit, because the boundaries between variable length codes are random and non-aligned) with table look-ups performed at speeds up to the input bit-rate. This is generally the only function in the receiver that is more complex to implement than its corresponding function within the encoder, because of the extensive high-speed bit-wise processingnecessary.

The inverse quantizer block multiplies the decoded coefficients by the corresponding values of the quantization matrix and the quantization scale factor. Clipping of the resulting coefficients is performed to the region -2048 to +2047, then an IDCT mismatch control is applied to prevent long term error propagation within the sequence.

The IDCT operation is given in Equation 2, and is seen to be similar to the DCT operation of Equation 1. As such, these two operations are very similar in implementation between encoder and decoder.

Next Up Previous
Next: MPEG-2, MPEG-3, and MPEG-4 **Up:** Decoding MPEG Video in **Previous:** Intra Frame Decoding

Non-Intra Frame Decoding

It was shown previously that the non-intra frame encoder built upon the basic building blocks of the intra frame encoder, with the addition of motion estimation and its associated support structures. This is also true of the non-intra frame decoder, as it contains the same core structure as the intra frame decoder with the addition of motion compensation support. Again, support for intra frame decoding is inherent in the structure, so I, P, and B frame decoding is possible. The decoder is shown in Figure 24.

Non-Intra Frame Decoding

MPEG-2, MPEG-3, and MPEG-4

- MPEG-2 target applications

Level	size	Pixels/sec	bit-rate (Mbits)	Application
Low	352 x 240	3 M	4	consumer tape equiv.
Main	720 x 480	10 M	15	studio TV
High 1440	1440 x 1152	47 M	60	consumer HDTV
High	1920 x 1080	63 M	80	film production

- Differences from MPEG-1

  1. Search on fields, not just frames.

  2. 4:2:2 and 4:4:4 macroblocks

  3. Frame sizes as large as 16383 x 16383

  4. Scalable modes: Temporal, Progressive,...

  5. Non-linear macroblock quantization factor

  6. A bunch of minor fixes (see MPEG FAQ for more details)

- MPEG-3: Originally for HDTV (1920 x 1080), got folded into MPEG-2

- MPEG-4: Originally targeted at very low bit-rate communication (4.8 to 64 kb/sec). Now addressing video processing...

Further Reading/Information

- G.K. Wallace, *The JPEG Still Picture Compression Standard*
- CCITT, *Recommendation H.261*
- D. Le Gall, *MPEG: A Video Compression Standard for Multimedia Applications*
- K. Patel, et. al., *Performance of a Software MPEG Video Decoder*
- P. Cosman, et. al., *Using Vector Quantization for Image Processing*
- "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s," ISO/IEC 11172-2: Video (November 1991).
- "Generic Coding of Moving Pictures and Associated Audio Information: Video," ISO/IEC 13818-2 : Draft International Standard (November 1994).
- Barry G. Haskell, Atul Puri, Arun N. Netravali, Digital Video: An Introduction to MPEG-2, Chapman and Hall, 1997.
- K.R. Rao, P. Yip, Discrete Cosine Transform - Algorithms, Advantages, Applications, Academic Press, Inc., 1990.
- Majid Rabbani, Paul W. Jones, Digital Image Compression Techniques, SPIE Optical Engineering Press, 1991.
- Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, Didier J. LeGall, MPEG Video Compression Standard, Chapman and Hall, 1997.
- IEEE Micro Magazine - Media Processing, IEEE Computer Society, Volume 16 Number 4, August 1996.
- MPEG Resources on the Web.
- The Official MPEG Committee

Audio Compression

As with video a number of compression techniques have been applied to audio.

Simple Audio Compression Methods

Traditional lossless compression methods (Huffman, LZW, etc.) usually don't work well on audio compression (the same reason as in image compression).

The following are some of the Lossy methods applied to audio compression:

- Silence Compression - detect the "silence", similar to run-length coding
- Adaptive Differential Pulse Code Modulation (ADPCM)

  e.g., in CCITT G.721 - 16 or 32 Kbits/sec.

  (a) encodes the difference between two consecutive signals,

  (b) adapts at quantization so fewer bits are used when the value is smaller.

  - It is necessary to predict where the waveform is headed -> difficult
  - Apple has proprietary scheme called ACE/MACE. Lossy scheme that tries to predict where wave will go in next sample. About 2:1 compression.
- Linear Predictive Coding (LPC) fits signal to speech model and then transmits parameters of model. Sounds like a computer talking, 2.4 kbits/sec.
- Code Excited Linear Predictor (CELP) does LPC, but also transmits error term - audio conferencing quality at 4.8 kbits/sec.

Psychoacoustics

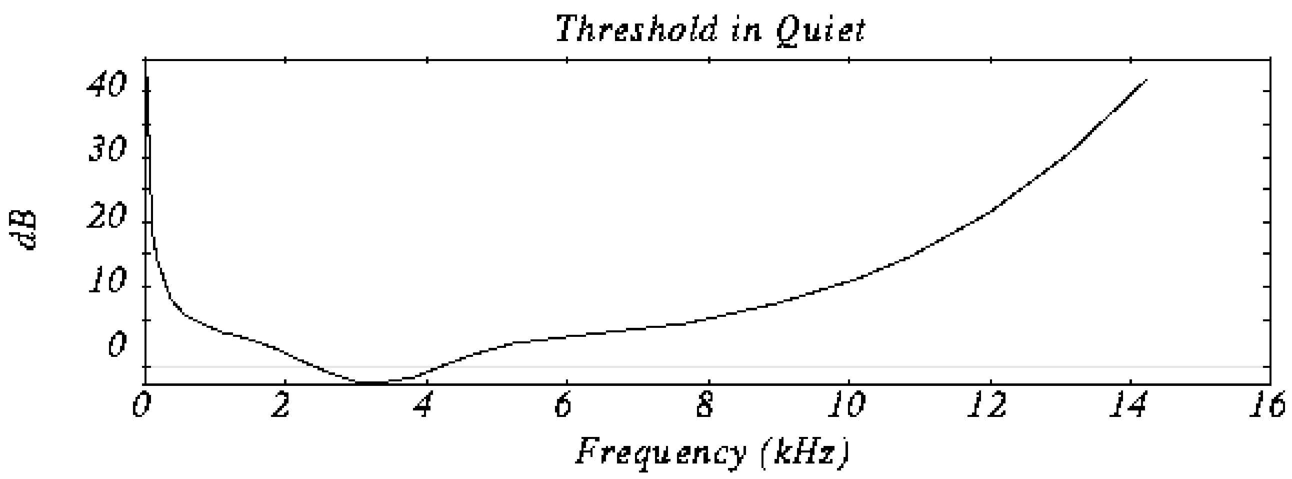
These methods are related to how humans actually hear sounds:

Human hearing and voice

- Range is about 20 Hz to 20 kHz, most sensitive at 2 to 4 KHz.
- Dynamic range (quietest to loudest) is about 96 dB
- Normal voice range is about 500 Hz to 2 kHz
    - Low frequencies are vowels and bass
    - High frequencies are consonants

Question: How sensitive is human hearing?

- Experiment: Put a person in a quiet room. Raise level of 1 kHz tone until just barely audible. Vary the frequency and plot



Frequency Masking

Question: Do receptors interfere with each other?

- Experiment: Play 1 kHz tone (*maskingtone*) at fixed level (60 dB). Play *test tone* at a different level (e.g., 1.1kHz), and raise level until just distinguishable.
- Vary the frequency of the test tone and plot the threshold when it becomes audible:
- Repeat for various frequencies of masking tones

Critical Bands

- Perceptually uniform measure of frequency, non-proportional to width of masking curve

  About 100 Hz for masking frequency < 500 Hz, grow larger and larger above 500 Hz.

- The width is called the size of the *critical band*

Barks

- Introduce new unit for frequency called a *bark* (after Barkhausen)

  1 Bark = width of one critical band

  For frequency < 500 Hz,

  For frequency > 500 Hz,

- Masking Thresholds on critical band scale:

Temporal masking

- If we hear a loud sound, then it stops, it takes a little while until we can hear a soft tone nearby

- Question: how to quantify?

- Experiment: Play 1 kHz *masking tone* at 60 dB, plus a *test tone* at 1.1 kHz at 40 dB. Test tone can't be heard (it's masked).

  Stop masking tone, then stop test tone after a short delay.

  Adjust delay time to the shortest time that test tone can be heard (e.g., 5 ms).

  Repeat with different level of the test tone and plot:

- Try other frequencies for test tone (masking tone duration constant). Total effect of masking

Summary

- If we have a loud tone at, say, 1 kHz, then nearby quieter tones are masked.
- Best compared on critical band scale - range of masking is about 1 critical band
- Two factors for masking - frequency masking and temporal masking
- Question: How to use this for compression?

MPEG Audio Compression

Some facts

- MPEG-1: 1.5 Mbits/sec for audio and video

  About 1.2 Mbits/sec for video, 0.3 Mbits/sec for audio

  (Uncompressed CD audio is 44,100 samples/sec * 16 bits/sample * 2 channels > 1.4 Mbits/sec)
- Compression factor ranging from 2.7 to 24.
- With Compression rate 6:1 (16 bits stereo sampled at 48 KHz is reduced to 256 kbits/sec) and optimal listening conditions, expert listeners could not distinguish between coded and original audio clips.
- MPEG audio supports sampling frequencies of 32, 44.1 and 48 KHz.
- Supports one or two audio channels in one of the four modes:

  1. Monophonic - single audio channel

  2. Dual-monophonic - two independent channels (similar to stereo)

  3. Stereo - for stereo channels that share bits, but not using joint-stereo coding

  4. Joint-stereo - takes advantage of the correlations between stereo channels

Steps in algorithm:

1. Use convolution filters to divide the audio signal (e.g., 48 kHz sound) into frequency subbands that approximate the 32 critical bands -> *sub-band filtering*.

2. Determine amount of masking for each band caused by nearby band using the results shown above (this is called the *psychoacoustic model*).

3. If the power in a band is below the masking threshold, don't encode it.

4. Otherwise, determine number of bits needed to represent the coefficient such that noise introduced by quantization is below the masking effect (Recall that 1 bit of quantization introduces about 6 dB of noise).

5. Format bitstream

Next Up Previous
Next: MPEG Layers **Up:** MPEG Audio Compression **Previous:** Steps in algorithm:

Example:

- After analysis, the first levels of 16 of the 32 bands are these:

Band	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Level (db)	0	8	12	10	6	2	10	60	35	20	15	2	3	5	3	1

- If the level of the 8th band is 60dB,

  it gives a masking of 12 dB in the 7th band, 15dB in the 9th.

  Level in 7th band is 10 dB (< 12 dB), so ignore it.

  Level in 9th band is 35 dB (> 15 dB), so send it.

  -> Can encode with up to 2 bits (= 12 dB) of quantization error.

MPEG Layers

- MPEG defines 3 layers for audio. Basic model is same, but codec complexity increases with each layer.

- Divides data into frames, each of them contains 384 samples, 12 samples from each of the 32 filtered subbands as shown below.

  Figure: Grouping of Sub-band Samples for Layer 1, 2, and 3

- Layer 1: DCT type filter with one frame and equal frequency spread per band. Psychoacoustic model only uses frequency masking.

- Layer 2: Use three frames in filter (before, current, next, a total of 1152 samples). This models a little bit of the temporal masking.

- Layer 3: Better critical band filter is used (non-equal frequencies), psychoacoustic model includes temporal masking effects, takes into account stereo redundancy, and uses Huffman coder.

Effectiveness of MPEG audio

Layer	Target	Ratio	Quality @	Quality @	Theoretical
	bitrate		64 kbits	128 kbits	Min. Delay
Layer 1	192 kbit	4:1	---	---	19 ms
Layer 2	128 kbit	6:1	2.1 to 2.6	4+	35 ms
Layer 3	64 kbit	12:1	3.6 to 3.8	4+	59 ms

- 5 = perfect, 4 = just noticeable, 3 = slightly annoying, 2 = annoying, 1 = very annoying
- Real delay is about 3 times theoretical delay

Streaming Audio (and video)

Popular new delivery medium for the Web and other Multimedia networks

Examples of streamed audio (and video) (and video)

- Real Audio
- Shockwave
- .wav files (not video obviously)

Here is an example of a real audio file recorded and produced for the Web by myself

- The file was originally recorded at CD Quality (44 Khz, 16-bit Stereo) and is nearly minutes in length.
- The original uncompressed file is about 80 Mb.
- The compressed file (at 33.3) is only 1.7 Mb in total and is still of very good quality.
- The file is downloaded to browser and not steamed above. Whilst real audio players and encoders are freely available (see {\em http://www.realaudio.com/}). Real audio servers {\bf cost money}.

  For further real audio and other .wav/.aiff fragments of my music go to Dave's Music Pages

  - Buffered Data:
    - Trick get data to destination before it's needed
    - Temporarily store in memory (Buffer)
    - Server keeps feeding the buffer
    - Client Application reads buffer
  - Needs Reliable Connection, moderately fast too.
  - Specialised client, Steaming Audio Protocol (PNM for real audio).

Further Exploration

- MPEG Audio Page
- MP3 Beginner's Guide

A good article on this subject is:

- ``A Tutorial on MPEG/Audio Compression", Davis Pan, ***IEEE Multimedia***, pp. 60-74, 1995.

See Video MPEG Further Reading Resources above also

Multimedia Integration, Interaction and Interchange

Dave Marshall
10/4/2001

Integrating Multimedia

So far we have been primarily concerned with each media type or format individually. We have noted that certain media (individually) are based on spatial and/or temporal representations, other may be static.

Once we start to integrate media spacial and temporal implications become even more critical. For example static text may need to index or label a portion of video at a given instant or segment of time and there the integration becomes temporal and spatial if the label is placed at a given location (or locations moving over time).

Clearly, it is important to know the tolerance and limits for each medium as integration will require knowledge of these for synchronisation and indeed create further limits (***e.g.*** bandwidth of two media types increase, if audio is encoded at a 48Khz sampling rate and it needs to accompany video being streamed out at 60 frames per second then inter-stream synchronisation is not necessarily straightforward.

It is common (obvious) that media types are bundled together for ease of delivery, storage ***etc.***. Therefore, it is not suprising that formats have been developed to support, store and deliver media in an integrated form.

The need for interchange between different multimedia applications probably running on different paltforms have lead to the evolution of common interchage file formats. Many of these formats build on underlying individual media formats (MPEG, JPEG ***etc.***) however further relationships are necessary when the media is truly integrated to become ***multimedia***. Spatial, temporal structural and procedural constraints will exist between the media. This especially true now that interaction is a common feature of multimedia.

Dave Marshall
10/4/2001

Interactive Multimedia

Modern multimedia presentation and applications are becoming increasingly interactive.

Simple interactions that simply start movie clips, audio segments animations etc are very common. Recently complex interactions between media is available. Following hyperlinks is instinctively non-linear and the advent of digital TV has lead to the need of a wide choice and the need for interactivity.

Interatcivity now needs to be incorporated as part of the media representation/format. The MHEG format (see below) has been developed expressely for such purposes.

We have now briefly addressed the need for integrated and interactive media formats. One last topic before we discuss specific formats is the need for common interchange formats.

Dave Marshall
10/4/2001

Multimedia Interchange

A Variety of multimedia applications runing on different paltforms will need to communicate with each other particularly if they are running on a distributed network.

Until recently (and it may even still pose some problems) the lack of a common interchage file format was a serious impediment to development of a market of multimedia applications.

A common interchange format needs to be widely adopted (be supported by many applications) and be sufficiently experessive to represent a wide variety of media content. These may be conflicting requirements since only when a wide variety of media is suported will it be widely adopted. Propriety application on support a small variety of media they require and may not readily adapt to other formats. Fortunately some widely accept standard that support a wide variety of media (with open standards even) are now developed.

The need for interchange formats are significant in several applications:

- As a final storage model for the creation and editing of multimedia decuments.
- As a format for delivery of final form digital medai. ***E.g.*** Compact Discs to end-use palyers.
- As a format for real-time delivery over a ditributed network
- for interapplication exchange of data.

Quicktime

Dave Marshall
10/4/2001

Introduction

QuickTime is the most widely used cross-platform multimedia technology available today. QuickTime now has powerful streaming capabilities, so you can enjoy watching live events as they happen. QuickTime 4 is the latest version and it includes streaming capabilities as well as the tools needed to create, edit, and save QuickTime movies. These tools include the QuickTime Player, PictureViewer, and the QuickTime Plug-in. Quicktime 5 is now in Public Beta development and is scheduled for release early in 2001.

QuickTime developed out of a multimedia extension for Apple's Macintosh(proprietry) System 7 operating system. It is now an international standard for multimedia interchange and is avalailbe for many platforms and as Web browser plug ins.

The following main features are summarised below:

Versatile support for web-based media

- Access to ***live*** and stored ***streaming media*** content with the QuickTime Player
- High-Quality Low-Bandwidth delievery of multimedia
- Easy view of QuickTime movies (with enhanced control) in Web Browsers and applications.
- Multi platform support.
- Built in support for most popular Internet media formats (Over 35 formats).
- Easy import/export of movies in the QuickTime Player

Sophisticated playback capabilities

- Play back full-screen video
- Play slide shows and movies continuously
- Work with video, still-image, and sound files in all leading formats

Easy content authoring and editing

- Create new QuickTime streaming movies by copying and pasting content from any supported format
- Enhance movies and still pictures with filters for sharpening, color tinting, embossing, and more
- Save files in multiple formats, including the new DV format for high-quality

video
- Create slide shows from pictures
- Add sound to a slide show

QuickTime is an ***open standard*** -- it embraces other standards and incorporates them into its environment. It supports every major file format for pictures, including BMP, GIF, JPEG, PICT, and PNG. QuickTime also supports every important professional file format for video, including AVI, AVR, DV, M-JPEG, MPEG-1, and OpenDML. Key standards for web streaming, including HTTP, RTP, and RTSP as set forth by the Internet Engineering Task Force, are supported as well. QuickTime supports Timecode tracks, including the critical standard for video Timecode set forth by SMPTE. And for musicians, QuickTime supports MIDI standards such as the Roland Sound Canvas and the GS format extensions. QuickTime is not a proprietary environment. Not only can QuickTime movies be played back on both Windows- and Mac OS-based systems (including Windows 95, Windows 98, and Windows NT), it can also be used on web servers in UNIX, Windows, or Mac OS environments. QuickTime movies can also be played back in any standard web browser, including Microsoft Internet Explorer, Netscape Navigator, Netscape Communicator, and America Online. Unlike more limited or proprietary formats, QuickTime makes it easy to combine media types and authoring tools from multiple platforms. Content creators can work on the platform of their choice and then deliver the output to a wide range of playback devices and computer platforms. Robust multiplatform support dramatically reduces production time, because different creators can simultaneously work on the same content using different platforms. QuickTime 4 extends this capability to any RTP/RTSP standards-based server running a QuickTime Streaming Server.

Next Up Previous

Next: Quicktime Support of Media **Up:** Quicktime **Previous:** Quicktime

Dave Marshall
10/4/2001

Quicktime Support of Media Formats

QuickTime can work with more types of media than any other technology. Whether you¹re creating streaming video web sites, CD-ROMs, DVDs, or professional video, QuickTime gives you the best options for quality and bandwidth efficiency.

Video
: -- QuickTime supports AVI, AVR, DV, OpenDML, and other professional digital video formats. Although AVI and other files can contain only audio and video, QuickTime can enhance these files with text, additional music tracks, and any other supported media types. QuickTime 4 features video compressors and decompressors that can handle needs ranging from CD-ROMs and DVDs to dial-up Internet access. QuickTime can stream video over the Internet even with 28.8-Kbps modems. QuickTime 4 includes Sorenson Video 2, which delivers the best possible video quality while maintaining the smallest file size. Sorenson Video incorporates the latest compression techniques, motion compensation, and data rate control methods to provide un-matched video at virtually any speed. With QuickTime, Cinepak, IMA, Intel Indeo Video, and the industry-standard H.263 compressor, you can target almost any audience, giving you the highest level of cross-platform compatibility.

MPEG format
: -- The MPEG standard for Macintosh is used extensively for consumer products, combining high-quality audio with low data rates. An Apple extension for QuickTime 4 for Macintosh provides direct access to MPEG-1 audio and video, including the popular MPEG-1, Layer 3 (MP3). You can also play back MP3 files with QuickTime 4 for Windows. The QuickTime file format for MPEG-4 has been chosen as an ISO standard, so QuickTime files you create today will be ready for MPEG-4 tomorrow.

Speech and music
: -- QuickTime brings high-quality digital audio to all types of applications, whether the audio is used by itself or with other media types. Support for Qualcomm¹s PureVoice technology gives media authors access to some of the highest-quality voice compression avail-able, in a format so compact that it can be enjoyed over a 14.4-Kbps modem. Since PureVoice is optimized for speech, it offers a much more effective solution than multipurpose audio formats. QuickTime 4 also provides amazing audio fidelity for music at greatly reduced bit rates through support of the QDesign Music 2 compressor. This compressor represents a breakthrough in audio encoding and decoding technology by providing unprecedented fidelity at as little as 1 percent of the original file size. Such efficiency is vitally important for the bandwidth-limited situations

frequently encountered on the Internet. For example, one minute of CD-quality audio can be reduced from an 11-megabyte file to a 150-kilobyte file that delivers full-bandwidth, 16-bit, 44.1-kHz stereo in real time over a 28.8-Kbps connection.
MIDI and music. MIDI music is an integral part of the QuickTime architecture. It can be used alone or with video, animation, still images, or other visual elements. In addition to playing music through internal or external speakers, QuickTime can route musical information to ex-ternal MIDI devices, effects processors, and drum machines. QuickTime 4 delivers CD-quality, low-bandwidth music by supporting over 200 instruments with the Roland Sound Canvas sound set. It also supports GS format extensions, which allow additional expressions for General or standard MIDI sequences. On Windows systems, QuickTime supports the use of the MIDI Mapper for use with external MIDI hardware.
Still images and pictures. The extensive collection of still-image importers in QuickTime 4 allows media authors to leverage photography and illustrations created in a wide range of formats. QuickTime 4 now has the ability to open FlashPix images with PictureViewer and export to PNG, TIFF, TARGA, and MacPaint images, while supporting 16-bit-per-channel files. It also supports multiple images in TIFF, FlashPix, and Adobe Photoshop formats. To enable you to work easily within workgroups, QuickTime 4 supports BMP, GIF, JPEG, PICT, PNG, and SGI formats.
Text. QuickTime supports searchable text tracks, with a Find command in the QuickTime Player or your custom application to allow content creators and users to quickly find a particular scene by searching through its script or annotation text. A single QuickTime movie can have multiple text tracks, simplifying the creation of multilingual movies. Text annotation can be used with any QuickTime-enabled media types: AVI for video files, and WAV, AIFF, or MPEG-1 for audio files. It can also be deployed as an HREF track, allowing you to embed URLs in your movies.
Animations and sprites. Macromedia Flash animation can now be played with any QuickTime application, including the QuickTime Plug-in. For superior animation with compact files, QuickTime 4 integrates a curve-based vector animation compressor and flexible sprite capabilities. For some types of animations, these vector-based tools can produce a dramatically smaller file than traditional compressors and prerendered video tracks can produce. QuickTime 4 also supports alpha channel compositing and special effects up to 16 bits per pixel.

Next Up Previous
Next: QuickTime Concepts **Up:** Quicktime **Previous:** Introduction
Dave Marshall
10/4/2001

QuickTime Concepts

To following concepts QuickTime are used by Quicktime:

Movies and Media Data Structures

-- A traditional movie, whether stored on film, laser disk, or tape, is a continuous stream of data. A QuickTime movie can be similarly constructed, but it need not be: a QuickTime movie can consist of data in sequences from different forms, such as analog video and CD-ROM. The movie is not the medium; it is the organizing principle.

A QuickTime movie may contain several tracks. Each track refers to a media that contains references to the movie data, which may be stored as images or sound on hard disks, floppy disks, compact discs, or other devices. The data references constitute the track's media. Each track has a single media data structure.

Components

--

QuickTime provides components so that every application doesn't need to know about all possible types of audio, visual, and storage devices. A component is a code resource that is registered by the Component Manager. The component's code can be available as a systemwide resource or in a resource that is local to a particular application. Each QuickTime component supports a defined set of features and presents a specified functional interface to its client applications. Applications are thereby isolated from the details of implementing and managing a given technology. For example, you could create a component that supports a certain data encryption algorithm. Applications could then use your algorithm by connecting to your component through the Component Manager, rather than by implementing the algorithm over again.

Image Compression

--

Image data requires a large amount of storage space. Storing a single 640-by-480 pixel image in 32-bit color can require as much as 1.2 MB. Similarly, sequences of images, like those that might be contained in a QuickTime movie, demand substantially more storage than single images. This is true even for sequences that consist of fairly small images, because the movie consists of a large number of those images. Consequently, minimizing the storage requirements for image data is an important consideration for any application that works with images or sequences of images.

The Image Compression Manager provides your application with an interface for compressing and decompressing images and sequences of images that is independent of devices and algorithms.

Time

--

Image compression is difficult but worthwhile-images, not to mention long sequences of images, take a lot of memory. Time management in QuickTime is equally essential. You must understand time management to understand the QuickTime functions and data structures.

Seemingly simple issues prove interesting-for example, determining the proper length (duration) of a movie. For many movies, the proper duration is the time required to play them in "real" time-that is, a rate in which human actions appear natural, and objects fall to earth accelerating at 32 feet per second per second. But what is the length of a movie that shows spreadsheet data charted over time, or a map of the earth that recapitulates continental drift? Add to this the differing clock speeds of different platforms, and the need to decompress in real time, and time proves, as ever, complex.

To manage these situations, QuickTime defines time coordinate systems, which anchor movies and their media data structures to a common temporal reality, the second. A time coordinate system contains a time scale that provides the translation between real time and the time in a movie. Time scales are marked in time units. The number of units that pass per second quantifies the scale-that is, a time scale of 26 means that 26 units pass per second and each time unit is 1/26 of a second. A time coordinate system also contains a duration, which is the length of a movie or a media in the number of time units it contains. Particular points in a movie can be identified by a time value, the number of time units elapsed to that point.

Each media has its own time coordinate system, which starts at time 0. The Movie Toolbox maps each type of media data from the movie's time coordinate system to the media's time coordinate system.

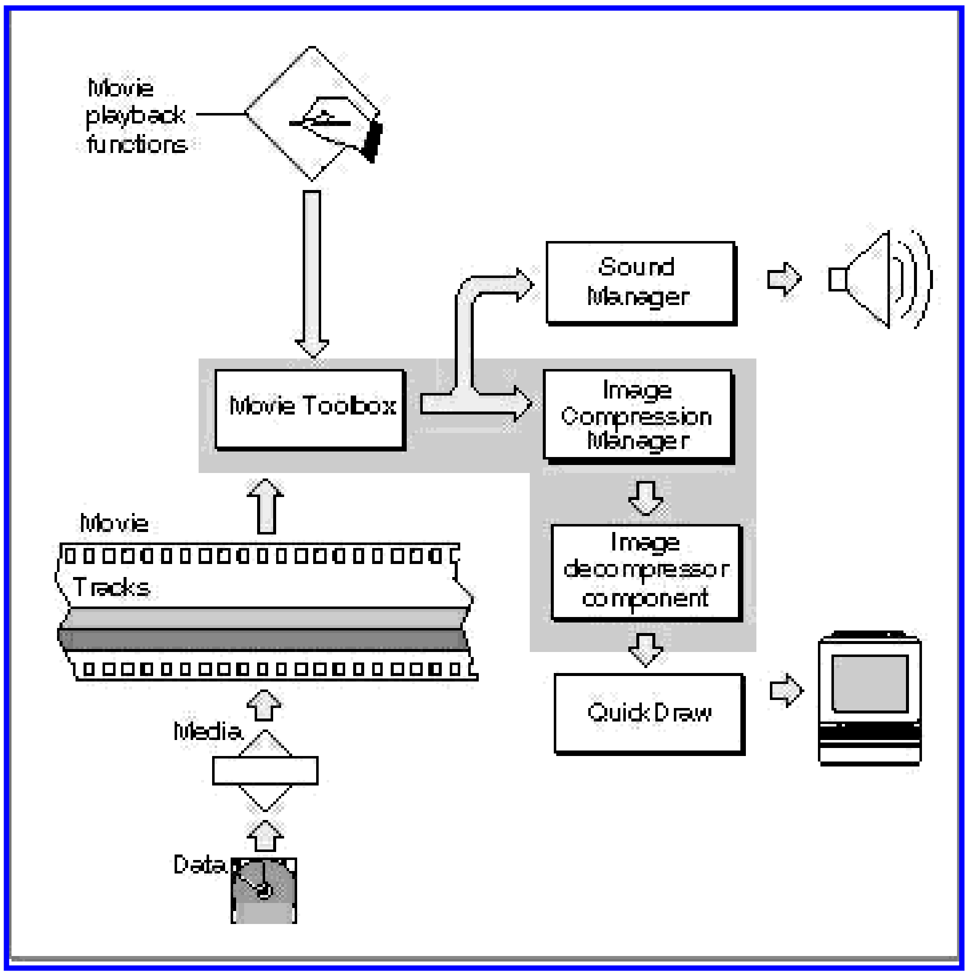
Next Up Previous

Next: The QuickTime Architecture **Up:** Quicktime **Previous:** Quicktime Support of Media

Dave Marshall
10/4/2001

The QuickTime Architecture

QuickTime comprises two managers: the Movie Toolbox and the Image Compression Manager. QuickTime also relies on the Component Manager, as well as a set of predefined components. Figure 1-1 shows the relationships of these managers and an application that is playing a movie.



Quicktime Architecture

The Movie Toolbox
-- Your application gains access to the capabilities of QuickTime by calling functions in the Movie Toolbox. The Movie Toolbox allows you to store, retrieve, and manipulate time-based data that is stored in QuickTime movies. A single movie may contain several types of data. For example, a movie that contains video information might include both video data and the sound data that accompanies the video.

The Movie Toolbox also provides functions for editing movies. For example, there are editing functions for shortening a movie by removing portions of the video and sound tracks, and there are functions for extending it with the addition of new data from other QuickTime movies.

The Movie Toolbox is described in the chapter "Movie Toolbox" later in this book. That chapter includes code samples that show how to play movies.

The Image Compression Manager

--

The Image Compression Manager comprises a set of functions that compress and decompress images or sequences of graphic images.

The Image Compression Manager provides a device-independent and driver-independent means of compressing and decompressing images and sequences of images. It also contains a simple interface for implementing software and hardware image-compression algorithms. It provides system integration functions for storing compressed images as part of PICT files, and it offers the ability to automatically decompress compressed PICT files on any QuickTime-capable Macintosh computer.

In most cases, applications use the Image Compression Manager indirectly, by calling Movie Toolbox functions or by displaying a compressed picture. However, if your application compresses images or makes movies with compressed images, you will call Image Compression Manager functions.

The Image Compression Manager is described in the chapter "Image Compression Manager" later in this book. This chapter also includes code samples that show how to compress images or make movies with compressed images.

The Component Manager

--

Applications gain access to components by calling the Component Manager. The Component Manager allows you to define and register types of components and communicate with components using a standard interface. A component is a code resource that is registered by the Component Manager. The component's code can be stored in a systemwide resource or in a resource that is local to a particular application.

Once an application has connected to a component, it calls that component directly. If you create your own component class, you define the function-level interface for the component type that you have defined, and all components of that type must support the interface and adhere to those definitions. In this manner, an application can freely choose among components of a given type with absolute confidence that each will work.

Next Up Previous

Next: QuickTime Components **Up:** Quicktime **Previous:** QuickTime Concepts

Dave Marshall
10/4/2001

QuickTime Components

QuickTime includes several components that are provided by Apple. These components provide essential services to your application and to the managers that make up the QuickTime architecture. The following Apple-defined components are among those used by QuickTime:

- movie controller components, which allow applications to play movies using a standard user interface standard image compression dialog components, which allow the user to specify the parameters for a compression operation by supplying a dialog box or a similar mechanism
- image compressor components, which compress and decompress image data sequence grabber components, which allow applications to preview and record video and sound data as QuickTime movies video digitizer components, which allow applications to control video digitization by an external device
- media data-exchange components, which allow applications to move various types of data in and out of a QuickTime movie derived media handler components, which allow QuickTime to support new types of data in QuickTime movies
- clock components, which provide timing services defined for QuickTime applications preview components, which are used by the Movie Toolbox's standard file preview functions to display and create visual previews for files sequence grabber components, which allow applications to obtain digitized data from sources that are external to a Macintosh computer
- sequence grabber channel components, which manipulate captured data for a sequence grabber component
- sequence grabber panel components, which allow sequence grabber components to obtain configuration information from the user for a particular sequence grabber channel component

We will study programming aspects of the above with Java in the next Chapter.

Dave Marshall
10/4/2001

Quicktime File Format

The Quicktime Movie File Format is a published (***http://developer.apple.com/techpubs/quicktime/qtdevdocs/PDF/QTFileFormat.pdf***) file format for storing multimedia content for Quicktime presentation. Several players are available on many platforms.

The Quicktime file format uses a track model for organising the temporally data of a movie. A movie can contain one or more tracks -- a track is a time-ordered sequence of a media type. The media are addressed using an ***edit list*** which is a list of end-points of digital media clips or segments.

See handouts, Ch 14 in ***Multimedia Systems*** by J. Buford (Addison Wesley) and the online Apple documentation for further details.

Dave Marshall
10/4/2001

Further Information

Quicktime Software, information, news etc. may be obtained from Apple's Quicktime Web site

Specific documents of interest to this course are:

- Quicktime documentation online
- Quicktime Information
- Quicktime Pro Features
- Quicktime file format

Dave Marshall
10/4/2001

Open Media Framework Interchange (OMFI) Format

The OMFI is a common interchange framework developed in repsonse to an industry led standardisation effort (inlcuding Avid -- a major digital video hardware/applications vendor)

Like Quicktime the primary concern of the OMFI format is concerned with temporal representation of media (such as video and audio) and a track model is used.

The primary emphasis is video production and an number of additional features reflect this:

- Source (analogue) material object represent videotape and film so that the origin of the data is readily identified. Final footage may resort to this original form so as to ensure highest possible quality.
- Special track types store (SMPTE) time codes for segements of data.
- Transitions and effects for overlapping and sequences of segments are predefined.
- ***Motion Control*** -- the ability to play one track at a speed which is a ratio of the speed of another track is supported.

The OMFI file format incorporates:

- a header -- include indicies for objects contained in file
- Object dictionary -- to enhance the OMFI class hierarchy in an application
- Object data
- Track data

Multimedia and Hypermedia Information Encoding Expert Group (MHEG)

The development of MHEG arose directly out of the increasing convergence of broadcast and interactive technologies. It specifies an encoding format for multimedia applications independently of service paradigms and network protocols. Like Quicktime and OMFI it is concerned with time-based media objects, whose encodings are determined by other standards.However, the scope of MHEG is larger in that it directly supports interactive media and real-time delivery over networks.

There have been a progressions of MHEG standards (much like MPEG) The current widespread standard is MHEG-5 but drafts standards exist up to MHEG-7.

Every design generally represents a compromise between conflicting goals. MHEG's design is no exception, especially if you consider that MHEG-5 (and later) targets a continuously growing and fiercely competitive market where broadcast and interactive technologies converge almost daily.

Converging technologies have often stimulated adopting standard solutions. Multimedia applications standards provide more than just the obvious objectives of portability and interoperability. A good multimedia standard can become a reference solution for system developers and application programmers. It also promotes the use of modular architectures that rely on common components that accomplish a specific functionality, such as interpreting and presenting MHEG applications to users. This task is performed by a compliant runtime engine (RTE), a resident software component that sched-ules delivery of an application to the user. It's aimed at a wide installation base within complete solutions, like a Video on Demand or an Interactive TV system. RTEs help improve a product's return on investment, abate a product's per unit costs, and provide high quality, robust products due to extensive product testing.

Next Up Previous

Next: The family of MHEG **Up:** Multimedia Integration, Interaction and **Previous:** Open Media Framework Interchange

Dave Marshall
10/4/2001

The family of MHEG standards

MHEG encompasses the family of standards issued by the ISO/IEC JTC1 joint technical committee's working group WG12‹information technology subcommitte SC29, coding of audio, picture multimedia, and hypermedia information. See Table 8.1 for the complete list of MHEG standards.

Table 8.1:MHEG Standards

Version	Complete Name	Status
MHEG-1	MHEG object representation-base	International standard
	notation (ASN.1)	
MHEG-2	MHEG object representation-alternate	Withdrawn
	notation (SGML)	
MHEG-3	MHEG script interchange representation	International standard
MHEG-4	MHEG registration procedure	International standard
MHEG-5	Support for base-level interactive applications	International standard
MHEG-6	Support for enhanced interactive applications	International standard
		(April 1998)
MHEG-7	Interoperability and conformance testing	Draft international standard
	for ISO/IEC 13522-5	(Jan 1999)

Since it was introduced first, MHEG-1 received the most attention. It's the generic standard for encoding multimedia objects without specific assumptions on the application area or on the target platform used for delivering and rendering these objects to the user. MHEG-3 provides a script extension to MHEG-1. MHEG-4 specifies a registration procedure for identifiers used by the objects to identify, for example, a specific format for content data. MHEG-5 can conceptually be considered a simplifying profile of MHEG-1. It addresses terminals with limited resources, like the set-top unit. Actually, an MHEG-1 decoder can't decode MHEG-5 applications due to some slightly different provisions added to optimize performance in VoD/ITV environments. MHEG-6 extends the ***declarative*** MHEG-5 approach with procedural code capabilities typical of a scripting language. It defines the interface (MHEG-5 API) and a script engine's runtime environment on top of an MHEG-5 engine using the Java virtual machine to provide a complete solution for application representation. MHEG-7, a new standard, addresses the conformance and interoperability of MHEG-5 engines and applications.

Next Up Previous

Next: MHEG-5 overview **Up:** Multimedia and Hypermedia Information **Previous:** Multimedia and Hypermedia Information

Dave Marshall
10/4/2001

MHEG-5 overview

SInce MHEG-5 is the widest MHEG standard in operation and is widely known we will highlight this MHEG standard.

A multimedia application can be conceived as a set of self-contained objects based on synchronization and spatial-temporal relationships of multiple media formats, structural composition, event-action associations, navigation, and user interaction capabilities. Controlling the playback of time-dependent contents, like streams of multiplexed audiovisual data requires specific support. These streams demand VCR control functions (play, pause, fast forward, and so on), as well as the capability to manage events generated during their presentation. For example, rendering text subtitles can be synchronized with timecode events generated during the playback of a stream. MHEG-5 represents an application, as a set of scenes, which contain objects common to all scenes. A scene supports the spatially and temporally coordinated presentation of audiovisual content consisting of graphics, bitmaps, text, and streams (based on the multiplex of audio and video components). Interaction can be performed via graphic elements like buttons, sliders, text entry boxes, and hypertext selections. Every scene, as well as an entire application, is a self-contained entity that can represent its localized behavior by links that are event-action associations. Events can be generated by users, expiration of timers, playback of streams, and other conditions within the RTE.

The global scope of MHEG-5 is to define the syntax and semantics of a set of object classes that can be used for interoperability of multimedia applications across minimal-resources platforms. The developed applications will reside on a server, and as portions of the application are needed, they will be downloaded to the client. In a broadcast environment, this download mechanism could rely, for instance, on cyclic rebroadcasting of all portions of the application. It is the responsibility of the client to have a runtime that interprets the application parts, presents the application to the user, and handles the local interaction with the user.

The major goals of MHEG-5 are:

- To provide a good standard framework for the development of client/server multimedia applications intended to run on a memory-constrained Client.
- To define a final-form coded representation for interchange of applications across platforms of different versions and brands.
- To provide the basis for concrete conformance levelling, guaranteeing that a conformant application will run on all conformant terminals.
- To allow the runtime engine on the Client to be ***small*** and ***easy*** to implement.
- To be free of strong constraints on the architecture of the Client.
- To allow the building of a wide range of applications. This means also providing access to external libraries. An application using external libraries will only be partly portable.
- To allow for application code that is guaranteed to be "safe" in the sense that it cannot harm other code in the Client, nor put the Client in an abnormal state.
- To allow automatic static analysis of (final-form) application code in order to help insure bug-free applications and minimize the debugging investment needed to get a robust application. Note that this analysis should be possible to implement independently of the authoring environment.
- To promote rapid application development by providing high-level primitives and provide a declarative paradigm for the application development.

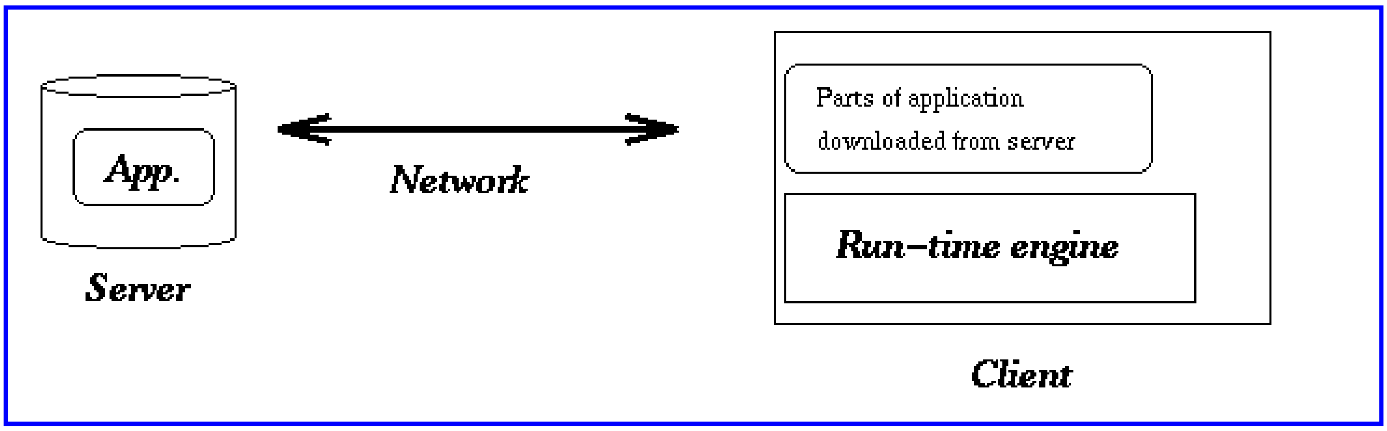
The MHEG-5 model is object-oriented. The actions are methods targeted to objects from different classes to perform a specific behavior and include:

- preparation,
- activation,

- controlling the presentation,
- user interaction,
- getting the value of attributes,
- and so on.

To allow interoperability across heterogeneous systems, MHEG-5 specifies the precise encoding syntax. Two notations are possible: the ASN.1 notation, which MHEG-1 also adopts, and a textual notation as illustrated below:.

In a client-server architecture (Fig 8.2), MHEG-5 applications are stored on the server and downloaded to the terminal for the RTE to interpret. This model is not limited to storage and retrieval services. In the broadcast environment, for example, the set of channels transmitted on a broadcast network can be considered a virtual server, where the download mechanism relies on cyclic rebroadcast of all portions of an application.



MHEG Client-Server Interaction

Next Up Previous
Next: MHEG Programming Principles **Up:** Multimedia and Hypermedia Information **Previous:** The family of MHEG
Dave Marshall
10/4/2001

MHEG Programming Principles

MHEG-5 provides suitable abstractions for managing active, autonomous, and reusable entities (since it adopts an object-oriented approach).

A class is specified by three kinds of properties:

- attributes that make up an object's structure,
- events that originate from an object, and
- actions that target an object to accomplish a specific behavior or to set or get an attribute's value.

The most significant classes of MHEG-5 are now breifly described:

Root
: -- A common Root superclass provides a uniform object identification mechanism and specifies the general semantics for preparation/destruction and activation/deactivation of objects, including notification of changes of an object's availability and running status. These general provisions are further specialized moving downwards through the inheritance tree, which first branches into the Group and Ingredient classes.

Group
: -- This abstract class handles the grouping of objects in the Ingredient class as a unique entity of interchange. In fact, Group objects can be addressed and independently downloaded from the server. A Group can be specialized into Application and Scene classes.

Application
: -- An MHEG-5 application is structurally organized into one Application and one or more Scene objects. The Application object represents the entry point that performs a transition to the presentation's first Scene. Generally, this transition occurs at startup (see below code examples) because a presentation can't happen without a Scene running.

  The Launch action activates an Application after quitting the active Application. The Quit action ends the active Application, which also terminates the active Scene's presentation. The Ingredients of an Application are available to the different Scenes that become active, thereby allowing an uninterrupted presentation of contents (for example, a bitmap can serve as the common background for all Scenes in an Application).

Scene
: -- This class allows spatially and temporally coordinated presentations of Ingredients. At most, one Scene can be active at one time. Navigating within an Application is performed via the `TransitionToaction` that closes the current Scene, including its Ingredients, and activates the new one. The `SceneCoordinateSystem` attribute specifies the

presentation space's 2D size for the Scene. If a user interaction occurs in this space, a UserInput event is generated. A Scene also supports timers. A Timer event is generated when a timer expires.

Ingredient

-- This abstract class provides the common behavior for all objects that can be included in an Application or a Scene.

The `OriginalContent` attribute maps object and content data. It contains either included content or a reference to an external data source (such as a URL or a DSMCC file name). The `ContentHook` attribute specifies the encoding format for the content. However, MHEG-5 does not list the supported encoding formats. See coding examples below for the use of content references and hooks.

The action `Preload` gives hints to the RTE for making the content available for presentation. Especially for streams, this action does not completely download the content, it just sets up the proper network connection to the site where the content is stored. The action `Unload` frees allocated resources for the content.

The Presentable, Stream, and Link classes are subclasses of the Ingredient class.

Presentable

-- This abstract class specifies the common aspects for information that can be seen or heard by the user. The `Run` and `Stop` actions activate and terminate the presentation, while generating the `IsRunning` and `IsStopped` events.

Visible

-- The Visible abstract class specializes the `Presentable` class with provisions for displaying objects in the active Scene's presentation space.

The `OriginalBoxSize` and `OriginalPosition` attributes respectively specify the size and position of the object's bounding box relative to the Scene's presentation space. The actions `SetSize` and `SetPosition` change the current values of these attributes.

The specialized objects in the Visible class include:

- ***Bitmap*** **--** This object displays a 2D array of pixels. The Tiling attribute specifies whether the content will be replicated throughout the `BoxSize` area. The action `ScaleBitmap` scales the content to a new size.

  Example, to create a simple bitmap object:

  ```
  (bitmap: BgndInfo
      content-hook: #bitmapHook
      content-data: referenced-content: "Info.bitmap"
      box-size: ( 320 240 )
      original-position: ( 0 0 )
  )
  ```

- ***LineArt, DynamicLineArt* --** A LineArt is a vectorial representation of graphical entities, like polylines and ellipses. `DynamicLineArt` draws lines and curves on the fly in the `BoxSize` area.
- ***Text* --** This object represents a text string with a set of rendition attributes. Essentially, these attributes specify fonts and formatting information like justification and wrapping.

Stream

-- This class (a subclass of Ingredient) controls the synchronized presentation of multiplexed audio-visual data (such as an MPEG-2 file). A Stream object consists of a list of components from the `Video`, `Audio`, and `RTGraphics` (animated graphics) classes. The `OriginalContent` attribute of the Stream object refers to the whole multiplex of data streams.

When a Stream object is running, its streams can be switched on and off independently. This lets users switch between different audio trails (different languages) or choose which video stream(s) to present among a range of available ones. For example, the Turin code example below contains an MPEG-1 Stream composed of one audio and one video component. These components automatically activate when a run action targets the whole Stream because their `InitiallyActive` attribute is set to ***true***.

Specific events are associated with playback: `StreamPlaying/StreamStopped` notifies the actual initiation/termination and `CounterTrigger` notifies the system when a previously booked time-code event occurs. The Turin code example below shows how the `CounterTrigger` event can be used to synchronize text subtitling and also illustrates the `SetCounterPosition` and `SetCounterEndPosition` actions to specify a temporal segment for presentation.

Link

-- The Link class implements event-action behavior by a condition and an effect. The `LinkCondition` contains an `EventSource` -‹ a reference to the object on which the event occurs -‹ an `EventType` that specifies the kind of event and a possible `EventData` that is a data value associated with the event.

MHEG-5 Action objects consist of a sequence of elementary actions. Elementary actions are comparable to methods in an object-oriented paradigm. The execution of an Action object means that each of its elementary actions are invoked sequentially.

As an example, consider the following Link, which transitions to another Scene when the character `A` is entered in the EntryField `EF1`.

Example, to create a simple link:

```
(link: Link1
    event-source: EF1
    event-type: #NewChar
```

```
        event-data: 'A'
        link-effect:
            (action: transition-to: Scene2)
    )
```

In the Turin code example below, `Link 49` triggers only if a `CounterTrigger` event on the video clip occurs with `EventData = 3`.

Specifically, this lets you associate a different effect with every booked value of the `CounterPosition`. The `LinkEffect` comprises a set of actions that are executed in sequence when an event that matches with the `LinkCondition` occurs. Every action specifies the target object and, possibly, other parameters depending on the type of action. MHEG-5 specifies more than 100 kinds of actions.

Interactible

-- This abstract class provides a way for users to interact with objects within the following sub-classes:

Hotspot, PushButton, and SwitchButton

-- These subclasses implement button selection capability and generate the `IsSelected` event. Example, to create a simple `SwitchButton`:

```
(switchbutton: Switch1
    style: #radiobutton
    position: ( 50 70 )
    label: "On"
)
```

Hypertext

-- This class extends the Text class with ***anchors***. When selected, these anchors link text content to associated information.

Slider and EntryField

-- Respectively, these objects let users adjust a numeric value (such as the volume of an audio stream) and edit text.

Example, to create a simple slider:

```
(slider: Slider1
    box-size: ( 40 5 )
    original-position: ( 100  100 )
    max-value: 20
    orientation: #right
)
```

Next Up Previous

Next: Interaction within a Scene **Up:** Multimedia and Hypermedia Information **Previous:**

MHEG-5 overview

Dave Marshall
10/4/2001

Interaction within a Scene

The MHEG application is event-driven, in the sense that all actions are called as the result of an event firing a link. Events can be divided into two main groups: synchronous events and asynchronous events. Asynchronous events are events that occur asynchronously to the processing of Links in the MHEG engine. These include timer events and user input events. An application area of MHEG-5 (such as DAVIC) must specify the permissible UserInput events within that area. Synchronous events are events that can only occur as the result of an MHEG-5 action being targeted to some objects. A typical example of a synchronous event is `IsSelected`, which can only occur as the result of the MHEG-5 action Select being invoked. Synchronous events are always dealt with immediately; asynchronous events are queued.

The mechanism at the heart of the MHEG engine, therefore, is the following:

1. After a period of of idleness, an asynchronous event occurs. The event can be a user input event, a timer event, a stream event, or some other type of event.
2. Possibly, a link that reacts on the event is found. This link is then fired. If no such link is found, the process starts again at 1.
3. The result of a link being fired is the execution of an action object, which is a sequence of elementary actions. These can change the state of other objects, create or destroy other objects, or cause events to occur.
4. As a result of the actions being performed, synchronous events may occur. These are dealt with immediately, i.e., before processing any other asynchronous events queued.

When all events have been processed, the process starts again at 1.

Dave Marshall
10/4/2001

Availability; Running Status

Before doing anything to an object, the MHEG-5 engine must prepare it. Preparing an object typically entails retrieving it from the server, decoding the interchange format and creating the corresponding internal data structures, and making the object available for further processing. The preparation of an object is asynchronous; its completion is signalled by an IsAvailable event.

All objects that are part of an application or a scene have a RunningStatus, which is either true or false. Objects whose RunningStatus is true are said to be running, which means that they perform the behaviour they are programmed for. More concretely:

- only running Visibles are actually visible on the screen,
- only running Audio objects are played out through the loudspeaker,
- only running Links will execute the action part if the associated event occurs, etc.

Dave Marshall
10/4/2001

Interactibles

The MHEG-5 mix-in class Interactible groups some functionality associated with user interface-related objects (Slider, HyperText, EntryField, Buttons).

These objects can all be highlighted (by setting their HighlightStatus to True).

They also have the attribute InteractionStatus, which, when set to true, allows the object to interact directly with the user, thus bypassing the normal processing of UserInput events by the MHEG-5 engine.

Exactly how an Interactible reacts when its InteractionStatus is true is implementation-specific.

As an example, the way that a user enters characters in an EntryField can be implemented in different ways in different MHEG-5 engines.

At most one Interactible at a time can have its InteractionStatus set to True. Interactibles

Dave Marshall
10/4/2001

Visual Representation

For objects that are visible on the screen, the following rules apply :

- Objects are drawn downwards and to the right of their position on the screen. This point can be changed during the life cycle of an object, thus making it possible to move objects.
- Objects are drawn without scaling. Objects that do not fit within their bounding box are clipped.
- Objects are drawn with "natural" priority, ***i.e.***, on top of already existing objects. However, it is possible to move objects to the top or the bottom of the screen, as well as putting them before or after another object.
- The screen can be frozen, allowing the application to perform many (possibly slow) changes and not update the screen until it's unfrozen.

Dave Marshall
10/4/2001

Object Sharing Between Scenes

It is possible within MHEG-5 to share objects between some or all scenes of an application. As an example, this can be used to have variables retain their value over scene changes, or to have an audio stream play on across a scene change. Shared objects are alway contained in an Application object. Since there is always exactly one Application object running whenever a scene is running, the objects contained in an application object are visible to each of its scenes.

Dave Marshall
10/4/2001

Object Encoding

The MHEG-5 specification does not prescribe any specific formats for the encoding of content. For example, it is conceivable that a Video object is encoded as MPEG or as motion-JPEG. This means that the group using MHEG-5 must define which content encoding schemes to apply for the different objects in order to achieve interoperability.

However, MHEG-5 does specify a final-form encoding of the MHEG-5 objects themselves. This encoding is an instance of ASN.1, using the Basic Encoding Rules (BER).

Dave Marshall
10/4/2001

Conformance

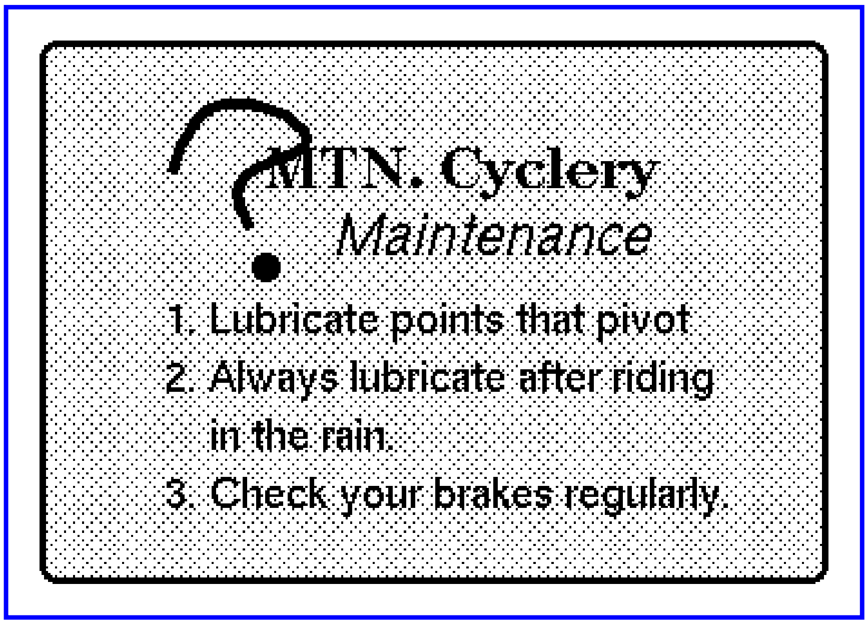
The issue of conformance, though of crucial importance, has not yet been extensively addressed by the MHEG committee. It is expected that a conformance definition for the standard will have to be drafted, probably lagging the standard itself by some period of time.

Dave Marshall
10/4/2001

MHEG Coding Examples

A Simple MHEG Example

Below (Fig 8.3) is a very simple scene that displays a bitmap and text. The user can press the 'Left' button on the input device and a transition is made from the current scene, `InfoScene1`, to a new scene, `InfoScene2`.


MTN. Cyclery
Maintenance
1. Lubricate points that pivot
2. Always lubricate after riding in the rain.
3. Check your brakes regularly.

Simle MHEG Example

The pseudo-code from the above scene may look like the following:

```
(scene:InfoScene1
    <other scene attributes here>
    group-items:
      (bitmap: BgndInfo
          content-hook: #bitmapHook
          original-box-size: (320 240)
          original-position: (0  0)
          content-data: referenced-content: "InfoBngd"
       )
       (text:
           content-hook: #textHook
           original-box-size: (280 20)
           original-position: (40 50)
           content-data: included-content: "1. Lubricate..."
       )
       links:
           (link: Link1
               event-source: InfoScene1
               event-type: #UserInput
               event-data: #Left
```

```
                    link-effect: action: transition-to: InfoScene2
                )
)
```

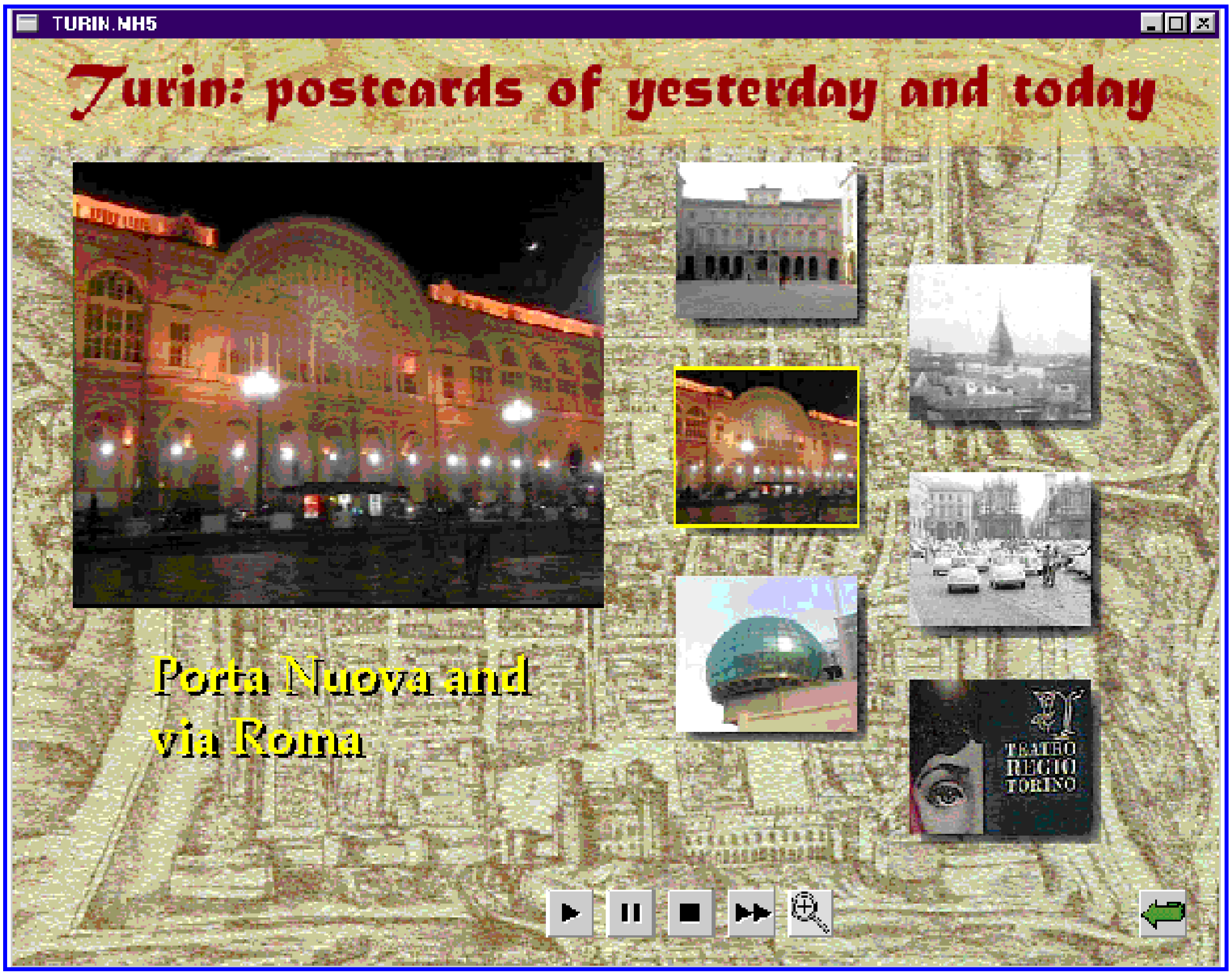
A More Complex Example (Turin)

Another example application lets users retrieve tourist information about the city of Turin, Italy. URL: http://drogo.cselt.stet.it/ufv/mediatouch/mh5webapp/to_audio_e_0.mh5

Figure 8.4 shows a screen shot of the ***main_scene*** object. This scene consists of

- a text title,
- a bitmap background,
- a video clip, which is playing a segment of Porta Nuova, the main railway station, enriched with a text subtitle object (left side),
- some interactive thumbnails (right side),
- a set of buttons that provide VCR controls,
- functions to switch between viewing the video in ***normal*** and ***zoom*** modes, and item a ***return to previous screen*** capability.

The playback of the video clip synchronizes with the subtitle's content. Selecting an interactive thumbnail constrains the playback to the associated temporal segment.



MHEG Example Presentation (Turin Guide)

The code listing below is an excerpt from the Turin application's MHEG textual notation:

```
{:Application ("turin.mh5" 0)
:OnStartUp ( // sequence of initialization
actions
:TransitionTo (("main_scene.mh5" 0)) //
activation of the first scene
)
}
{:Scene ("main_scene.mh5" 0)
:OnStartUp ( // sequence of initialization
actions
preload (2) // the connection to the
source of the video clip is set up
...
setCounterTrigger (2 3 190000) // book a
time code event at 190000 msec
...
)
:Items ( // both presentable ingredients and
links
{:Bitmap 1 // background bitmap
:InitiallyActive true
:CHook 3 // JPEG
:OrigContent
:ContentRef ("background.jpg")
:OrigBoxSize 800 600
:OrigPosition 0 0
}
{:Stream 2 // video clip
:InitiallyActive false
:CHook 101 // MPEG-1
:OrigContent
:ContentRef ("turin.mpg")
:Multiplex (
{:Audio 3 // audio component of the
video clip
:ComponentTag 1 // refers to audio
elementary stream
:InitiallyActive true
}
{:Video 4 // video component of the
video clip
:ComponentTag 2 // refers to
video elementary stream
:InitiallyActive true
:OrigBoxSize 352 288
:OrigPosition 40 80
}
)
}
{:HotSpot 20 // "Porta Nuova" hotspot
:InitiallyActive true
:OrigBoxSize 120 100
:OrigPosition 440 214
}
... // 25 more presentable ingredients
{:Link 30 // selecting an "interactive"
thumbnail
:EventSource (20) // "Porta Nuova"
hotspot
:EventType IsSelected
:LinkEffect (
:SetSpeed (2 1 1) // video clip:
speed is set to 1/1 (normal)
:SetCounterPosition (2 190000) //
initial point of the segment
:SetCounterEndPosition (2 246500) //
end point of the segment
:Run (2) // activate playback with
the above settings
)
}
{:Link 49 // the video clip crosses a pre
defined time code position
:EventSource (2) // video clip
:EventType CounterTrigger
:EventData 3 // booked at startup by
setCounterTrigger (2 3 190000)
:LinkEffect (
:SetData (5 // text subtitle is set
to a new string, that is
:NewRefContent ("st9.txt")) //
"Porta Nuova and via Roma"
:SetHighlightStatus (20 true) //
hotspot 20 is highlighted
)
}
... // 58 more links
)
:SceneCS 800 600 // size of the scene's
presentation space
}
```

Dave Marshall
10/4/2001

An MHEG Player Java Applet -- Futher MHEG Examples

The Technical University of Berlin have produced a MHEG Java Engine (***http://enterprise.prz.tu-berlin.de/imw/">http://enterprise.prz.tu-berlin.de/imw/***

Java Class libraries (with JavaDoc documentation) and details on installation/compilation ***etc*** are also available. Several examples of MHEG coding, including an MHEG Introduction written in MHEG.

Dave Marshall
10/4/2001

Running the MHEG Engine

The MHEG engine exists as a Java applet (although you can of course is the class librarie in your own java code (applications and applets)).

The MHEG engine is available in the ***MHEG:MHEG Java Applet*** folder on the Macintosh ***Applications HD*** in the Multimedia Lab.

You can run the applet through any Java enabled browser or appletviewer.

Here as an example of how to run the main applet provided for the ***demo*** MHEG example:

```
<applet name="MHEG 5 Engine"
         code="mheg5/POM/Mheg5Applet.class"
         codebase="applications/"
         archive="mhegwww.zip"
         width="510"
         height="346"
         align="center"
         <param name="objectBasePath" value="file:.">
         <param  name="groupIdentifier" value="demo/startup">
         <param name="mon" value="false">
</applet>
```

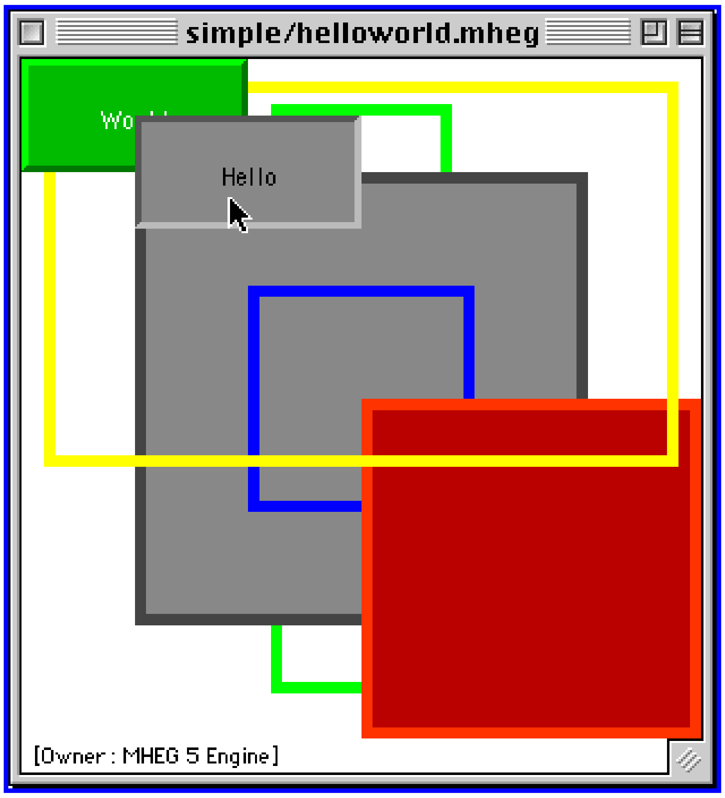
If you use the applet yourself you may need to change:

- the `code` and `codebase` paths -- these specufy where the applications and applet clasess reside.
- the `groupIdentifier` ***value*** -- for most of the applicatioin demos a `startup` MHEG file is reference first in a folder for each application. (see other examples below.

Dave Marshall
10/4/2001

MHEG Example -- The Simple MHEG Presentation

The Simple example produces the following output shown in Fig 8.5



MHEG Simple application Example

The presentation create two buttons, labelled ``Hello'' and ``World'' respectively, and some recangle graphics.

When pressed the button is brought to the foreground of the display.

The applet is called via:

```
<html>
  <head>
    <title>MHEG-5 Engine written in Java</title>
  </head>
  <body bgcolor="#ffffff">
    <center>
    <h1>MHEG-5 Engine</h1>
      <applet
         name="MHEG 5 Engine"
         code="mheg5/POM/Mheg5Applet.class"
         codebase="applications/"
         archive="mhegwww.zip"
         width="510"
         height="346"
         align="center"
         alt="If you had a java-enabled browser, you would see an applet
here.">
         <hr>If your browser recognized the applet tag,
           you would see an applet here.<hr>
           <param name="objectBasePath" value="file:.">
           <param  name="groupIdentifier" value="simple/startup">
           <param name="mon" value="false">
      </applet>
    </center>
  </body>
</html>
```

The MHEG modules for this presentation are:

startup
  -- calls `helloworld.mheg`
helloworld.mheg
  -- sets up main presentation
scene1.mheg
  -- called in `helloworld.mheg`

The MHEG code for each module is a follows:

startup:

```
{:Application  ("simple/startup" 0)
  //:OnStartUp    (:TransitionTo(("simple/helloworld.mheg" 0)))
  :Items(
          {:Link 1
           :EventSource 1
           :EventType IsRunning
           :LinkEffect (:TransitionTo(("simple/helloworld.mheg" 0)))
          }
        )
}
\end{verbatim


{\bf helloworld.mheg}

\footnotesize
\begin{verbatim}
{:Scene ( "simple/helloworld.mheg" 0 )

 :Items
 (

   {:Rectangle 4000
    :OrigBoxSize 80 260 :OrigPosition 110 20
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Green :OrigRefFillColour transparent
   }
   {:Rectangle 4001
    :OrigBoxSize 200 200 :OrigPosition 50 50 :OrigLineWidth 5
```

```
:OrigLineStyle 1
    :OrigRefLineColour DarkGray :OrigRefFillColour Gray
   }
   {:Rectangle 4002
    :OrigBoxSize 100 100 :OrigPosition 100 100
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Blue :OrigRefFillColour transparent
   }
   {:Rectangle 4003
    :OrigBoxSize 150 150 :OrigPosition 150 150
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour "#FF2211" :OrigRefFillColour DarkRed
   }
   {:Rectangle 4004
    :OrigBoxSize 280 170 :OrigPosition 10 10
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Yellow :OrigRefFillColour transparent
   }

   {:PushButton 4005 :InitiallyActive true
    :OrigBoxSize 100 50
    :ButtonRefColour DarkGreen :OrigLabel "World"
   }
   {:PushButton 4006 :InitiallyActive true
    :OrigBoxSize 100 50 :OrigPosition 50 25
    :ButtonRefColour Gray :OrigLabel "Hello"
   }


   {:Link 1015
    :EventSource 4006 :EventType IsSelected
    :LinkEffect ( :BringToFront ( 4005 )
                  :SetHighlightStatus ( 4005 true ) )
   }
   {:Link 1016
    :EventSource 4005 :EventType IsSelected
    :LinkEffect ( :BringToFront ( 4006 ) )
   }
   {:Link 1017
    :EventSource 4005 :EventType CursorEnter
    :LinkEffect ( :BringToFront ( 4005 ) )
   }
   {:Link 1018
    :EventSource 4006 :EventType CursorEnter
    :LinkEffect ( :BringToFront ( 4006 ) )
   }

   {:ObjectRefVar 100 :OrigValue :ObjectRef 4001 }
   {:ObjectRefVar 101 :OrigValue :ObjectRef 4002 }
   {:Link 1000
    :EventSource 0 :EventType UserInput :EventData 1
    :LinkEffect ( :SetPosition( 4006 50 25 ) )
   }
   {:Link 1001
    :EventSource 0 :EventType UserInput :EventData 2
    :LinkEffect ( :SetPosition ( 4006 50 200 ) )
```

```
}
{:Link 1002
 :EventSource 0 :EventType UserInput :EventData 3
 :LinkEffect ( :Stop ( :IndirectRef 101 ) )
}
{:Link 1003
 :EventSource 0 :EventType UserInput :EventData 4
 :LinkEffect ( :Run ( :IndirectRef 101 ) )
}

{:Link 1004
 :EventSource 0 :EventType UserInput :EventData 15
 :LinkEffect ( :TransitionTo( ( "simple/scene1.mheg" 0) ) )
}

{:Link 1005
 :EventSource 0 :EventType UserInput :EventData 5
 :LinkEffect ( :BringToFront ( 4000 ) )
}
{:Link 1006
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect ( :BringToFront ( 4001 ) )
}
{:Link 1007
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect ( :BringToFront ( 4002 ) )
}
{:Link 1008
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect ( :BringToFront ( 4003 ) )
}
{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect ( :BringToFront ( 4004 ) )
}

{:Link 1010
 :EventSource 0 :EventType UserInput :EventData 10
 :LinkEffect ( :SendToBack ( 4000 ) )
}
{:Link 1011
 :EventSource 0 :EventType UserInput :EventData 11
 :LinkEffect ( :SendToBack ( 4001 ) )
}
{:Link 1012
 :EventSource 0 :EventType UserInput :EventData 12
 :LinkEffect ( :SendToBack ( 4002 ) )
}
{:Link 1013
 :EventSource 0 :EventType UserInput :EventData 13
 :LinkEffect ( :SendToBack ( 4003 ) )
}
{:Link 1014
 :EventSource 0 :EventType UserInput :EventData 14
 :LinkEffect ( :SendToBack ( 4004 ) )
}
```

```
 )

 :InputEventReg 1
 :SceneCS 300 300
 :MovingCursor true
}
```

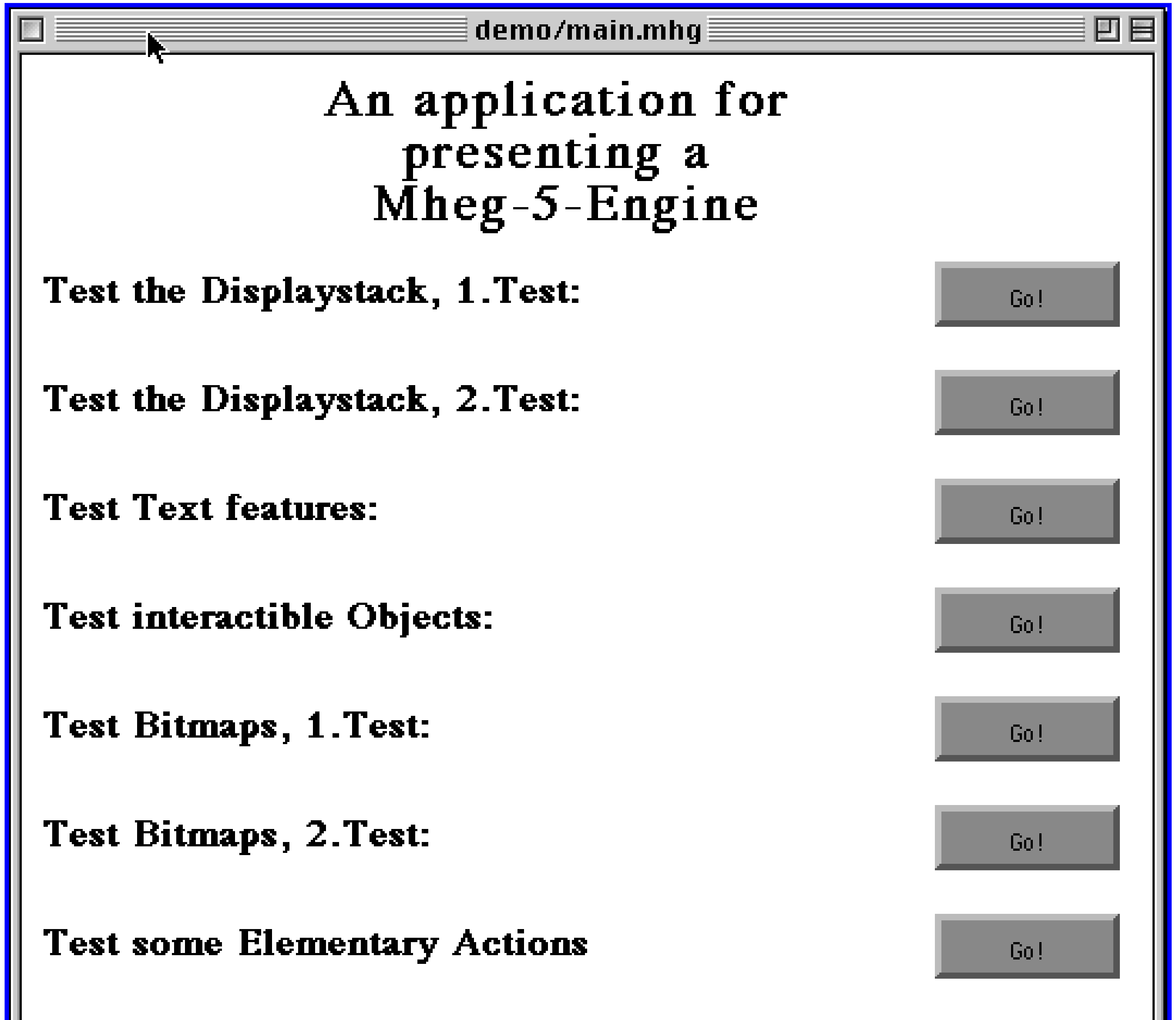
scene1.mheg

```
{:Scene ( "simple/scene1.mheg" 0 )

 :Items
 (
   {:Rectangle 4000
      :OrigBoxSize 300 300 //  200 200
    :OrigPosition 20 80
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour "Black" :OrigRefFillColour DarkRed
   }
   {:Rectangle 4001
    :OrigBoxSize 200 200 :OrigPosition 100 180
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Blue :OrigRefFillColour DarkBlue
   }

   {:Link 1000
    :EventSource 0 :EventType UserInput :EventData 6
    :LinkEffect ( :PutBehind ( 4003 4000 ) )
   }
   {:Link 1001
    :EventSource 0 :EventType UserInput :EventData 7
    :LinkEffect ( :PutBehind ( 4002 4000 ) )
   }
   {:Link 1002
    :EventSource 0 :EventType UserInput :EventData 8
    :LinkEffect ( :PutBehind ( 4001 4000 ) )
   }

   {:Link 1003
    :EventSource 0 :EventType UserInput :EventData 9
    :LinkEffect ( :PutBefore ( 4003 4000 ) )
   }
   {:Link 1004
    :EventSource 0 :EventType UserInput :EventData 10
    :LinkEffect ( :PutBefore ( 4002 4000 ) )
   }
   {:Link 1005
    :EventSource 0 :EventType UserInput :EventData 11
    :LinkEffect ( :PutBefore ( 4001 4000 ) )
   }

   {:Link 1006
    :EventSource 0 :EventType UserInput :EventData 1
    :LinkEffect ( :SetPosition ( 4000 20 80 ) )
   }
   {:Link 1007
    :EventSource 0 :EventType UserInput :EventData 2
    :LinkEffect ( :SetPosition ( 4000 20 120 ) )
   }

   {:Link 1008
    :EventSource 0 :EventType UserInput :EventData 3
      :LinkEffect ( :SetBoxSize ( 4000 200 300 ) )   // 200 200
   }
   {:Link 1009
    :EventSource 0 :EventType UserInput :EventData 4
    :LinkEffect ( :SetBoxSize ( 4000 300 300 ) )
   }

   {:Link 1100 :InitiallyActive true
    :EventSource 0 :EventType UserInput :EventData 16
    :LinkEffect ( :TransitionTo( ( "simple/helloworld.mheg" 0) ) )
   }
 )

 :InputEventReg 1
 :SceneCS 400 400
}
```

Dave Marshall
10/4/2001

MHEG Example -- The Demo MHEG Presentation

The Demo example produces the output shown in Fig 8.6.



An application for presenting a Mheg-5-Engine

Test the Displaystack, 1.Test:

Test the Displaystack, 2.Test:

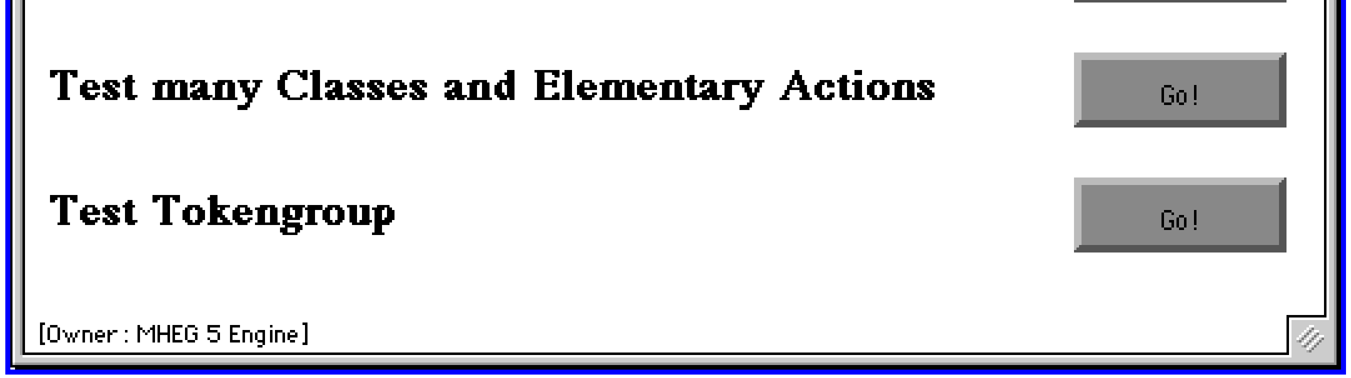
Test Text features:

Test interactible Objects:

Test Bitmaps, 1.Test:

Test Bitmaps, 2.Test:

Test some Elementary Actions

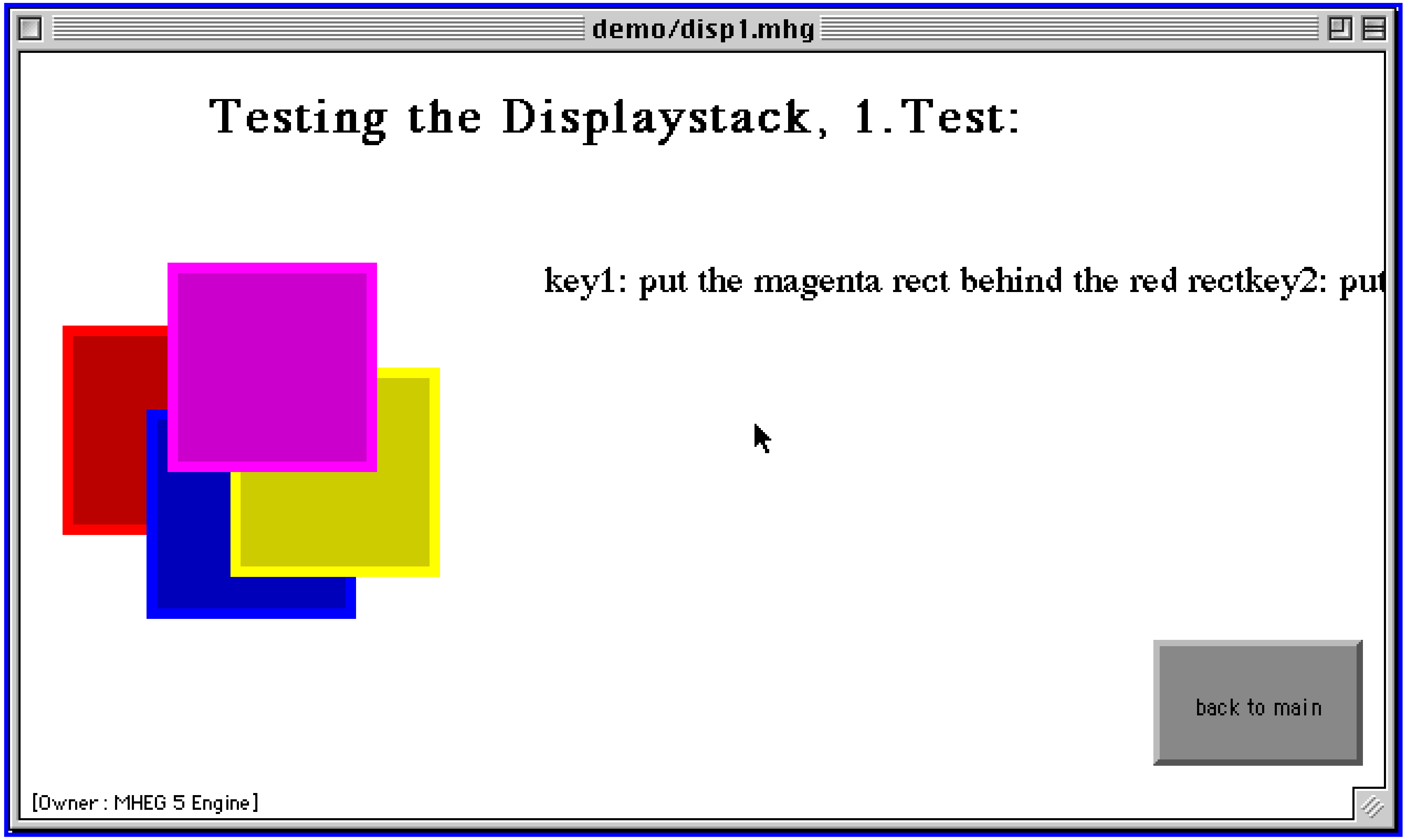


MHEG Demo application Example

As can be seen many of the key features of MHEG are illustrated in further sub-windows (click on button to move to respective window). Try these out for yourself.

The following MHEG modules are used:

startup
-- intial module
main.mhg
-- Called by startup
disp1.mhg
-- input from numeric keys 1 and 2 to tile rectangles (Fig 8.7)



Testing the Displaystack, 1.Test:

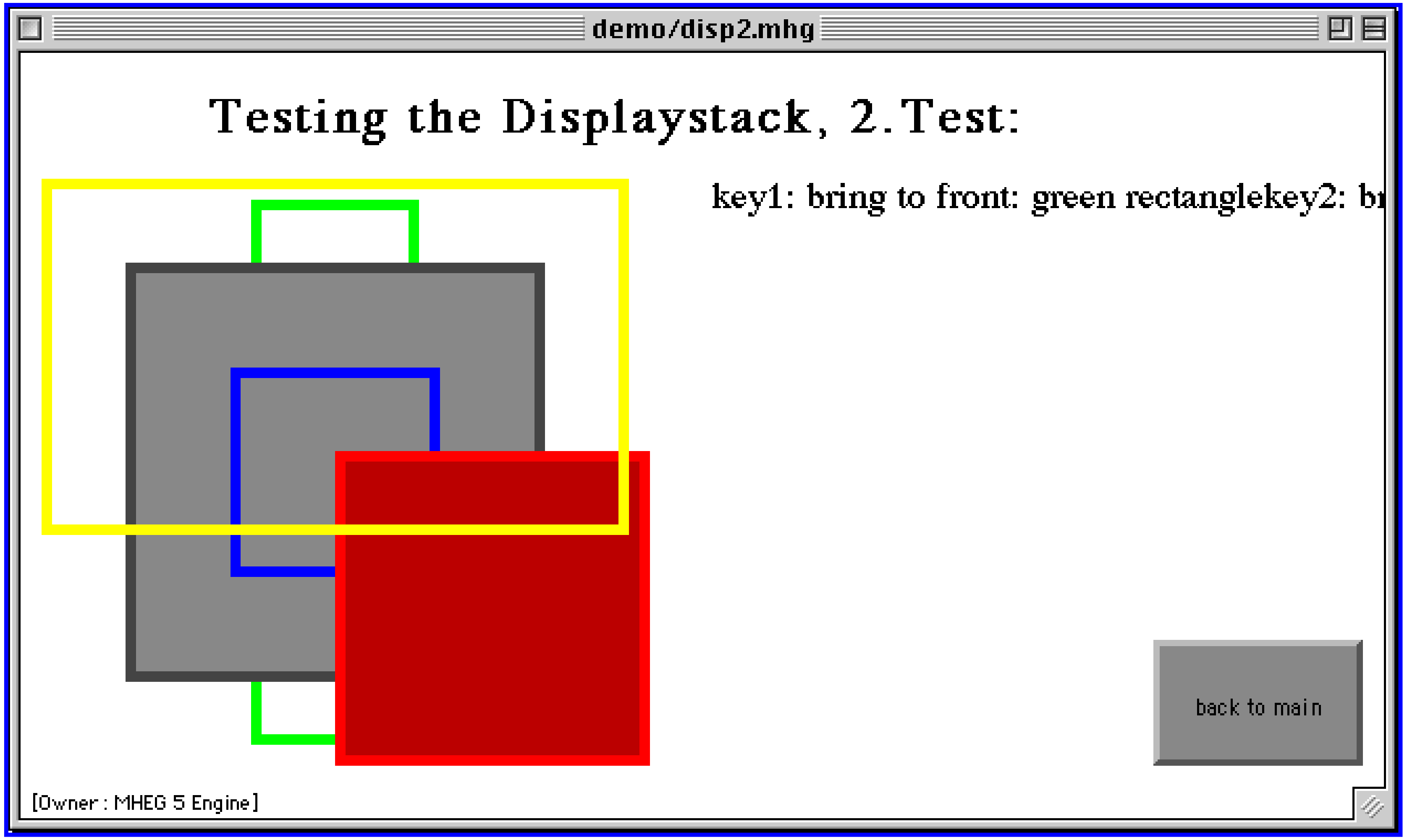
key1: put the magenta rect behind the red rectkey2: put

[Owner : MHEG 5 Engine]

MHEG Demo application Display1 Example

disp2.mhg

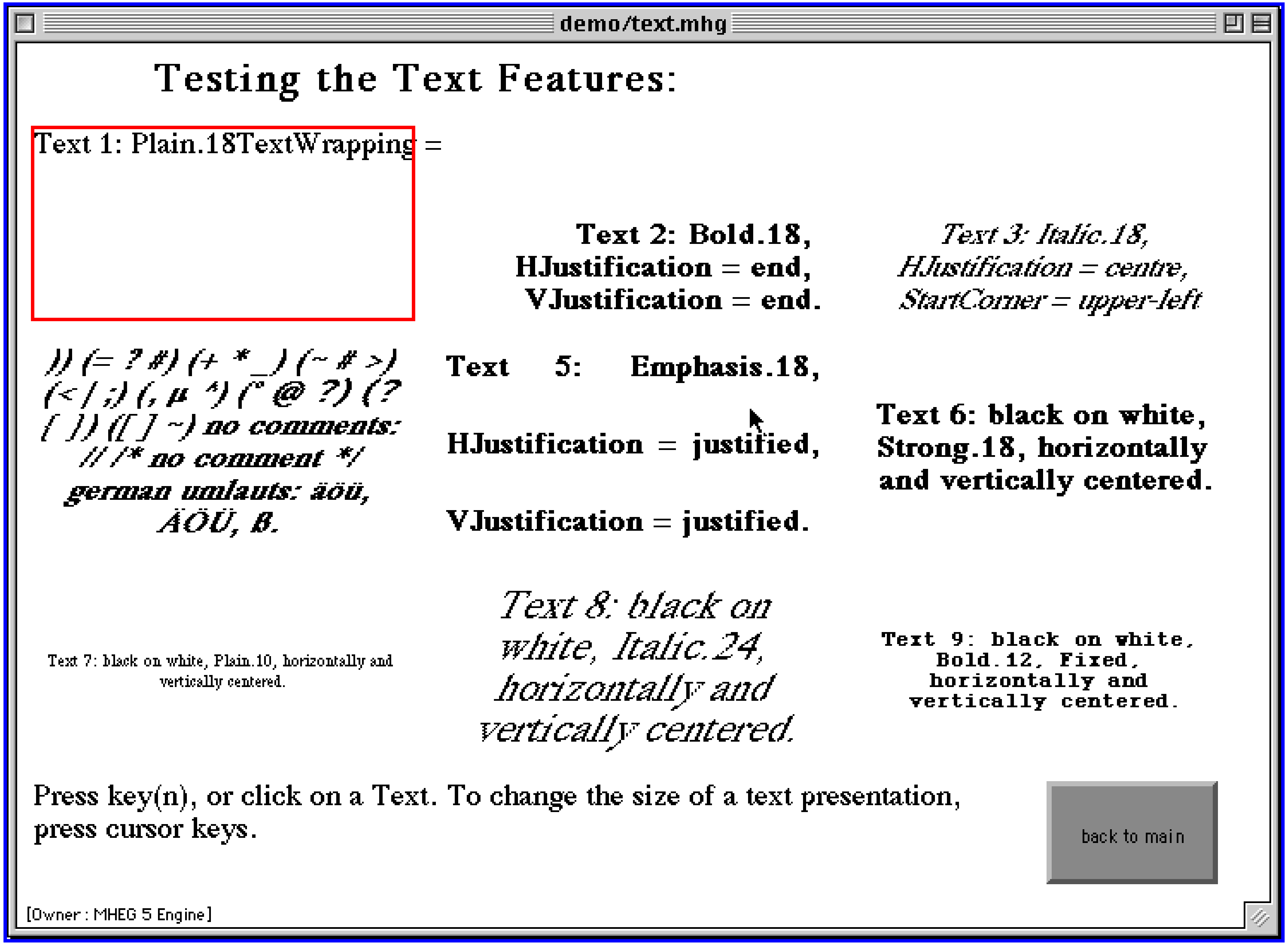
-- input from numeric keys 1 and 2 to tile rectangles (different display) (Fig 8.8)


demo/disp2.mhg
Testing the Displaystack, 2.Test:
key1: bring to front: green rectanglekey2: br
back to main
[Owner : MHEG 5 Engine]

MHEG Demo application Display2 Example

text.mhg

-- illustrates MHEG control of text display (Fig 8.9)



Testing the Text Features:

Text 1: Plain.18TextWrapping =

Text 2: Bold.18,
HJustification = end,
VJustification = end.

Text 3: Italic.18,
HJustification = centre,
StartCorner = upper-left

)) (= ? #) (+ *_) (~ # >)
(< / ;) (, µ ´) (˚ @ ?) (?
{ }) ([ ] ~) no comments:
// /* no comment */
german umlauts: äöü,
ÄÖÜ, ß.

Text 5: Emphasis.18,
HJustification = justified,
VJustification = justified.

Text 6: black on white,
Strong.18, horizontally
and vertically centered.

Text 7: black on white, Plain.10, horizontally and
vertically centered.

Text 8: black on
white, Italic.24,
horizontally and
vertically centered.

Text 9: black on white,
Bold.12, Fixed,
horizontally and
vertically centered.

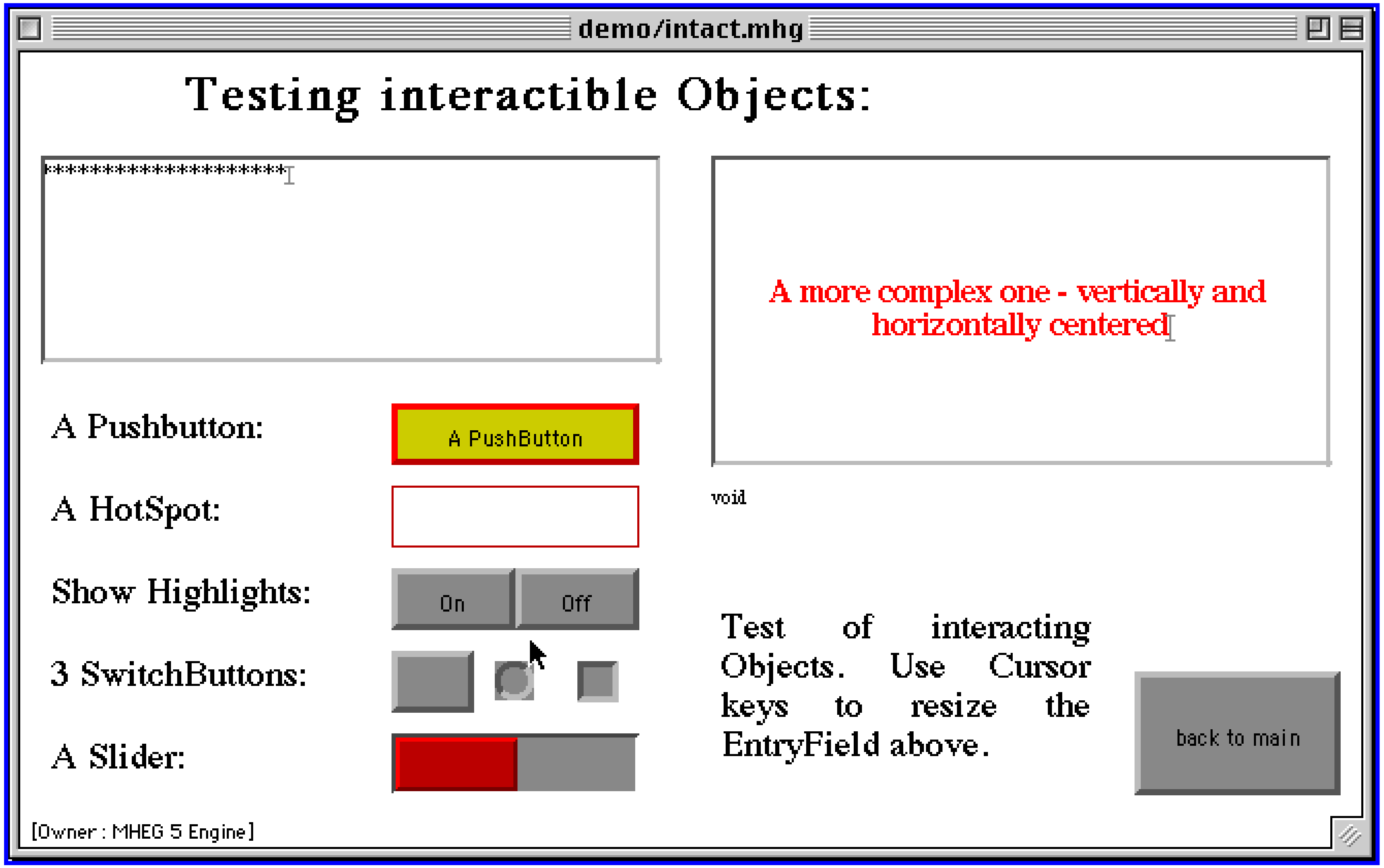
Press key(n), or click on a Text. To change the size of a text presentation,
press cursor keys.

[Owner : MHEG 5 Engine]

MHEG Demo application Text Example

intact.mhg

-- illustrates MHEG interactive objects (Fig 8.10)


demo/intact.mhg
Testing interactible Objects:
****
A more complex one - vertically and horizontally centered
A Pushbutton:
A PushButton
A HotSpot:
void
Show Highlights:
On
Off
3 SwitchButtons:
A Slider:
Test of interacting Objects. Use Cursor keys to resize the EntryField above.
back to main
[Owner : MHEG 5 Engine]

MHEG Demo application Interactive Objects Example

bitmap1.mhg

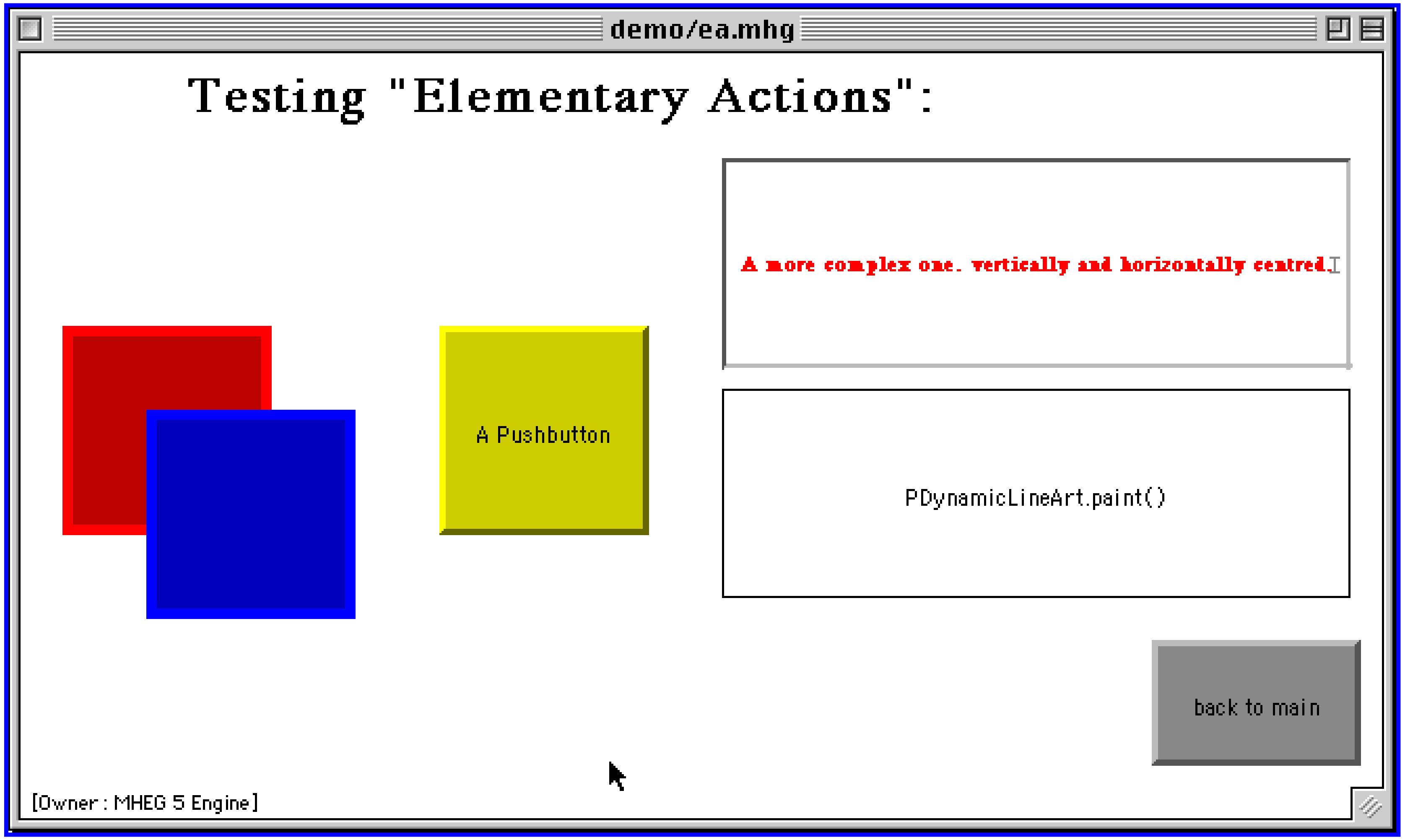
-- illustraes MHEG display of bitmaps

bitmap2.mhg

-- illustraes MHEG display of bitmaps

ea.mhg

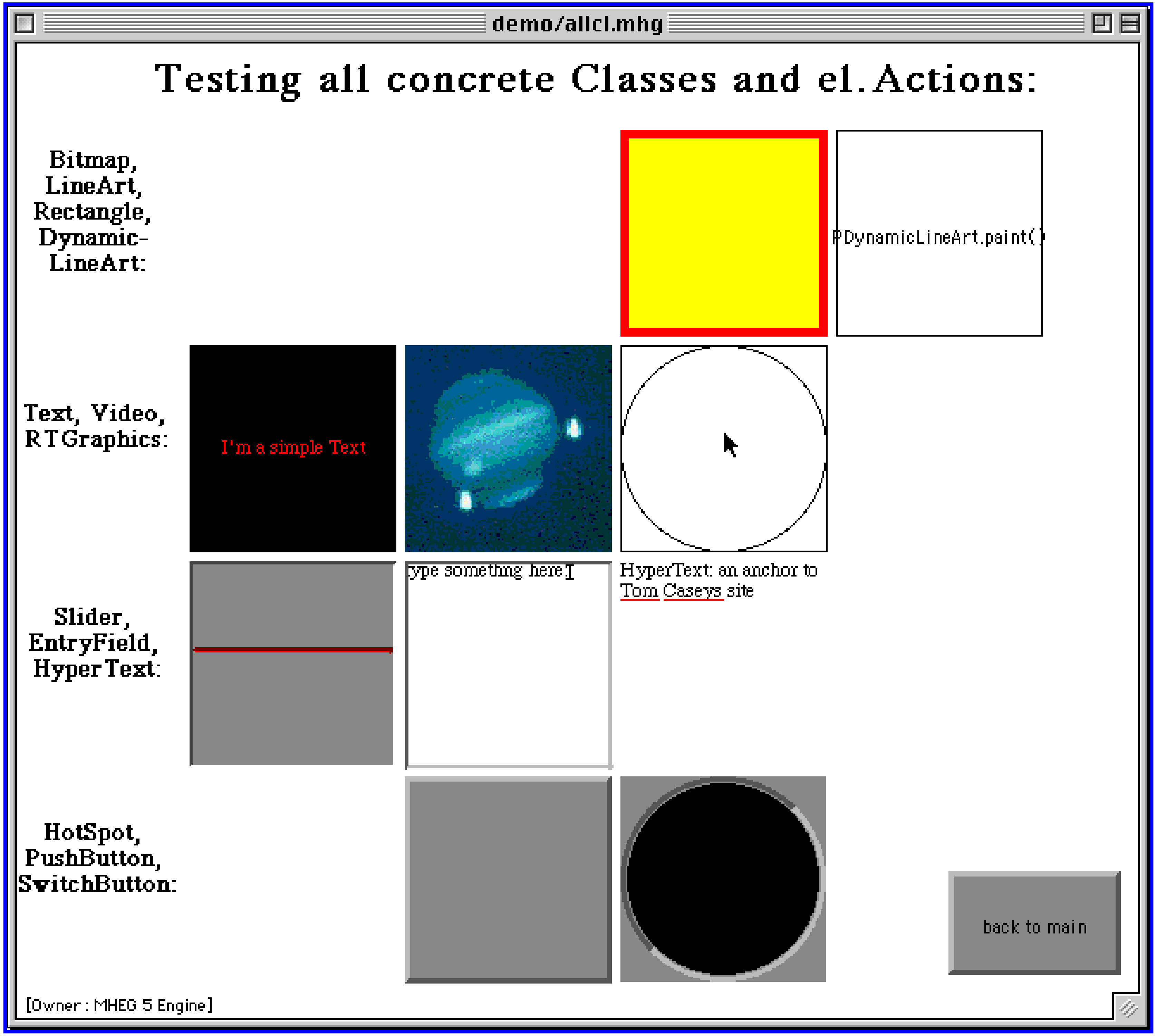
-- illustraes MHEG elementary actions (Fig 8.11)


demo/ea.mhg
Testing "Elementary Actions":
A more complex one. vertically and horizontally centred.
A Pushbutton
PDynamicLineArt.paint()
back to main
[Owner : MHEG 5 Engine]

MHEG Demo application Elementary Actions Example

allcl.mhg

-- MHEG conctrete classes and elementary actions (Fig 8.12)



demo/allcl.mhg

Testing all concrete Classes and el. Actions:

Bitmap, LineArt, Rectangle, Dynamic-LineArt:

PDynamicLineArt.paint()

Text, Video, RTGraphics:

I'm a simple Text

Slider, EntryField, HyperText:

ype something here

HyperText: an anchor to Tom Caseys site

HotSpot, PushButton, SwitchButton:

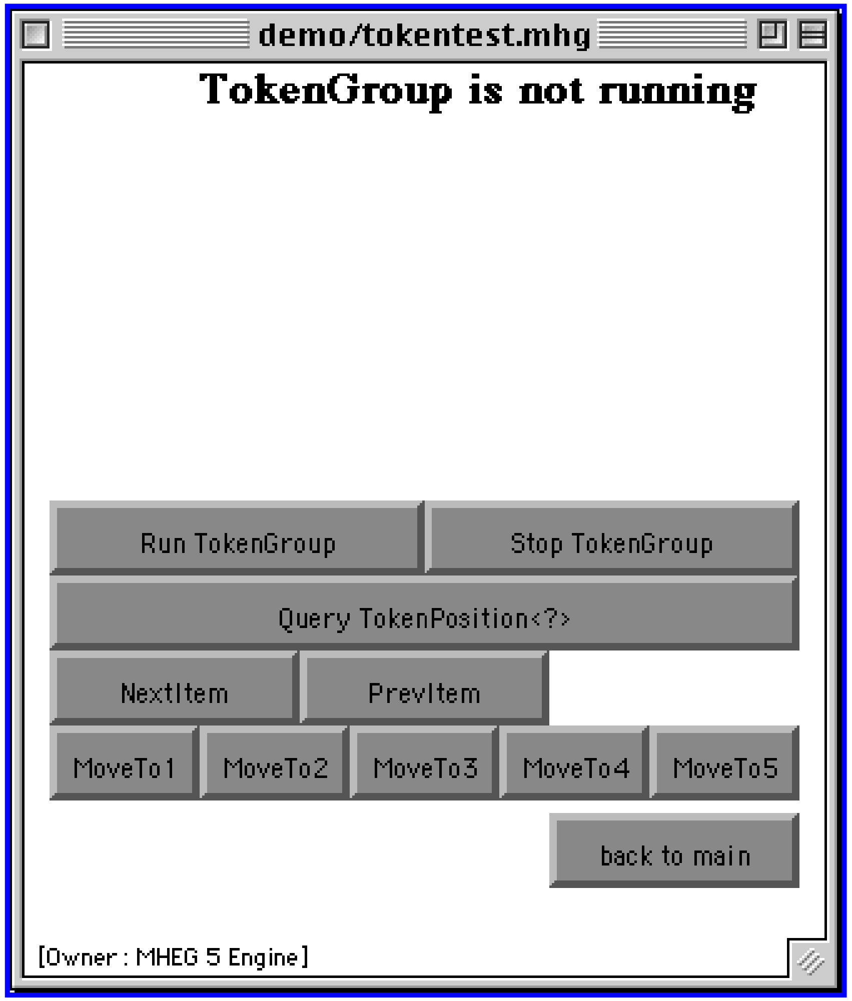
back to main

[Owner : MHEG 5 Engine]

MHEG Demo application Conctrete Classes Example

token.mhg

-- MHEG token groups example (Fig 8.13)



MHEG Demo application Token Groups Example

The MHEG code for the modules is as follows:

Startup:

```
{:Application  ("demo/startup" 0)
  // :OnStartUp    (:TransitionTo(("demo/main.mhg" 0)))
  :Items(
          {:Link 1
           :EventSource 1
           :EventType IsRunning
           :LinkEffect (:TransitionTo(("demo/main.mhg" 0)))
          }
        )
}
```

main.mhg:

```
{:Scene ( "demo/main.mhg" 0 )

 :Items
 (
   {:Text 100
    :OrigContent  'An application for presenting a Mheg-5-Engine'
    :OrigBoxSize 320 80
    :OrigPosition 90 10
    :FontAttributes Bold.24 :FontName Proportional
    :HJustification centre
    :TextWrapping true
   }

   {:Bitmap 101
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 15
             :Tiling false
             }

   {:Bitmap 102
    :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
             :OrigPosition 420 15
             :Tiling false
             }

   {:Bitmap 103
    :OrigContent :ContentRef ( "demo/prz.gif" )
```

```
          :OrigBoxSize 48 43
          :OrigPosition 470 15
          :Tiling false
          }

{:Link  110
 :EventSource 0
 :EventType UserInput
 :EventData 16
 :LinkEffect (
              :Quit (( "demo/startup" 0 ))
             )
}


// ----------------------------------------------------------------

{:Text 1000
 :OrigContent  'Test the Displaystack, 1.Test:'
 :OrigBoxSize 300 30
 :OrigPosition 10 100
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 1001
 :OrigBoxSize   85  30
 :OrigPosition 420 95
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 1002
 :EventSource 1001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/disp1.mhg" 0) )
             )
}

{:Link 1003
 :EventSource 1001 :EventType CursorEnter
 :LinkEffect ( :Activate(1005) )
}
{:Link 1004
 :EventSource 1001 :EventType CursorLeave
 :LinkEffect ( :DeActivate(1005) )
```

```
}
{:Link 1005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
   :LinkEffect ( :TransitionTo( ( "demo/disp1.mhg" 0) )
               )
}


// ----------------------------------------------------------------

{:Text 2000
 :OrigContent  'Test the Displaystack, 2.Test:'
 :OrigBoxSize 300 30
 :OrigPosition 10 150
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 2001
 :OrigBoxSize   85  30
 :OrigPosition 420 145
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 2002
 :EventSource 2001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/disp2.mhg" 0) )
             )
}


{:Link 2003
 :EventSource 2001 :EventType CursorEnter
 :LinkEffect ( :Activate( 2005 ) )
}
{:Link 2004
 :EventSource 2001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 2005 ) )
}
{:Link 2005
```

```
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
   :LinkEffect ( :TransitionTo( ( "demo/disp2.mhg" 0) )
               )
}


// ----------------------------------------------------------------

{:Text 3000
 :OrigContent  'Test Text features:'
 :OrigBoxSize 300 30
 :OrigPosition 10 200
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 3001
 :OrigBoxSize   85  30
 :OrigPosition 420 195
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 3002
 :EventSource 3001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/text.mhg" 0) )
             )
}
{:Link 3003
 :EventSource 3001 :EventType CursorEnter
 :LinkEffect ( :Activate( 3005 ) )
}
{:Link 3004
 :EventSource 3001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 3005 ) )
}
{:Link 3005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
```

```
    :LinkEffect ( :TransitionTo( ( "demo/text.mhg" 0) )
                )
 }

 // ----------------------------------------------------------------

 {:Text 4000
  :OrigContent  'Test interactible Objects:'
  :OrigBoxSize 300 30
  :OrigPosition 10 250
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 4001
  :OrigBoxSize   85  30
  :OrigPosition 420 245
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 4002
  :EventSource 4001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/intact.mhg" 0) )
              )
 }
 {:Link 4003
  :EventSource 4001 :EventType CursorEnter
  :LinkEffect ( :Activate( 4005 ) )
 }
 {:Link 4004
  :EventSource 4001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 4005 ) )
 }
 {:Link 4005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/intact.mhg" 0) )
                )
 }

// ----------------------------------------------------------------
```

```
{:Text 5000
 :OrigContent  'Test Bitmaps, 1.Test:'
 :OrigBoxSize 300 30
 :OrigPosition 10 300
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 5001
 :OrigBoxSize   85  30
 :OrigPosition 420 295
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 5002
 :EventSource 5001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/bitmap1.mhg" 0) )
             )
}

{:Link 5003
 :EventSource 5001 :EventType CursorEnter
 :LinkEffect ( :Activate( 5005 ) )
}
{:Link 5004
 :EventSource 5001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 5005 ) )
}
{:Link 5005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
   :LinkEffect ( :TransitionTo( ( "demo/bitmap1.mhg" 0) )
               )
}


// -----------------------------------------------------------------

{:Text 6000
 :OrigContent  'Test Bitmaps, 2.Test:'
```

```
   :OrigBoxSize 300 30
   :OrigPosition 10 350
   :FontAttributes Bold.18 :FontName Proportional
  }

  {:PushButton 6001
   :OrigBoxSize   85  30
   :OrigPosition 420 345
   :ButtonRefColour gray
   :OrigLabel "Go!"
  }

  {:Link 6002
   :EventSource 6001 :EventType IsSelected
   :LinkEffect (  :TransitionTo( ( "demo/bitmap2.mhg" 0) )
               )
  }

  {:Link 6003
   :EventSource 6001 :EventType CursorEnter
   :LinkEffect ( :Activate( 6005 ) )
  }
  {:Link 6004
   :EventSource 6001 :EventType CursorLeave
   :LinkEffect ( :DeActivate( 6005 ) )
  }
  {:Link 6005
     :InitiallyActive false
     :EventSource 0
     :EventType UserInput
     :EventData 15
     :LinkEffect ( :TransitionTo( ( "demo/bitmap2.mhg" 0) )
                 )
  }


 // ----------------------------------------------------------------

  {:Text 7000
   :OrigContent  'Test some Elementary Actions'
   :OrigBoxSize 300 30
   :OrigPosition 10 400
   :FontAttributes Bold.18 :FontName Proportional
```

```
}

{:PushButton 7001
 :OrigBoxSize   85  30
 :OrigPosition 420 395
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 7002
 :EventSource 7001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/ea.mhg" 0) )
             )
}
{:Link 7003
 :EventSource 7001 :EventType CursorEnter
 :LinkEffect ( :Activate( 7005 ) )
}
{:Link 7004
 :EventSource 7001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 7005 ) )
}
{:Link 7005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
   :LinkEffect ( :TransitionTo( ( "demo/ea.mhg" 0) )
               )
}

// -----------------------------------------------------------------
{:Text 8000
 :OrigContent  'Test many Classes and Elementary Actions'
 :OrigBoxSize 400 30
 :OrigPosition 10 450
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 8001
 :OrigBoxSize   85  30
 :OrigPosition 420 445
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 8002
 :EventSource 8001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/allcl.mhg" 0) )
             )
}

{:Link 8003
 :EventSource 8001 :EventType CursorEnter
 :LinkEffect ( :Activate( 8005 ) )
}
{:Link 8004
 :EventSource 8001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 8005 ) )
}
{:Link 8005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
   :LinkEffect ( :TransitionTo( ( "demo/allcl.mhg" 0) )
               )
}

// ----------------------------------------------------------------

{:Text 9000
 :OrigContent  'Test Tokengroup'
 :OrigBoxSize 400 30
 :OrigPosition 10 500
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 9001
 :OrigBoxSize   85  30
 :OrigPosition 420 495
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 9002
```

```
    :EventSource 9001 :EventType IsSelected
    :LinkEffect (  :TransitionTo( ( "demo/token.mhg" 0) )
                )
   }
   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/token.mhg" 0) )
                  )
   }


 )
 :InputEventReg 1
 :SceneCS 520 550
 :MovingCursor true
}
```

disp1.mhg:

```
{:Scene ( "demo/disp1.mhg" 0 )

 :Items
 (
  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }
   {:Text 1000
    :OrigContent  'Testing the Displaystack, 1.Test:'
    :OrigBoxSize 500 30
    :OrigPosition 90 20
    :FontAttributes Bold.24 :FontName Proportional
   }

   {:Text 1001
    :OrigContent  'key1: put the magenta rect behind the red rect
key2: put the yellow rect behind the red rect
key3: put the blue rect behind the red rect
key4: put the magenta rect before the red rect
key5: put the yellow rect before the red rect
key6: put the blue rect before the red rect.'
    :OrigBoxSize 400 200
    :OrigPosition 250 100
    :FontAttributes Plain.18 :FontName Proportional
    :TextWrapping false
   }

   {:Bitmap 1010
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 15
             :Tiling false
             }

   {:Bitmap 1011
    :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
             :OrigPosition 540 15
             :Tiling false
             }

   {:Bitmap 1012
    :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
             :OrigPosition 590 15
             :Tiling false
             }
```

```
// -----------------------------------------------------------------

{:Rectangle 4000
 :OrigBoxSize 100 100 :OrigPosition 20 130
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Red :OrigRefFillColour DarkRed
}
{:Rectangle 4001
 :OrigBoxSize 100 100 :OrigPosition 60 170
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Blue :OrigRefFillColour DarkBlue
}
{:Rectangle 4002
 :OrigBoxSize 100 100 :OrigPosition 100 150
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Yellow :OrigRefFillColour DarkYellow
}
{:Rectangle 4003
 :OrigBoxSize 100 100 :OrigPosition 70 100
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Magenta :OrigRefFillColour DarkMagenta
}

// ------------------------------------------------------------

{:Link 2000
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect ( :PutBehind ( 4003 4000 ) )
}
{:Link 2001
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect ( :PutBehind ( 4002 4000 ) )
}
{:Link 2002
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect ( :PutBehind ( 4001 4000 ) )
}

{:Link 2003
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect ( :PutBefore ( 4003 4000 ) )
}
```

```
  {:Link 2004
   :EventSource 0 :EventType UserInput :EventData 10
   :LinkEffect ( :PutBefore ( 4002 4000 ) )
  }
  {:Link 2005
   :EventSource 0 :EventType UserInput :EventData 11
   :LinkEffect ( :PutBefore ( 4001 4000 ) )
  }

  // ----------------------------------------------------------

  {:PushButton 9001
   :OrigBoxSize   100  60
   :OrigPosition  540 280
   :ButtonRefColour gray
   :OrigLabel "back to main"
  }

  {:Link 9002
   :EventSource 9001 :EventType IsSelected
   :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
  }

  {:Link 9003
   :EventSource 9001 :EventType CursorEnter
   :LinkEffect ( :Activate( 9005 ) )
  }
  {:Link 9004
   :EventSource 9001 :EventType CursorLeave
   :LinkEffect ( :DeActivate( 9005 ) )
  }
  {:Link 9005
     :InitiallyActive false
     :EventSource 0
     :EventType UserInput
     :EventData 15
     :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                 )
  }

)

:InputEventReg 1
```

```
 :SceneCS 650 350
 :MovingCursor true
}
```

disp2.mhg:

```
{:Scene ( "demo/disp2.mhg" 0 )

 :Items
 (
  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }

   {:Text 1000
    :OrigContent  'Testing the Displaystack, 2.Test:'
    :OrigBoxSize 500 30
    :OrigPosition 90 20
    :FontAttributes Bold.24 :FontName Proportional
   }

   {:Text 1001
    :OrigContent  'key1: bring to front: green rectangle
key2: bring to front: gray rectangle
key3: bring to front: blue rectangle
key4: bring to front: red rectangle
key5: bring to front: yellow rectangle
key6: send to back: green rectangle
key7: send to back: gray rectangle
key8: send to back: blue rectangle
key9: send to back: red rectangle
key0: send to back: yellow rectangle'
    :OrigBoxSize 300 220
    :OrigPosition 330 60
    :FontAttributes Plain.18 :FontName Proportional
    :TextWrapping false
   }
```

```
{:Bitmap 101
 :OrigContent :ContentRef ( "demo/tu_klein.gif" )
         :OrigBoxSize 51 39     // 0 0
         :OrigPosition 10 15
         :Tiling false
         }

{:Bitmap 102
 :OrigContent :ContentRef ( "demo/fsp_pv.gif")
         :OrigBoxSize 48 43
         :OrigPosition 540 15
         :Tiling false
         }

{:Bitmap 103
 :OrigContent :ContentRef ( "demo/prz.gif" )
         :OrigBoxSize 48 43
         :OrigPosition 590 15
         :Tiling false
         }


// ----------------------------------------------------------------

{:Rectangle 4000
 :OrigBoxSize 80 260 :OrigPosition 110 70
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Green :OrigRefFillColour transparent
}
{:Rectangle 4001
 :OrigBoxSize 200 200 :OrigPosition 50 100
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour DarkGray :OrigRefFillColour Gray
}
{:Rectangle 4002
 :OrigBoxSize 100 100 :OrigPosition 100 150
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Blue :OrigRefFillColour transparent
}
{:Rectangle 4003
 :OrigBoxSize 150 150 :OrigPosition 150 190
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Red :OrigRefFillColour DarkRed
}
{:Rectangle 4004
 :OrigBoxSize 280 170 :OrigPosition 10 60
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Yellow :OrigRefFillColour transparent
}

// ------------------------------------------------------------------


{:Link 1005
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect ( :BringToFront ( 4000 ) )
}
{:Link 1006
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect ( :BringToFront ( 4001 ) )
}
{:Link 1007
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect ( :BringToFront ( 4002 ) )
}
{:Link 1008
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect ( :BringToFront ( 4003 ) )
}
{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 10
 :LinkEffect ( :BringToFront ( 4004 ) )
}

{:Link 1010
 :EventSource 0 :EventType UserInput :EventData 11
 :LinkEffect ( :SendToBack ( 4000 ) )
}
{:Link 1011
 :EventSource 0 :EventType UserInput :EventData 12
 :LinkEffect ( :SendToBack ( 4001 ) )
}
{:Link 1012
 :EventSource 0 :EventType UserInput :EventData 13
 :LinkEffect ( :SendToBack ( 4002 ) )
```

```
  }
  {:Link 1013
   :EventSource 0 :EventType UserInput :EventData 14
   :LinkEffect ( :SendToBack ( 4003 ) )
  }
  {:Link 1014
   :EventSource 0 :EventType UserInput :EventData 5
   :LinkEffect ( :SendToBack ( 4004 ) )
  }

  // ----------------------------------------------------------

  {:PushButton 9001
   :OrigBoxSize   100  60
   :OrigPosition  540 280
   :ButtonRefColour gray
   :OrigLabel "back to main"
  }

  {:Link 9002
   :EventSource 9001 :EventType IsSelected
   :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
  }

  {:Link 9003
   :EventSource 9001 :EventType CursorEnter
   :LinkEffect ( :Activate( 9005 ) )
  }
  {:Link 9004
   :EventSource 9001 :EventType CursorLeave
   :LinkEffect ( :DeActivate( 9005 ) )
  }
  {:Link 9005
     :InitiallyActive false
     :EventSource 0
     :EventType UserInput
     :EventData 15
     :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                 )
  }

)
```

```
 :InputEventReg 1
 :SceneCS 650 350
 :MovingCursor true
}
```

text.mhg:

```
{:Scene ( "demo/text.mhg" 0 )

 :Items
 (
   {:Text 1000
    :OrigContent  'Testing the Text Features:'
    :OrigBoxSize 500 30
    :OrigPosition 80 10
    :FontAttributes Bold.24 :FontName Proportional
   }


   {:Text 3000
    :OrigContent  'Text 1: Plain.18
TextWrapping = false'
    :OrigBoxSize 220 110
    :OrigPosition 10 50
    :OrigFont 'OctetString in OriginalFont'
    :FontAttributes Plain.18
    :FontName Proportional
    :TextColour black
    :BackgroundColour white
    :CharacterSet 1848
    :HJustification start
    :VJustification start
    :LineOrientation horizontal
    :StartCorner upper-left
    :TextWrapping false
   }

  {:Bitmap 101
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 5
             :Tiling false
```

```
           }

 {:Bitmap 102
  :OrigContent :ContentRef ( "demo/fsp_pv.gif")
           :OrigBoxSize 48 43
           :OrigPosition 625 5
           :Tiling false
           }

 {:Bitmap 103
  :OrigContent :ContentRef ( "demo/prz.gif" )
           :OrigBoxSize 48 43
           :OrigPosition 675 5
           :Tiling false
           }


{:Link  110
  :EventSource 0
  :EventType UserInput
  :EventData 16
  :LinkEffect (
                :Quit (( "demo/startup" 0 ))
              )
 }

 {:HotSpot 3100
  :OrigBoxSize 220 110
  :OrigPosition 10 50
 }

 {:Text 3001
  :OrigContent  'Text 2: Bold.18,
                 HJustification = end,
                 VJustification = end.'
  :OrigBoxSize 220 110
  :OrigPosition 250 50
  :OrigFont 'OctetString in OriginalFont'
  :FontAttributes Bold.18
  :TextColour black
  :BackgroundColour white
  :CharacterSet 1848
  :HJustification end
  :VJustification end
  :LineOrientation horizontal
  :StartCorner upper-left
  :TextWrapping true
 }
 {:HotSpot 3101
  :OrigBoxSize 220 110
  :OrigPosition 250 50
 }

 {:Text 3002
  :OrigContent  'Text 3: Italic.18,
                 HJustification = centre,
                 StartCorner = upper-left'
  :OrigBoxSize 220 110
  :OrigPosition 490 50
  :OrigFont 'OctetString in OriginalFont'
  :FontAttributes Italic.18
  :TextColour black
  :BackgroundColour white
  :CharacterSet 1848
  :HJustification centre
  :VJustification end
  :LineOrientation horizontal
  :StartCorner upper-left
  :TextWrapping true
 }
 {:HotSpot 3102
  :OrigBoxSize 220 110
  :OrigPosition 490 50
 }

 {:Text 3003
  :OrigContent  :ContentRef ( "demo/to_load.txt" )
  :OrigBoxSize 220 110
  :OrigPosition 10 180
  :OrigFont 'OctetString in OriginalFont'
  :FontAttributes Bold-Italic.18
  :TextColour black
  :BackgroundColour white
  :CharacterSet 1848
  :HJustification centre
  :VJustification end
  :LineOrientation horizontal
  :StartCorner upper-left
  :TextWrapping true
 }
 {:HotSpot 3103
  :OrigBoxSize 220 110
  :OrigPosition 10 180
 }

 {:Text 3004
  :OrigContent  'Text 5: Emphasis.18,
                 HJustification = justified,
                 VJustification = justified.'
  :OrigBoxSize 220 110
  :OrigPosition 250 180
  :OrigFont 'OctetString in OriginalFont'
  :FontAttributes Emphasis.18
  :TextColour black
  :BackgroundColour white
  :CharacterSet 1848
  :HJustification justified
  :VJustification justified
  :LineOrientation horizontal
  :StartCorner upper-left
  :TextWrapping true
 }
 {:HotSpot 3104
  :OrigBoxSize 220 110
  :OrigPosition 250 180
  }

 {:Text 3005
  :OrigContent  'Text 6: black on white, Strong.18,
                 horizontally and vertically centered.'
  :OrigBoxSize 220 110
  :OrigPosition 490 180
  :FontAttributes Strong.18
  :TextColour black
  :BackgroundColour white
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }
```

```
{:HotSpot 3105
 :OrigBoxSize 220 110
 :OrigPosition 490 180
}

{:Text 3006
 :OrigContent  'Text 7: black on white, Plain.10,
                horizontally and vertically centered.'
 :OrigBoxSize 220 110
 :OrigPosition 10 310
 :FontAttributes Plain.10
 :TextColour black
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}
{:HotSpot 3106
 :OrigBoxSize 220 110
 :OrigPosition 10 310
}

{:Text 3007
 :OrigContent  'Text 8: black on white, Italic.24,
                horizontally and vertically centered.'
 :OrigBoxSize 220 110
 :OrigPosition 250 310
 :FontAttributes Italic.24
 :TextColour black
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}
{:HotSpot 3107
 :OrigBoxSize 220 110
 :OrigPosition 250 310
}

{:Text 3008
 :OrigContent  'Text 9: black on white, Bold.12, Fixed,
                horizontally and vertically centered.'
 :OrigBoxSize 220 110
```

```
     :OrigPosition 490 310
     :FontAttributes Bold.12
     :FontName Fixed
     :TextColour black
     :BackgroundColour white
     :HJustification centre
     :VJustification centre
     :TextWrapping true
    }
    {:HotSpot 3108
     :OrigBoxSize 220 110
     :OrigPosition 490 310
    }

    // ----------------------------------------------------------

    {:Rectangle 2000
     :OrigBoxSize 224 114 :OrigPosition 8 48
     :OrigLineWidth 2 :OrigLineStyle 1
     :OrigRefLineColour Red
     :OrigRefFillColour transparent
    }

    {:IntegerVar 32000 :OrigValue 350 }
    {:IntegerVar 32001 :OrigValue 120 }
    {:ObjectRefVar 32002 :OrigValue :ObjectRef 3000 }   // the text
    {:ObjectRefVar 32003 :OrigValue :ObjectRef 3100 }   // the hotspot

    {:IntegerVar 2001  :OrigValue 8 }
    {:IntegerVar 2002  :OrigValue 48 }
    {:IntegerVar 2003  :OrigValue 224 }
    {:IntegerVar 2004  :OrigValue 114 }


    {:Link 1004
     :EventSource 0 :EventType UserInput :EventData 1
     :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                   :GetBoxSize (:IndirectRef 32002 2003 2004 )
                   :Subtract ( 32001 1 )
                   :Add       ( 2003 4 )
                   :Add       ( 2004 3 )
                   :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef
32001 )
```

```
                    :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef
32001 )
                    :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                    :BringToFront (2000)
                    :BringToFront (:IndirectRef 32003)
                  )
   }
   {:Link 1005
    :EventSource 0 :EventType UserInput :EventData 2
    :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                  :GetBoxSize (:IndirectRef 32002 2003 2004 )
                  :Add ( 32001 1 )
                  :Add      ( 2003 4 )
                  :Add      ( 2004 5 )
                  :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef
32001 )
                  :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef
32001 )
                  :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                  :BringToFront (2000)
                  :BringToFront (:IndirectRef 32003)
                )
   }
   {:Link 1006
    :EventSource 0 :EventType UserInput :EventData 3
    :LinkEffect ( :GetBoxSize (:IndirectRef 32002  32000 32001 )
                  :GetBoxSize (:IndirectRef 32002 2003 2004 )
                  :Subtract ( 32000 1 )
                  :Add      ( 2003 3 )
                  :Add      ( 2004 4 )
                  :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef
32001 )
                  :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef
32001 )
                  :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                  :BringToFront (2000)
                  :BringToFront (:IndirectRef 32003)
                )
   }
   {:Link 1007
    :EventSource 0 :EventType UserInput :EventData 4
    :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                  :GetBoxSize (:IndirectRef 32002 2003 2004 )
                  :Add ( 32000 1 )
                  :Add       ( 2003 5 )
                  :Add       ( 2004 4 )
                  :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef
32001 )
                  :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef
32001 )
                  :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                  :BringToFront (2000)
                  :BringToFront (:IndirectRef 32003)

                )
   }


   // ----------------------------------------------------------

   {:Link 1008
    :EventSource 0 :EventType UserInput :EventData 6
    :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                   :SetVariable ( 32003  :GObjectRef 3100 )
                       :GetPosition ( :IndirectRef 32002 2001 2002 )
                       :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                       :Subtract ( 2001 2 )
                       :Subtract ( 2002 2 )
                       :Add      ( 2003 4 )
                       :Add      ( 2004 4 )
                       :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                       :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                       :BringToFront ( 2000 )
                       :BringToFront ( :IndirectRef 32002 )
                       :BringToFront ( :IndirectRef 32003 )

                )
   }
   {:Link 1108
    :EventSource 3100 :EventType IsSelected
    :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                   :SetVariable ( 32003  :GObjectRef 3100 )
                       :GetPosition ( :IndirectRef 32002 2001 2002 )
                       :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                       :Subtract ( 2001 2 )
```

```
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3001 )
                :SetVariable ( 32003  :GObjectRef 3101 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1109
 :EventSource 3101 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3001 )
                :SetVariable ( 32003  :GObjectRef 3101 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1010
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect (   :SetVariable ( 32002  :GObjectRef 3002 )
                :SetVariable ( 32003  :GObjectRef 3102 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1110
 :EventSource 3102 :EventType IsSelected
 :LinkEffect (   :SetVariable ( 32002  :GObjectRef 3002 )
                :SetVariable ( 32003  :GObjectRef 3102 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---
```

```
{:Link 1011
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3003 )
                :SetVariable ( 32003  :GObjectRef 3103 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1111
 :EventSource 3103 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3003 )
                :SetVariable ( 32003  :GObjectRef 3103 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1012
 :EventSource 0 :EventType UserInput :EventData 10
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3004 )
                :SetVariable ( 32003  :GObjectRef 3104 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
```

```
                      :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                      :Subtract ( 2001 2 )
                      :Subtract ( 2002 2 )
                      :Add      ( 2003 4 )
                      :Add      ( 2004 4 )
                      :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                      :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                      :BringToFront ( 2000 )
                      :BringToFront ( :IndirectRef 32002 )
                      :BringToFront ( :IndirectRef 32003 )
              )
}
{:Link 1112
 :EventSource 3104 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3004 )
                :SetVariable ( 32003  :GObjectRef 3104 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
              )
}

// ---

{:Link 1013
 :EventSource 0 :EventType UserInput :EventData 11
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3005 )
                :SetVariable ( 32003  :GObjectRef 3105 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1113
 :EventSource 3105 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3005 )
                :SetVariable ( 32003  :GObjectRef 3105 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1014
 :EventSource 0 :EventType UserInput :EventData 12
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3006 )
                :SetVariable ( 32003  :GObjectRef 3106 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )                     )
}
{:Link 1114
```

```
 :EventSource 3106 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3006 )
                :SetVariable ( 32003  :GObjectRef 3106 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
            )
}

// ---

{:Link 1015
 :EventSource 0 :EventType UserInput :EventData 13
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3007 )
                :SetVariable ( 32003  :GObjectRef 3107 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
            )
}
{:Link 1115
 :EventSource 3107 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3007 )
                :SetVariable ( 32003  :GObjectRef 3107 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1016
 :EventSource 0 :EventType UserInput :EventData 14
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3008 )
                :SetVariable ( 32003  :GObjectRef 3108 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1116
 :EventSource 3108 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3008 )
                :SetVariable ( 32003  :GObjectRef 3108 )
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
             )
}


// ----------------------------------------------------------


{:Text 9000
 :OrigContent  'Press key(n), or click on a Text.
      To change the size of a text presentation,
      press cursor keys.'
 :OrigBoxSize 550 110
 :OrigPosition 10 430
 :FontAttributes Plain.18 :FontName Proportional
 :TextWrapping true
}

{:PushButton 9001
 :OrigBoxSize   100  60
 :OrigPosition  600 430
 :ButtonRefColour gray
 :OrigLabel "back to main"
}

{:Link 9002
 :EventSource 9001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
}

{:Link 9003
 :EventSource 9001 :EventType CursorEnter
 :LinkEffect ( :Activate( 9005 ) )
}
{:Link 9004
 :EventSource 9001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 9005 ) )
}
{:Link 9005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
```

```
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                  )
   }

 )

 :InputEventReg 1
 :SceneCS 730 500
 :MovingCursor true
}
```

intact.mhg:

```
{:Scene ( "demo/intact.mhg" 0 )

 :Items
 (
  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }
   {:Text 1000
    :OrigContent  'Testing interactible Objects:'
    :OrigBoxSize 500 30
    :OrigPosition 80 10
    :FontAttributes Bold.24 :FontName Proportional
   }

  {:Bitmap 101
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 5
             :Tiling false
             }

   {:Bitmap 102
    :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
```

```
          :OrigPosition 545 5
          :Tiling false
          }

{:Bitmap 103
 :OrigContent :ContentRef ( "demo/prz.gif" )
          :OrigBoxSize 48 43
          :OrigPosition 595 5
          :Tiling false
          }



{:Text 1001
 :OrigContent  'Test of interacting Objects. Use Cursor keys to resize
                the EntryField above.'
 :OrigBoxSize 180 200
 :OrigPosition 340 270
 :FontAttributes Plain.18 :FontName Proportional
 :HJustification justified
 :TextWrapping true
}


{:EntryField 3001
 :OrigContent 'A simple Entryfield.'
 :OrigBoxSize 300 100
 :OrigPosition 10 50
 :ObscuredInput true
}

//   'A more complex one - vertically and horizontally centered'
{:EntryField 3002
 :OrigContent 'A more complex one - vertically and horizontally centered'
 :OrigBoxSize 300 150
 :OrigPosition 335 50
 :FontAttributes Bold.16
 :FontName Proportional
 :TextColour red
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
}
```

```
{:OStringVar 3003
 :OrigValue  "void"
}

{:Link 3004
 :EventSource 3002 :EventType InteractionCompleted
 :LinkEffect ( :GetTextData ( 3002 3003 )
               :SetData (3005 :indirectref 3003) )
}

{:Text 3005
 :OrigContent  'void'
 :OrigBoxSize 300 50
 :OrigPosition 335 210
 :FontAttributes Plain.10 :FontName Proportional
}


// ----------------------------------------------------------


{:Text 4000
 :OrigContent  'A Pushbutton:'
 :OrigBoxSize 120 30
 :OrigPosition 15 173
 :FontAttributes Plain.18 :FontName Proportional
}

{:PushButton 4001
 :OrigBoxSize  120 30
 :OrigPosition  180 170
 :ButtonRefColour darkYellow
 :OrigLabel "A PushButton"
}

{:Rectangle 4002
 :InitiallyActive false
 :OrigBoxSize 30 30
 :OrigPosition 130 170
 :OrigLineWidth 5
 :OrigLineStyle 1
 :OrigRefLineColour DarkRed
 :OrigRefFillColour Red
}
```

```
// ----

{:Text 4010
 :OrigContent  'A HotSpot:'
 :OrigBoxSize 120 30
 :OrigPosition 15 213
 :FontAttributes Plain.18 :FontName Proportional
}

{:HotSpot 4011
 :OrigBoxSize  120 30
 :OrigPosition   180 210
}

{:Rectangle 4012
 :InitiallyActive false
 :OrigBoxSize 30 30
 :OrigPosition 130 210
 :OrigLineWidth 5
 :OrigLineStyle 1
 :OrigRefLineColour DarkRed
 :OrigRefFillColour Red
}

// ----

{:Text 4020
 :OrigContent  '3 SwitchButtons:'
 :OrigBoxSize 135 30
 :OrigPosition 15 293
 :FontAttributes Plain.18 :FontName Proportional
}

{:SwitchButton 4021
 :OrigBoxSize  40 30
 :OrigPosition   180 290
 :OrigLabel "A SwitchButton"
 :ButtonStyle pushbutton
}
{:SwitchButton 4022
 :OrigBoxSize  20 20
 :OrigPosition   230 295
 :OrigLabel "A SwitchButton"
 :ButtonStyle radiobutton
}
{:SwitchButton 4023
 :OrigBoxSize  20 20
 :OrigPosition   270 295
 :OrigLabel "A SwitchButton"
 :ButtonStyle checkbox
}

{:Text 4025
 :OrigContent  'A Slider:'
 :OrigBoxSize 135 30
 :OrigPosition 15 333
 :FontAttributes Plain.18 :FontName Proportional
}

{:Slider 4026
 :OrigBoxSize  120 30
 :OrigPosition   180 330
 :Orientation right
 :MaxValue 100
 :InitialPortion 50
 :SliderStyle proportional
}

//{:Rectangle 4029
// :InitiallyActive false
// :OrigBoxSize 30 30
// :OrigPosition 130 250
// :OrigLineWidth 5
// :OrigLineStyle 1
// :OrigRefLineColour DarkRed
// :OrigRefFillColour Red
//}

// ----------------------------------------------------------

{:Text 4040
 :OrigContent  'Show Highlights:'
 :OrigBoxSize 150 30
 :OrigPosition 15 253
 :FontAttributes Plain.18 :FontName Proportional
}
```

```
{:PushButton 4041
 :OrigBoxSize   60  30
 :OrigPosition 180 250
 :ButtonRefColour gray
 :OrigLabel "On"
}

{:PushButton 4051
 :OrigBoxSize   60  30
 :OrigPosition 240 250
 :ButtonRefColour gray
 :OrigLabel "Off"
}

// -----------------------------------------------------------


{:Link 5000
 :EventSource 4041 :EventType IsSelected
 :LinkEffect ( :SetHighlightStatus ( 4001 true )
               :SetHighlightStatus ( 4011 true )
             )
}
{:Link 5001
 :EventSource 4051 :EventType IsSelected
 :LinkEffect ( :SetHighlightStatus ( 4001 false )
               :SetHighlightStatus ( 4011 false )
             )
}

// -----------------------------------------------------------

{:Link 5010
 :EventSource 4001 :EventType IsSelected
 :LinkEffect ( :Run ( 4002 ) )
}
{:Link 5011
 :EventSource 4001 :EventType IsDeselected
 :LinkEffect ( :Stop ( 4002 ) )
}

{:Link 5020
 :EventSource 4011 :EventType IsSelected
 :LinkEffect ( :Run ( 4012 ) )
}
{:Link 5021
 :EventSource 4011 :EventType IsDeselected
 :LinkEffect ( :Stop ( 4012 ) )
}

// -------------------------------------------------------------



{:IntegerVar 32000 :OrigValue -1 }
{:IntegerVar 32001 :OrigValue -1 }

{:Link 1020
 :EventSource 0 :EventType UserInput :EventData 4
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
              :Add        ( 32000 4 )
              :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1021
 :EventSource 0 :EventType UserInput :EventData 3
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
              :Subtract   ( 32000 4 )
              :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1022
 :EventSource 0 :EventType UserInput :EventData 2
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
              :Add        ( 32001 4 )
              :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1023
 :EventSource 0 :EventType UserInput :EventData 1
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
              :Subtract   ( 32001 4 )
              :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}

// -------------------------------------------------------------

{:PushButton 9001
 :OrigBoxSize   100  60
 :OrigPosition  540 300
 :ButtonRefColour gray
 :OrigLabel "back to main"
}
```

```
   {:Link 9002
    :EventSource 9001 :EventType IsSelected
    :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
   }

   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                  )
   }

 )

 :InputEventReg 1
 :SceneCS       650 370
 :MovingCursor  true
}
```

bitmap1.mhg:

```
{:Scene ( "demo/bitmap.mhg" 0 )

  :Items (
       {:Text 100
        :OrigContent  'Testing the drawing of Bitmaps, 1.Test:'
        :OrigBoxSize 500 30
        :OrigPosition 80 10
        :FontAttributes Bold.24
        :FontName Proportional
       }
```

```
    {:Bitmap 101
          :OrigContent :ContentRef ( "demo/tu_klein.gif" )
          :OrigBoxSize 51 39     // 0 0
          :OrigPosition 10 5
          :Tiling false
          }

    {:Bitmap 102
     :OrigContent :ContentRef ( "demo/fsp_pv.gif")
          :OrigBoxSize 48 43
          :OrigPosition 595 5
          :Tiling false
          }

    {:Bitmap 103
     :OrigContent :ContentRef ( "demo/prz.gif" )
          :OrigBoxSize 48 43
          :OrigPosition 645 5
          :Tiling false
          }

{:Link  110
  :EventSource 0
  :EventType UserInput
  :EventData 16
  :LinkEffect (
               :Quit (( "demo/startup" 0 ))
             )
 }
        // ----------------------------

        {:Bitmap 3000
           :OrigContent
           :ContentRef ( "demo/pirogue.jpg" )
           :OrigBoxSize 158 158
           :OrigPosition 10 100
           :Tiling false
           }
        {:HotSpot 3001
           :OrigBoxSize 158 158
           :OrigPosition 10 100
        }
        {:Text 3002
```

```
   :OrigContent  'A JPEG-encoded picture, drawn in another size:'
   :OrigBoxSize 330 40
   :OrigPosition 10 48
   :FontAttributes Plain.18
   :FontName Proportional
   :TextWrapping true
}


// ----------------------------

{:Bitmap 3100
   :OrigContent
   :ContentRef ( "demo/pirogue.gif" )
   :OrigBoxSize 158 158
   :OrigPosition 350 100
   :Tiling false
   }
{:HotSpot 3101
   :OrigBoxSize 158 158
   :OrigPosition 350 100
}
{:Text 3102
   :OrigContent  'The GIF-encoded picture rescaled:'
   :OrigBoxSize 300 20
   :OrigPosition 350 48
   :FontAttributes Plain.18 :FontName Proportional
   :TextWrapping true
}

// ----------------------------

{:Bitmap 3200
   :OrigContent
   :ContentRef ( "demo/pirogue.jpg" )
   :OrigBoxSize 0 0
   :OrigPosition 10 310
   :Tiling false
   }
{:Text 3202
   :OrigContent  'The JPEG-encoded picture in original size:'
   :OrigBoxSize 320 20
   :OrigPosition 10 270
   :FontAttributes Plain.18 :FontName Proportional
}

// ----------------------------

{:Bitmap 3300
   :OrigContent
   :ContentRef ( "demo/pirogue.gif" )
   :OrigBoxSize 0 0
   :OrigPosition 350 310
   :Tiling false
   }
{:Text 3302
   :OrigContent  'The GIF-encoded picture in original size:'
   :OrigBoxSize 320 20
   :OrigPosition 350 270
   :FontAttributes Plain.18 :FontName Proportional
}

// ----------------------------------------------------------

{:Rectangle 2000
 :OrigBoxSize 162 162 :OrigPosition 8 98
 :OrigLineWidth 2 :OrigLineStyle 1
 :OrigRefLineColour Red
 :OrigRefFillColour transparent
}

{:IntegerVar 32000 :OrigValue 158 }
{:IntegerVar 32001 :OrigValue 158 }
{:ObjectRefVar 32002 :OrigValue :ObjectRef 3000 }        // the bitmap
{:ObjectRefVar 32003 :OrigValue :ObjectRef 3001 }        // the hotspot

{:IntegerVar 2001  :OrigValue 8 }
{:IntegerVar 2002  :OrigValue 98 }
{:IntegerVar 2003  :OrigValue 162 }
{:IntegerVar 2004  :OrigValue 162 }


{:Link 1004
 :EventSource 0 :EventType UserInput :EventData 1
 :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
               :GetBoxSize (:IndirectRef 32002 2003 2004 )
```

```
                         :Subtract ( 32001 1 )
                         :Add       ( 2003 4 )
                         :Add       ( 2004 3 )
                         :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000
:IndirectRef 32001 )
                         :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000
:IndirectRef 32001 )
                         :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                         :BringToFront (2000)
                         :BringToFront (:IndirectRef 32003)
                  )
          }
          {:Link 1005
           :EventSource 0 :EventType UserInput :EventData 2
           :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                         :GetBoxSize (:IndirectRef 32002 2003 2004 )
                         :Add ( 32001 1 )
                         :Add       ( 2003 4 )
                         :Add       ( 2004 5 )
                         :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000
:IndirectRef 32001 )
                         :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000
:IndirectRef 32001 )
                         :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                         :BringToFront (2000)
                         :BringToFront (:IndirectRef 32003)
                  )
          }
          {:Link 1006
           :EventSource 0 :EventType UserInput :EventData 3
           :LinkEffect ( :GetBoxSize (:IndirectRef 32002  32000 32001 )
                         :GetBoxSize (:IndirectRef 32002 2003 2004 )
                         :Subtract ( 32000 1 )
                         :Add       ( 2003 3 )
                         :Add       ( 2004 4 )
                         :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000
:IndirectRef 32001 )
                         :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000
:IndirectRef 32001 )
                         :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                         :BringToFront (2000)
                         :BringToFront (:IndirectRef 32003)
```

```
        )
 }
 {:Link 1007
  :EventSource 0 :EventType UserInput :EventData 4
  :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                :GetBoxSize (:IndirectRef 32002 2003 2004 )
                :Add ( 32000 1 )
                :Add       ( 2003 5 )
                :Add       ( 2004 4 )
                :SetBoxSize (:IndirectRef 32002
                   :IndirectRef 32000 :IndirectRef 32001 )
                :SetBoxSize (:IndirectRef 32002
                   :IndirectRef 32000 :IndirectRef 32001 )
                :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                :BringToFront (2000)
                :BringToFront (:IndirectRef 32003)
        )
 }


 // ----------------------------------------------------------

 {:Link 1008
  :EventSource 0 :EventType UserInput :EventData 6
  :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                 :SetVariable ( 32003  :GObjectRef 3001 )
                 :GetPosition ( :IndirectRef 32002 2001 2002 )
                 :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                 :Subtract ( 2001 2 )
                 :Subtract ( 2002 2 )
                 :Add      ( 2003 4 )
                 :Add      ( 2004 4 )
                 :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                 :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                 :BringToFront ( 2000 )
                 :BringToFront ( :IndirectRef 32002 )
                 :BringToFront ( :IndirectRef 32003 )
              )
}
{:Link 1108
 :EventSource 3001 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                :SetVariable ( 32003  :GObjectRef 3001 )
```

```
                    :GetPosition ( :IndirectRef 32002 2001 2002 )
                    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                    :Subtract ( 2001 2 )
                    :Subtract ( 2002 2 )
                    :Add      ( 2003 4 )
                    :Add      ( 2004 4 )
                    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                    :BringToFront ( 2000 )
                    :BringToFront ( :IndirectRef 32002 )
                    :BringToFront ( :IndirectRef 32003 )
                )
}

// ---

{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect (   :SetVariable ( 32002  :GObjectRef 3100 )
                 :SetVariable ( 32003  :GObjectRef 3101 )
                 :GetPosition ( :IndirectRef 32002 2001 2002 )
                 :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                 :Subtract ( 2001 2 )
                 :Subtract ( 2002 2 )
                 :Add      ( 2003 4 )
                 :Add      ( 2004 4 )
                 :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                 :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                 :BringToFront ( 2000 )
                 :BringToFront ( :IndirectRef 32002 )
                 :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1109
 :EventSource 3101 :EventType IsSelected
 :LinkEffect (   :SetVariable ( 32002  :GObjectRef 3100 )
                 :SetVariable ( 32003  :GObjectRef 3101 )
                 :GetPosition ( :IndirectRef 32002 2001 2002 )
                 :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
                 :Subtract ( 2001 2 )
                 :Subtract ( 2002 2 )
                 :Add      ( 2003 4 )
                 :Add      ( 2004 4 )
```

```
                  :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
                  :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
                  :BringToFront ( 2000 )
                  :BringToFront ( :IndirectRef 32002 )
                  :BringToFront ( :IndirectRef 32003 )
              )
}

// -----------------------------------------------------

 {:Text 9000
  :OrigContent  'Press key1 or key2,
                 or click on one of the upper bitmaps.
                 To change the size of the
                 bitmap presentation, press cursor keys.'
  :OrigBoxSize 550 110
  :OrigPosition 10 490
  :FontAttributes Plain.18 :FontName Proportional
  :TextWrapping true
 }

{:PushButton 9001
  :OrigBoxSize   100  60
  :OrigPosition  590 480
  :ButtonRefColour gray
  :OrigLabel "back to main"
 }

 {:Link 9002
  :EventSource 9001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
 }

 {:Link 9003
  :EventSource 9001 :EventType CursorEnter
  :LinkEffect ( :Activate( 9005 ) )
 }
 {:Link 9004
  :EventSource 9001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 9005 ) )
 }
 {:Link 9005
  :InitiallyActive false
```

```
           :EventSource 0
           :EventType UserInput
           :EventData 15
           :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                  )
          }

    )

    :InputEventReg 1
    :SceneCS 700 560
    :MovingCursor true
}
```

bitmap2.mhg:

```
{:Scene ( "demo/bitmap2.mhg" 0 )

  :Items (
          {:Text 1000
           :OrigContent  'Testing the drawing of Bitmaps, 2.Test:'
           :OrigBoxSize 500 30
           :OrigPosition 80 10
           :FontAttributes Bold.24
           :FontName Proportional
          }

          {:Bitmap 101
             :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 5
             :Tiling false
             }

          {:Bitmap 102
             :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
             :OrigPosition 595 5
             :Tiling false
             }

          {:Bitmap 103
             :OrigContent :ContentRef ( "demo/prz.gif" )
```

```
                :OrigBoxSize 48 43
                :OrigPosition 645 5
                :Tiling false
                }

 {:Link  110
   :EventSource 0
   :EventType UserInput
   :EventData 16
   :LinkEffect (
                 :Quit (( "demo/startup" 0 ))
               )
  }

         {:Bitmap 3000
            :OrigContent
            :ContentRef ( "demo/colors.gif" )
            :OrigBoxSize 680 400
            :OrigPosition 10 50
            :Tiling false
            }


         {:Bitmap 3100
            :OrigContent
            :ContentRef ( "demo/capbla.gif" )
            :OrigBoxSize 180 208
            :OrigPosition 50 100
            :Tiling false
            }
         {:Bitmap 3200
            :OrigContent
            :ContentRef ( "demo/capblb.gif" )
            :OrigBoxSize  180 208
            :OrigPosition 150 100
            :Tiling false
            }
         {:Bitmap 3300
            :OrigContent
            :ContentRef ( "demo/capblc.gif" )
            :OrigBoxSize 180 208
            :OrigPosition 50 200
            :Tiling false
            }
         {:Bitmap 3400
            :OrigContent
            :ContentRef ( "demo/capbld.gif" )
            :OrigBoxSize 180 208
            :OrigPosition 150 200
            :Tiling false
            }

         // {:Bitmap 3500
         //   :OrigContent
         //   :ContentRef ( "demo/globe.gif" )
         //   :OrigBoxSize 0 0
         //   :OrigPosition 450 200
         //   :Tiling false
         //   }

         // ----------------------------

         {:Bitmap 5000
            :InitiallyActive false
            :OrigContent
            :ContentRef ( "demo/red_ball.gif" )
            :OrigBoxSize 50 50
            :OrigPosition 10 10
            :Tiling true
            }

         // ----------------------------

         {:IntegerVar 32000 :OrigValue 320 }
         {:IntegerVar 32001 :OrigValue 158 }

         {:Link 1020
            :EventSource 0 :EventType UserInput :EventData 4
            :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                          :Add        ( 32000 5 )
                          :SetBoxSize ( 5000
                            :IndirectRef 32000 :IndirectRef 32001 ))}

         {:Link 1021
            :EventSource 0 :EventType UserInput :EventData 3
            :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                          :Subtract   ( 32000 5 )
```

```
                  :SetBoxSize ( 5000
                   :IndirectRef 32000 :IndirectRef 32001 ))}

{:Link 1022
   :EventSource 0 :EventType UserInput :EventData 2
   :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                 :Add        ( 32001 5 )
                 :SetBoxSize ( 5000
                    :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1023
   :EventSource 0 :EventType UserInput :EventData 1
   :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                 :Subtract   ( 32001 5 )
                 :SetBoxSize ( 5000
                    :IndirectRef 32000 :IndirectRef 32001 ))}

// -------


{:Link 1030
   :EventSource 0 :EventType UserInput :EventData 14
   :LinkEffect ( :Run ( 5000 )
                 :BringToFront ( 9001 )
               )
}

{:Link 1031
   :EventSource 0 :EventType UserInput :EventData 5
   :LinkEffect ( :Stop ( 5000 )
               )
}

{:Link 2000
   :EventSource 0 :EventType UserInput :EventData 6
   :LinkEffect ( :BringToFront ( 3100 ) )
}
{:Link 2001
   :EventSource 0 :EventType UserInput :EventData 7
   :LinkEffect ( :BringToFront ( 3200 ) )
}
{:Link 2002
   :EventSource 0 :EventType UserInput :EventData 8
   :LinkEffect ( :BringToFront ( 3300 ) )
```

```
 }
 {:Link 2003
    :EventSource 0 :EventType UserInput :EventData 9
    :LinkEffect ( :BringToFront ( 3400 ) )
 }

 {:Link 2004
    :EventSource 0 :EventType UserInput :EventData 10
    :LinkEffect ( :PutBefore ( 3100 3000 ) )
 }
 {:Link 2005
    :EventSource 0 :EventType UserInput :EventData 11
    :LinkEffect ( :PutBefore ( 3200 3000 ) )
 }
 {:Link 2006
    :EventSource 0 :EventType UserInput :EventData 12
    :LinkEffect ( :PutBefore ( 3300 3000 ) )
 }
 {:Link 2007
    :EventSource 0 :EventType UserInput :EventData 13
    :LinkEffect ( :PutBefore ( 3400 3000 ) )
 }


 // ----------------------------------------------------------

 {:Text 9000
  :OrigContent  'Press Key9 to activate a tiled bitmap,
                    Key0 to deactivate it,
                 and Cursor Keys to resize that bitmap.'
  :OrigBoxSize 550 110
  :OrigPosition 10 480
  :FontAttributes Plain.18 :FontName Proportional
  :TextWrapping true
 }

{:PushButton 9001
  :OrigBoxSize   100  60
  :OrigPosition  590 460
  :ButtonRefColour gray
  :OrigLabel "back to main"
 }
```

```
          {:Link 9002
           :EventSource 9001 :EventType IsSelected
           :LinkEffect (   :TransitionTo( ( "demo/main.mhg" 0) ) )
          }

          {:Link 9003
           :EventSource 9001 :EventType CursorEnter
           :LinkEffect ( :Activate( 9005 ) )
          }
          {:Link 9004
           :EventSource 9001 :EventType CursorLeave
           :LinkEffect ( :DeActivate( 9005 ) )
          }
          {:Link 9005
           :InitiallyActive false
           :EventSource 0
           :EventType UserInput
           :EventData 15
           :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                       )
          }


    )

    :InputEventReg 1
    :SceneCS 700 530
    :MovingCursor true
}
```

ea.mhg:

```
{:Scene ( "demo/ea.mhg" 0 )

 :Items
 (

   {:Text 1000
     :OrigContent  'Testing "Elementary Actions":'
     :OrigBoxSize 500 30
     :OrigPosition 80 10
     :FontAttributes Bold.24 :FontName Proportional
   }
```

```
     {:Bitmap 101
             :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 5
             :Tiling false
             }

       {:Bitmap 102
        :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
             :OrigPosition 545 5
             :Tiling false
             }

       {:Bitmap 103
        :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
             :OrigPosition 595 5
             :Tiling false
             }

{:Link  110
  :EventSource 0
  :EventType UserInput
  :EventData 16
  :LinkEffect (
                :Quit (( "demo/startup" 0 ))
              )
 }

 // ----------------------------------------------------

 // ElementaryActions for Rectangle

 {:Rectangle 4000
  :OrigBoxSize 100 100 :OrigPosition 20 130
  :OrigLineWidth 5 :OrigLineStyle 1
  :OrigRefLineColour Red :OrigRefFillColour DarkRed
 }
 {:Rectangle 4001
  :OrigBoxSize 100 100 :OrigPosition 60 170
  :OrigLineWidth 5 :OrigLineStyle 1
  :OrigRefLineColour Blue :OrigRefFillColour DarkBlue
 }

 // apply all ElementaryActions possible for Rectangle
 {:Link 4102
  :EventSource 0 :EventType UserInput :EventData 6
  :LinkEffect (
                // Root
                   :GetAvailabilityStatus( 4000 2103)
                   :GetRunningStatus( 4000 2103 )
                // Ingredient
                   :Clone( 4000 4200 )
                   :Preload( 4000 )
                   :SetData( 4000 "new data_1")
                   :Unload( 4000 )
                // Presentable
                   :Run( 4000 )
                   :SetData( 4000 "new data_2" )
                   :Stop( 4000 )
                   :Run( 4000 )
                // Visible
                   :BringToFront( 4000 )
                   :GetBoxSize( 4000 2101 2102 )
                   :GetPosition ( 4000 2101 2102 )
                   :PutBefore(4000 4001)
                   :PutBehind(4000 4001)
                   :SendToBack(4000)
                   :SetBoxSize(4000 150 100 )
                   :SetPaletteRef(4000 100)
                   :SetPosition(4000 20 130)
                // LineArt
                   :SetFillColour (4000 :NewAbsoluteColour blue)
                   :SetLineColour (4000 :NewAbsoluteColour red)
                   :SetLineStyle (4000 3)
                   :SetLineWidth (4000 6)
                   :PutBehind(4000 4001)
              )
  }


 // ----------------------------------------------------------


 // ElementaryActions for PushButton
```

```
{:PushButton 5000
 :OrigBoxSize  100 100
 :OrigPosition  200 130
 :ButtonRefColour darkYellow
 :OrigLabel "A Pushbutton"
}

{:Link 5100
 :EventSource 0 :EventType UserInput :EventData 7
 //:EventSource 5000 :EventType IsSelected
 :LinkEffect (
                //Root:
                  :GetAvailabilityStatus(5000 2101)
                  :GetRunningStatus(5000 2103)
                //Ingredient:
                  :Clone(5000 5200)
                  :Preload(5000)
                  :SetData(5000 "setdata, ingredient")
                  :Unload(5000)
                //Presentable:
                  :Run(5000)
                  :SetData(5000 "setdata, presentable")
                  :Stop(5000)
                  :Run(5000)
                //Visible:
                  :BringToFront(5000)
                  :GetBoxSize(5000 2101 2102)
                  :GetPosition(5000 2101 2102)
                  :PutBefore(5000 4000)
                  :PutBehind(5000 4000)
                  :SendToBack(5000)
                  :SetBoxSize(5000 120 90)
                  :SetPaletteRef(5000 1)
                  :SetPosition(5000 190 120)
                //Button:
                  :Deselect(5000)
                  :GetInteractionStatus(5000 2101)
                  :Select(5000)
                  :SetInteractionStatus(5000  true)
                  :Deselect(5000)
                //PushButton:
                  :GetLabel(5000 5300)
                  :Select(5000)
```

```
                    :SetLabel(5000 "Label set by EA")
               )
}


 // -----------------------------------------------------

 // ElementaryActions for EntryField

 {:EntryField 2000
  :OrigContent 'A more complex one. vertically and horizontally centred.'
  :OrigBoxSize 300 100
  :OrigPosition 335 50
  :FontAttributes Bold.10 :FontName Proportional
  :TextColour red
  :BackgroundColour white
  :HJustification centre
  :VJustification centre
 }

 {:IntegerVar 2101 :OrigValue 1 }
 {:IntegerVar 2102 :OrigValue 1 }
 {:BooleanVar 2103 :OrigValue true}

// apply all ElementaryActions possible for Entryfield
 {:Link 2100
  :EventSource 0 :EventType UserInput :EventData 9
  :LinkEffect (
               // Root
                     :GetAvailabilityStatus( 2000 2101 )
                     :GetRunningStatus( 2000 2103 )
               // Ingredient
                     :Clone( 2000 4200 )
                     :Preload( 2000 )
                     :SetData( 2000 "new data_1")
                     :Unload( 2000 )
               // Presentable
                     :Run( 2000 )
                     :SetData( 2000 "new data_2" )
                     :Stop( 2000 )
                     :Run( 2000 )
               // Visible
                     :BringToFront( 2000 )
```

```
                        :GetBoxSize( 2000 2101 2102 )
                        :GetPosition ( 2000 2101 2102 )
                        :PutBefore(2000 4000)
                        :PutBehind(2000 4000)
                        :SendToBack(2000)
                        :SetBoxSize(2000 300 100 )
                        :SetPaletteRef(2000 100)
                        :SetPosition(2000 335 50)
                  // Text
                        :GetTextContent(2000 2101)
                        :GetTextData(2000 2101)
                        :SetData( 2000 "new data_3" )
                        //:SetFontRef(2000 "Bold:20")
                  // Interactible
                        :GetHighlightStatus(2000 2101)
                        :GetInteractionStatus(2000 2101)
                        :SetHighlightStatus(2000 true)
                        :SetInteractionStatus(2000 true)
                  // EntryField
                       :GetEntryPoint(2000 2101)
                       :GetOverwriteMode(2000 2101)
                       :SetEntryPoint(2000 5)
                       :SetOverwriteMode(2000 true)
                 )
  }

// ------------------------------------------------------------

  // ElementaryActions for DynamicLineArt

  {:DynamicLineArt 3000
   :OrigBoxSize 300 100
   :OrigPosition 335 160
  }

  // apply all ElementaryActions possible for DynamicLineArt
  {:Link 3100
   :EventSource 0 :EventType UserInput :EventData 8
   :LinkEffect (
                  // Root
                        :GetAvailabilityStatus( 3000 2101 )
                        :GetRunningStatus( 3000 2103 )
                  // Ingredient
```

```
            :Clone( 3000 4200 )
            :Preload( 3000 )
            :SetData( 3000 "new data_1")
            :Unload( 3000 )
      // Presentable
            :Run( 3000 )
            :SetData( 3000 "new data_2" )
            :Stop( 3000 )
            :Run( 3000 )
      // Visible
            :BringToFront( 3000 )
            :GetBoxSize( 3000 2101 2102 )
            :GetPosition ( 3000 2101 2102 )
            :PutBefore(3000 4000)
            :PutBehind(3000 4000)
            :SendToBack(3000)
            :SetBoxSize(3000 300 100 )
            :SetPaletteRef(3000 100)
            :SetPosition(3000 335 160)

      // LineArt
            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour blue)
            :SetLineStyle(3000 3)
            :SetLineWidth(3000 2)

      //DynamicLineArt:
            :BringToFront(3000)
            :Clear(3000)


     :SetPosition(3000 335 160)
            :PutBefore(3000 2000)
            :PutBehind(3000 2000)
            :SendToBack(3000)
            :SetBoxSize(3000 300 100)

            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour green)
            :SetLineStyle(3000 2)
            :SetLineWidth(3000 2)
            :DrawArc(3000 5 5 100 50 30 180)
```

```
                    :SetFillColour(3000 :NewAbsoluteColour  red)
                    :SetLineColour(3000 :NewAbsoluteColour blue)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 10)
                    :DrawLine(3000 5 5 300 100)

                    :SetFillColour(3000 :NewAbsoluteColour green)
                    :SetLineColour(3000 :NewAbsoluteColour yellow)
                    :SetLineStyle(3000 1)
                    :SetLineWidth(3000 2)
                    :DrawOval(3000 5 50 100 45)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour yellow)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 6)
                    :DrawPolygon(3000
                ((100 5) (100 45) ( 150 30) (199 45) (199 5)))

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour black)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 3)
                    :DrawPolyLine(3000
                ((100 60) (100 95) (150 80 ) (199 95) (199 60)))

                    :SetFillColour(3000 :NewAbsoluteColour gray)
                    :SetLineColour(3000 :NewAbsoluteColour red)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 3)
                    :DrawRectangle (3000 205 5  90 40)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour green)
                    :SetLineStyle(3000 3)
                    :SetLineWidth(3000 2)
                    :DrawSector(3000 205 50 90 45 -30 180 )



    )
}

{:IntegerVar 3101 :OrigValue 30 }      // start_angle for arc
```

```
{:IntegerVar 3102 :OrigValue 10 }       // linewidth for line
{:IntegerVar 3103 :OrigValue 100 }      // x-size for oval
{:IntegerVar 3104 :OrigValue 150 }      // xPos of a point of polygon
{:IntegerVar 3105 :OrigValue 80 }       // yPos of a point for polyline
{:IntegerVar 3106 :OrigValue 90 }       // xSize for rectangle
{:IntegerVar 3107 :OrigValue 180 }       // arc_angle for sector


{:Link 3200
 :EventSource 0 :EventType UserInput :EventData 1
 :LinkEffect (
                    :Add       ( 3101 5 )
                    :Add       ( 3102 1 )
                    :Add       ( 3103 10 )
                    :Add       ( 3104 5 )
                    :Add       ( 3105 5 )
                    :Add       ( 3106 10 )
                    :Add       ( 3107 10 )

                    :Clear(3000)
                    :GetBoxSize( 3000 2101 2102 )

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour green)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 2)
                    :DrawArc(3000 5 5 100 50  :IndirectRef 3101 180)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour blue)
                    :SetLineStyle(3000 3)
                    :SetLineWidth(3000  :IndirectRef 3102)
                    :DrawLine(3000 5 5 300 100)

                    :SetFillColour(3000 :NewAbsoluteColour green)
                    :SetLineColour(3000 :NewAbsoluteColour yellow)
                    :SetLineStyle(3000 1)
                    :SetLineWidth(3000 4)
                    :DrawOval(3000 5 50  :IndirectRef 3103 45)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour yellow)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 6)
                    :DrawPolygon(3000 ((100 5) (100 45)
                  (  :IndirectRef 3104 30) (199 45) (199 5)))

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour black)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 3)
                    :DrawPolyLine(3000 ((100 60) (100 95)
                  (150  :IndirectRef 3105 ) (199 95) (199 60)))

                    :SetFillColour(3000 :NewAbsoluteColour gray)
                    :SetLineColour(3000 :NewAbsoluteColour red)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 3)
                    :DrawRectangle (3000 205 5   :IndirectRef 3106 40)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour green)
                    :SetLineStyle(3000 3)
                    :SetLineWidth(3000 2)
                    :DrawSector(3000 205 50 90 45 -30  :IndirectRef 3107 )
         )
}
{:Link 3201
 :EventSource 0 :EventType UserInput :EventData 2
 :LinkEffect (
                    :Subtract       ( 3101 5 )
                    :Subtract       ( 3102 1 )
                    :Subtract       ( 3103 10 )
                    :Subtract       ( 3104 5 )
                    :Subtract       ( 3105 5 )
                    :Subtract       ( 3106 10 )
                    :Subtract       ( 3107 10 )

                    :Clear(3000)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour green)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 2)
                    :DrawArc(3000 5 5 100 50  :IndirectRef 3101 180)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour blue)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000  :IndirectRef 3102)
                    :DrawLine(3000 5 5 300 100)

                    :SetFillColour(3000 :NewAbsoluteColour green)
                    :SetLineColour(3000 :NewAbsoluteColour yellow)
                    :SetLineStyle(3000 1)
                    :SetLineWidth(3000 2)
                    :DrawOval(3000 5 50  :IndirectRef 3103 45)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour yellow)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 6)
                    :DrawPolygon(3000 ((100 5) (100 45)
                  (  :IndirectRef 3104 30) (199 45) (199 5)))

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour black)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 3)
                    :DrawPolyLine(3000 ((100 60) (100 95)
                  (150  :IndirectRef 3105 ) (199 95) (199 60)))

                    :SetFillColour(3000 :NewAbsoluteColour gray)
                    :SetLineColour(3000 :NewAbsoluteColour red)
                    :SetLineStyle(3000 2)
                    :SetLineWidth(3000 3)
                    :DrawRectangle (3000 205 5   :IndirectRef 3106 40)

                    :SetFillColour(3000 :NewAbsoluteColour red)
                    :SetLineColour(3000 :NewAbsoluteColour green)
                    :SetLineStyle(3000 3)
                    :SetLineWidth(3000 2)
                    :DrawSector(3000 205 50 90 45 -30  :IndirectRef 3107 )
           )
  }


// ----------------------------------------------------------
```

```
   {:PushButton 9001
    :OrigBoxSize   100  60
    :OrigPosition  540 280
    :ButtonRefColour gray
    :OrigLabel "back to main"
   }

   {:Link 9002
    :EventSource 9001 :EventType IsSelected
    :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
   }

   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                  )
   }

 )

 :InputEventReg 1
 :SceneCS 650 350
 :MovingCursor true
}
```

allcl.mhg:

```
{:Scene ( "demo/allcl.mhg" 0 )

 :Items
 (
   {:Text 1000
     :OrigContent  'Testing all concrete Classes and el.Actions:'
     :OrigBoxSize 500 30
     :OrigPosition 80 10
     :FontAttributes Bold.24 :FontName Proportional
   }

     {:Bitmap 101
             :OrigContent :ContentRef ( "demo/tu_klein.gif" )
             :OrigBoxSize 51 39     // 0 0
             :OrigPosition 10 5
             :Tiling false
             }

       {:Bitmap 102
        :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
             :OrigPosition 545 5
             :Tiling false
             }

       {:Bitmap 103
        :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
             :OrigPosition 595 5
             :Tiling false
             }

  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }
  // ------------------------------------------------------------

  // Root 1008
  // Group 1009
  // Application 1010
  // Scene 1011
  // Ingredient 1012
  // Link 1013

  // ---- Programs (1014) ----

  {:ResidentPrg 1015
   :Name "Hello World, I'm a ResidentProgram."
  }

  {:RemotePrg 1016
   :Name "Hello World, I'm a RemoteProgram."
  }

  {:InterchgPrg 1017
   :Name "Hello World, I'm a InterchangedProgram."
  }

  {:Palette 1018
  }

  {:Font 1019
  }

  {:CursorShape 1020
  }

  // ---- Variables 1021 ----

  {:BooleanVar 1022
   :OrigValue true
  }

  {:IntegerVar 1023
   :OrigValue 1
  }

  {:OStringVar 1024
   :OrigValue "hello world"
  }

  {:ObjectRefVar 1025
   :OrigValue :ObjectRef 1024
  }

  {:ContentRefVar 1026
```

```
 :OrigValue :ContentRef "demo/to_load.txt"
}

// ---- Presentables 1027 ----
// TokenManager 1028

{:TokenGroup 1029
 :TokenGroupItems ( ( 1024 ) )
}

{:ListGroup 1030
 :TokenGroupItems ( ( 1024 ) )
 :Positions ( ( 10 10 ) )
}

{:Stream 1037
 :Multiplex ( {:Audio 1038 :ComponentTag 100 }  )
}

{:Audio 1038
 :ComponentTag 100
}

// ---- Visibles ----

{:Text 2000
  :OrigContent  'Bitmap, LineArt, Rectangle, Dynamic- LineArt:'
  :OrigBoxSize 95 95
  :OrigPosition 0 50
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Bitmap 1032
 :OrigContent :ContentRef ( "demo/pirogue.jpg" )
 :OrigBoxSize 120 120
 :OrigPosition 100 50
}

{:LineArt 1033
 :OrigBoxSize 120 120
 :OrigPosition 225 50
}

{:Rectangle 1034
 :OrigBoxSize 120 120
 :OrigPosition 350 50
 :OrigLineWidth 5
 :OrigLineStyle 1
 :OrigRefLineColour Red
 :OrigRefFillColour Yellow
}

{:DynamicLineArt 1035
 :OrigBoxSize 120 120
 :OrigPosition 475 50
}

// ---------------------------

{:Text 2001
  :OrigContent  'Text, Video, RTGraphics:'
  :OrigBoxSize 95 95
  :OrigPosition 0 175
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Text 1036
 :OrigContent  "I'm a simple Text"
 :OrigBoxSize 120 120
 :OrigPosition 100 175
 :TextColour red
 :BackgroundColour black
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}

{:Video 1039
 :InitiallyActive true
 :OrigContent
 :ContentRef ( "demo/saao.m1v" )
 :OrigBoxSize 120 120
 :OrigPosition 225 175
 :ComponentTag 100
 :Termination loop
}

{:RTGraphics 1040
 :OrigBoxSize 120 120
 :OrigPosition 350 175
 :ComponentTag 100
}

// ---- Interactibles 1041 ----

{:Text 2002
  :OrigContent  'Slider, EntryField, HyperText:'
  :OrigBoxSize 95 95
  :OrigPosition 0 300
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Slider 1042
 :OrigBoxSize 120 120
 :OrigPosition 100 300
 :Orientation up
 :MaxValue 100
 :SliderStyle proportional
}

{:EntryField 1043
 :OrigBoxSize 120 120
 :OrigPosition 225 300
}

{:HyperText 1044
 :OrigContent 'HyperText: an anchor to
     <a href="http://www.mheg.org">Tom Caseys</a> site'
 :OrigBoxSize 120 120
 :OrigPosition 350 300

}

// ---- Buttons 1045 ----

{:Text 2003
  :OrigContent  'HotSpot, PushButton, SwitchButton:'
  :OrigBoxSize 95 95
  :OrigPosition 0 425
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Hotspot 1046
 :OrigBoxSize 120 120
 :OrigPosition 100 425
}

{:PushButton 1047
 :OrigBoxSize 120 120
 :OrigPosition 225 425
}

{:SwitchButton 1048
 :OrigBoxSize 120 120
 :OrigPosition 350 425
 :ButtonStyle radiobutton
}

// TODO:  ___TODO___:
//{:Action
// (:Activate 1000)
//}


// ----------------------------------------------------------


{:IntegerVar   3100 :OrigValue 1 }
```

```
{:IntegerVar   3101 :OrigValue 1 }
{:BooleanVar   3102 :OrigValue true }
{:ObjectRefVAr 3103 :OrigValue :ObjectRef 3000}

// apply all ElementaryActions possible for all classes
{:Link 3000
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect (
                // Root (using the bitmap: 1032)
                 :GetAvailabilityStatus( 1032 3102 )
                 :GetRunningStatus( 1032 3102 )

                // Scene
                 //:SendEvent(0 1)               // todo.
                 :GetCursorPosition(0 3100 3101)
                 :SetCursorPosition(0 100 100)
                 :SetCursorShape(1000 1000)
                 :SetTimer(0 1 2 :AbsoluteTime true)
                 //:TransitionTo()              // not so useful here.

               //Application:
                 //:CloseConnection()
                 //:GetEngineSupport()
                 //:Launch()
                 //:LockScreen()
                 //:OpenConnection()
                 //:Quit()
                 //:ReadPersistent()
                 //:Spawn()
                 //:StorePersistent()
                 //:UnlockScreen()

                // Ingredient (using the bitmap: 1032)
                 :Clone( 1032 1050 )
                 :Preload( 1032 )
                 :SetData( 1032 "new data_1")
                 :Unload( 1032 )

                //Link:
                 //:Activate()
                 //:Deactivate()

                //Program (using resident program: 1015)
                 //:Call(1015 )
                 //:Fork(1015)
                 //:SetData(1015)
                 //:Stop(1015)

                //Presentable (using the bitmap: 1032)
                 //:Run(1032)
                 //:SetData(1032)
                 //:Stop(1032)

                //Stream:
                 //:Clone()
                 :SetCounterEndPosition(1037 5)
                 :SetCounterPosition(1037 1)
                 :SetCounterTrigger(1037 2)
                 //:SetData(1037)
                 :SetSpeed(1037 25)

                //Visible (using the bitmap: 1032)
                 :BringToFront(1032)
                 :GetBoxSize(1032 3100 3101)
                 :GetPosition(1032 3100 3101)
                 :PutBefore(1032 1035)
                 :PutBehind(1032 1035)
                 :SendToBack(1032)
                 :SetBoxSize(1032 60 60)
                 //:SetPaletteRef(1032)
                 :SetPosition(1032 100 50)

                //RTGraphics:
                 //:Clone(1040)
                 //:SetData(1040)

                //LineArt:
                 //:SetFillColour(1033)
                 //:SetLineColour(1033)
                 :SetLineStyle(1033 3)
                 :SetLineWidth(1033 2)

                //DynamicLineArt:
                 :BringToFront(1035)
                 :Clear(1035)
                 :DrawArc(1035 10 10 30 30 0 180)
                 :DrawLine(1035 10 10 50 50)
                 :DrawOval(1035 20 20 50 30)
                 //:DrawPolygon(1035)
                 //:DrawPolyline(1035)
                 :DrawRectangle(1035 10 10 30 50)
                 :DrawSector(1035 30 30 40 40 90 180)
                 //:GetFillColour(1035)
                 //:GetLineColour(1035)
                 :GetLineStyle(1035 3100)
                 :GetLineWidth(1035 3100)
                 :PutBefore(1035 1032)
                 :PutBehind(1035 1032)
                 :SendToBack(1035)
                 :SetBoxSize(1035 90 90)
                 //:SetFillColour(1035)
                 //:SetLineColour(1035)
                 :SetLineStyle(1035 5)
                 :SetLineWidth(1035 6)
                 :SetPosition(1035 400 50)

                //Bitmap:
                 :ScaleBitmap(1032 50 50)
                 :SetTransparency(1032 25)

                //Text:
                 //:GetTextContent(1036)
                 //:GetTextData(1036)
                 //:SetData(1036)
                 //:SetFontRef(1036)

                //Interactible (using EntryField):
                 :GetHighlightStatus(1043 3102)
                 :GetInteractionStatus(1043 3102)
                 :SetHighlightStatus(1043 true)
                 :SetInteractionStatus(1043 true)

                //EntryField:
                 :GetEntryPoint(1043 3101)
                 :GetOverwriteMode(1043 3102)
                 :SetEntryPoint(1043 4)
                 :SetOverwriteMode(1043 false)

                //HyperText:
                 //:GetLastAnchorField(1044)

                //Video:
                 //:Clone(1039)
                 :ScaleVideo(1039 30 30)
                 //:SetData(1039)

                //Slider:
                 :GetPortion(1042 3101)
                 :GetSliderValue(1042 3101)
                 :SetPortion(1042 50)
                 :SetSliderValue(1042 25)
                 :Step(1042 1)

                //Button (using PushButton)
                 :Deselect(1047)
                 :GetInteractionStatus(1047 3102)
                 :Select(1047)
                 :SetInteractionStatus(1047 true)

                //HotSpot:
                 :Select(1046)

                //PushButton:
                 //:GetLabel(1047)
                 :Select(1047)
                 :SetLabel(1047 "new Label")

                //SwitchButton:
                 :Deselect(1048)
                 :GetSelectionStatus(1048 3102)
                 :Select(1048)
                 :SetLabel(1048 "new Label")
                 //:Toggle(1048)

                //Audio:
                 //:Clone(1038)
                 :GetVolume(1038 3100)
                 //:SetData(1038)
                 //:SetVolume(1038 100)

                //TokenGroup:
                 //:CallActionSlot(1029)
```

```
//ListGroup:
 :AddItem(1030 1 1032)
 :DelItem(1030 1032)
 :DeselectItem(1030 2)
 :GetCellItem(1030 2 1032)
 :GetFirstItem(1030 1032)
 //:GetItemStatus(1030)
 //:GetListItem(1030)
 //:GetListSize(1030)
 //:ScrollItems(1030)
 //:SelectItem(1030)
 //:SetFirstItem(1030)
 //:ToggleItem(1030)

//TokenManager (using TokenGroup):
 //:GetTokenPosition(1029)
 //:Move(1029)
 //:MoveTo(1029)

//Variable (using IntegerVariable):
 //:SetVariable(1023)
 //:TestVariable(1023)

//IntegerVariable:
 //:Add(1023)
 //:Divide(1023)
 //:Modulo(1023)
 //:Multiply(1023)
 //:SetVariable(1023)
 //:Subtract(1023)

//BooleanVariable:
 //:SetVariable(1022 )

//OctetStringVariable:
 //:SetVariable(1024)
 //:Append(1024)

//ObjectRefVariable:
 //:SetVariable(1025)

//ContentRefVariable:
 //:SetVariable(1026)
```

```
   )
  }

  // ----------------------------------------------------------


   {:PushButton 9001
    :OrigBoxSize   100  60
    :OrigPosition  540 480
    :ButtonRefColour gray
    :OrigLabel "back to main"
   }

   {:Link 9002
    :EventSource 9001 :EventType IsSelected
    :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
   }

   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                  )
   }


 )

 :InputEventReg 1
 :SceneCS 650 550
 :MovingCursor true
}
```

token.mhg:

```
{:Scene ( "demo/tokentest.mhg" 0 )

 :Items
 (
   {:Text 500 :InitiallyActive false
    :OrigContent  'TokenGroup is running'
    :OrigBoxSize 500 30
    :OrigPosition 70 2
    :FontAttributes Bold.18 :FontName Proportional
   }
   {:Text 501 :InitiallyActive true
    :OrigContent  'TokenGroup is not running'
    :OrigBoxSize 500 30
    :OrigPosition 70 2
    :FontAttributes Bold.18 :FontName Proportional
   }

   {:PushButton 50 :InitiallyActive true
    :OrigBoxSize 150 30 :OrigPosition 10 175
    :ButtonrefColour Gray :OrigLabel "Run TokenGroup" }
   {:Link 60 :EventSource 50 :EventType IsSelected
    :LinkEffect ( :MoveTo( 1000 0 )
                  :CallActionSlot( 1000 1 )
                  :Run( 1000 ) )}

   {:PushButton 51 :InitiallyActive true
    :OrigBoxSize 150 30 :OrigPosition 160 175
    :ButtonRefColour Gray :OrigLabel "Stop TokenGroup" }
   {:Link 62 :EventSource 51 :EventType IsSelected
    :LinkEffect ( :MoveTo( 1000 0 )
                  :CallActionSlot( 1000 2 )
                  :Stop( 1000 ))}

   {:Link 100 :EventSource 101 :EventType IsSelected
    :LinkEffect ( :GetTokenPosition( 1000 104 )
                  :SetVariable( 103 :GInteger :IndirectRef 104 )
                  :SetVariable( 102 :GOctetString "Query TokenPosition<" )
                  :Append     ( 102 :IndirectRef 103 )
                  :Append     ( 102 ">" )
                  :SetLabel   ( 101 :IndirectRef 102 ) )
   }
```

```
{:PushButton 101 :InitiallyActive true
 :OrigBoxSize 300 30 :OrigPosition 10 205
 :ButtonRefColour Gray :OrigLabel "Query TokenPosition<?>" }
{:OStringVar 102 :OrigValue "" }
{:OStringVar 103 :OrigValue "" }
{:IntegerVar 104 :OrigValue -1 }

{:PushButton 106 :InitiallyActive true
 :OrigBoxSize 100 30 :OrigPosition 10 235
 :ButtonRefColour Gray :OrigLabel "NextItem" }
{:PushButton 107 :InitiallyActive true
 :OrigBoxSize 100 30 :OrigPosition 110 235
 :ButtonRefColour Gray :OrigLabel "PrevItem" }

{:Link 111 :EventSource 106 :EventType IsSelected
 :LinkEffect( :Move( 1000 1 ) ) }
{:Link 112 :EventSource 107 :EventType IsSelected
 :LinkEffect( :Move( 1000 2 ) ) }

{:PushButton 120 :InitiallyActive true
 :OrigBoxSize 60 30 :OrigPosition 10 265
 :ButtonRefColour Gray :OrigLabel "MoveTo1" }
{:PushButton 121 :InitiallyActive true
 :OrigBoxSize 60 30 :OrigPosition 70 265
 :ButtonRefColour Gray :OrigLabel "MoveTo2" }
{:PushButton 122 :InitiallyActive true
 :OrigBoxSize 60 30 :OrigPosition 130 265
 :ButtonRefColour Gray :OrigLabel "MoveTo3" }
{:PushButton 123 :InitiallyActive true
 :OrigBoxSize 60 30 :OrigPosition 190 265
 :ButtonRefColour Gray :OrigLabel "MoveTo4" }
{:PushButton 124 :InitiallyActive true
 :OrigBoxSize 60 30 :OrigPosition 250 265
 :ButtonRefColour Gray :OrigLabel "MoveTo5" }

{:Link 125 :EventSource 120 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 1 ) ) }
{:Link 126 :EventSource 121 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 2 ) ) }
{:Link 127 :EventSource 122 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 3 ) ) }
{:Link 128 :EventSource 123 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 4 ) ) }
```

```
{:Link 129 :EventSource 124 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 5 ) ) }

{:Rectangle 4711 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 40 25 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkRed :OrigRefFillColour Red
}
{:Rectangle 4712 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 78 45 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkGray :OrigRefFillColour Gray
}
{:Rectangle 4713 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 116 65 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkBlue :OrigRefFillColour Blue
}
{:Rectangle 4714 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 154 85 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkGreen :OrigRefFillColour Green
}
{:Rectangle 4715 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 192 105 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkMagenta :OrigRefFillColour Magenta
}

{:Link 1001 :EventSource 1000 :EventType TokenMovedFrom :EventData 1
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1002 :EventSource 1000 :EventType TokenMovedTo :EventData 1
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1003 :EventSource 1000 :EventType TokenMovedFrom :EventData 2
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1004 :EventSource 1000 :EventType TokenMovedTo :EventData 2
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1005 :EventSource 1000 :EventType TokenMovedFrom :EventData 3
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1006 :EventSource 1000 :EventType TokenMovedTo :EventData 3
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1007 :EventSource 1000 :EventType TokenMovedFrom :EventData 4
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1008 :EventSource 1000 :EventType TokenMovedTo :EventData 4
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1009 :EventSource 1000 :EventType TokenMovedFrom :EventData 5
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1010 :EventSource 1000 :EventType TokenMovedTo :EventData 5
```

```
  :LinkEffect ( :CallActionSlot( 1000 2 ) )}


 {:TokenGroup 1000
  :InitiallyActive false
  :Shared false
  :MovementTable
    (
      (2 3 4 5 1 )
      (5 1 2 3 4 )
    )
  :TokenGroupItems
    (
      (4711
        :ActionSlots
        (
          (:SetFillColour(4711 :NewAbsoluteColour red))
          (:SetFillColour(4711 :NewAbsoluteColour transparent))
        )
      )
      ( 4712
        :ActionSlots
        (
          (:SetFillColour(4712 :NewAbsoluteColour gray))
          (:SetFillColour(4712 :NewAbsoluteColour transparent))
        )
      )
      ( 4713
        :ActionSlots
        (
          (:SetFillColour(4713 :NewAbsoluteColour blue))
          (:SetFillColour(4713 :NewAbsoluteColour transparent))
        )
      )
      ( 4714
        :ActionSlots
        (
          (:SetFillColour(4714 :NewAbsoluteColour green))
          (:SetFillColour(4714 :NewAbsoluteColour transparent))
        )
      )
      ( 4715
        :ActionSlots
```

```
           (
             (:SetFillColour(4715 :NewAbsoluteColour magenta))
             (:SetFillColour(4715 :NewAbsoluteColour transparent))
           )
         )
       )
     :NoTokenActionSlots
       (
         (:Run( 500 ) :Stop( 501 ) )
         (:Run( 501 ) :Stop( 500 ) )
       )
     }



     {:PushButton 9001
      :OrigBoxSize   100  30
      :OrigPosition  210 300
      :ButtonRefColour gray
      :OrigLabel "back to main"
     }

     {:Link 9002
      :EventSource 9001 :EventType IsSelected
      :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
     }


    {:Link 9003
     :EventSource 9001 :EventType CursorEnter
     :LinkEffect ( :Activate( 9005 ) )
    }
    {:Link 9004
     :EventSource 9001 :EventType CursorLeave
     :LinkEffect ( :DeActivate( 9005 ) )
    }
    {:Link 9005
       :InitiallyActive false
       :EventSource 0
       :EventType UserInput
       :EventData 15
       :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
                   )
     }



  )

  :InputEventReg 1
  :SceneCS 320 350
  :MovingCursor true
}
```

Dave Marshall
10/4/2001

More Examples

Further examples are available in the applications folder:

bitmap
-- further examples of bitmaps in MHEG
interacting
-- further examples of interactiion in MHEG
intvar
-- integer variables
jmf
-- video and audio
quiz2
-- MHEG quiz in MHEG
text
-- further text in MHEG.

Dave Marshall
10/4/2001

Relationships to Major Standards

Important relationships exist between MHEG-5 and other standards and specifications.

Davic (Digital Audio Visual Council)
-- aims to maximize interoperability across applications and services for the broadcast and interactive domains. It selects existing standards and specifies their interfaces in the realm of a complete reference model. This comprises the content provider system, the service provider system, and the service consumer system. This forum has prepared a series of specifications: Davic 1.0 selected MHEG-5 for encoding base level applications and Davic 1.1 relies on MHEG-6 to extend these applications in terms of the Java virtual machine that uses services from the MHEG-5 RTE.

DVB (Digital Video Broadcasting)s
-- provides a complete solution for digital television and data broadcasting across a range of delivery media where audio and video signals are encoded in MPEG-2. The specification includes an open service information system, known as DVB-SI, which provides the elements necessary for developing a basic electronic program guide (EPG) to support navigation amongst the new digital television services.

MPEG
-- the well-known(Chapter 7) family of standards used for coding audiovisual information (such as movies, video, and music) in a digital compressed format. MPEG-1 and MPEG-2 streams are likely to be used by MHEG-5 applications, which can easily control their playback through the facilities provided by the Stream class.

DSMCC (Digital Storage Media Command and Control)
-- specifies a set of protocols for controlling and managing MPEG streams in a client-server environment. The user-to-user protocol (both the client and server are users) consists of VCR commands for playback of streams stored on the server, as well as commands for downloading other data (bitmaps, text, and so on). The playback of MPEG streams from within MHEG-5 applications has a counterpart at the DSMCC layer.

More Examples

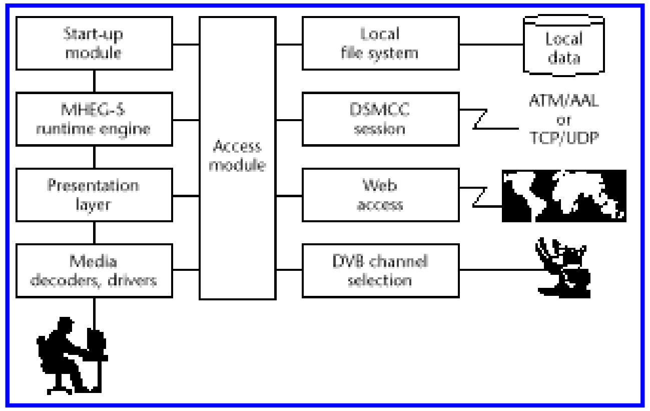
Dave Marshall
10/4/2001

MHEG Implementation

Several components may be requires in implementing and MHEG systems:

Runtime Engine (RTE)

-- MHEG-5 runtime engines generally run accross a client-server architecture (See The Armida system (Figure 8.14) referenced below for an example application).



Armedia Client Architecture
Armedia is a client-server based interactive multimedia application retrieval system.

A preceding ***Start-up*** Module may be used to perform general initialization ***etc.***:

- The client can be launched either as an autonomous Windows application or
- as a plug-in by an HTML browser, allowing seamless navigation between the World Wide Web and the webs of MHEG-5 applications. (See Armida system for more details).

The MHEG-5 RTE is the kernel of the client's architecture. It performs the pure interpretation of MHEG-5 objects and, as a platform-independent module, issues I/O and data access requests to other components that are optimized for the specific runtime platform.

The RTE performs two main tasks. First, it prepares the presentation and handles accessing, decoding, and managing MHEG-5 objects in their internal format. The second task is the actual presentation, which is based on an event loop where events trigger actions. These actions then become requests to the Presentation layer along with other actions that internally affect the engine.

Presentation layer

-- The presentation layer (PL) manages windowing resources, deals with low-level events, and performs decoding and rendering of contents from different media to the user. This functionality is available to the RTE via an object-oriented API that encapsulates all I/O platform specific aspects. The basic MHEG-5 Presentable classes have counterparts at this API level, which makes provisions for initialization/termination, data access and decoding, setting specific attributes (such as text font, color, and so on), and performing spatial and temporal controls. In addition, an informative flow exists from the PL back to the RTE, which notifies user interaction and stream events.

Access module

-- This module provides a consistent API for accessing information from different sources. It's used by the RTE to get objects and the PL to access content data (either downloaded or streamed). Note that the selection of a particular delivery strategy is out of MHEG-5's scope, and hence remains an implementation issue. Typical applications should support:

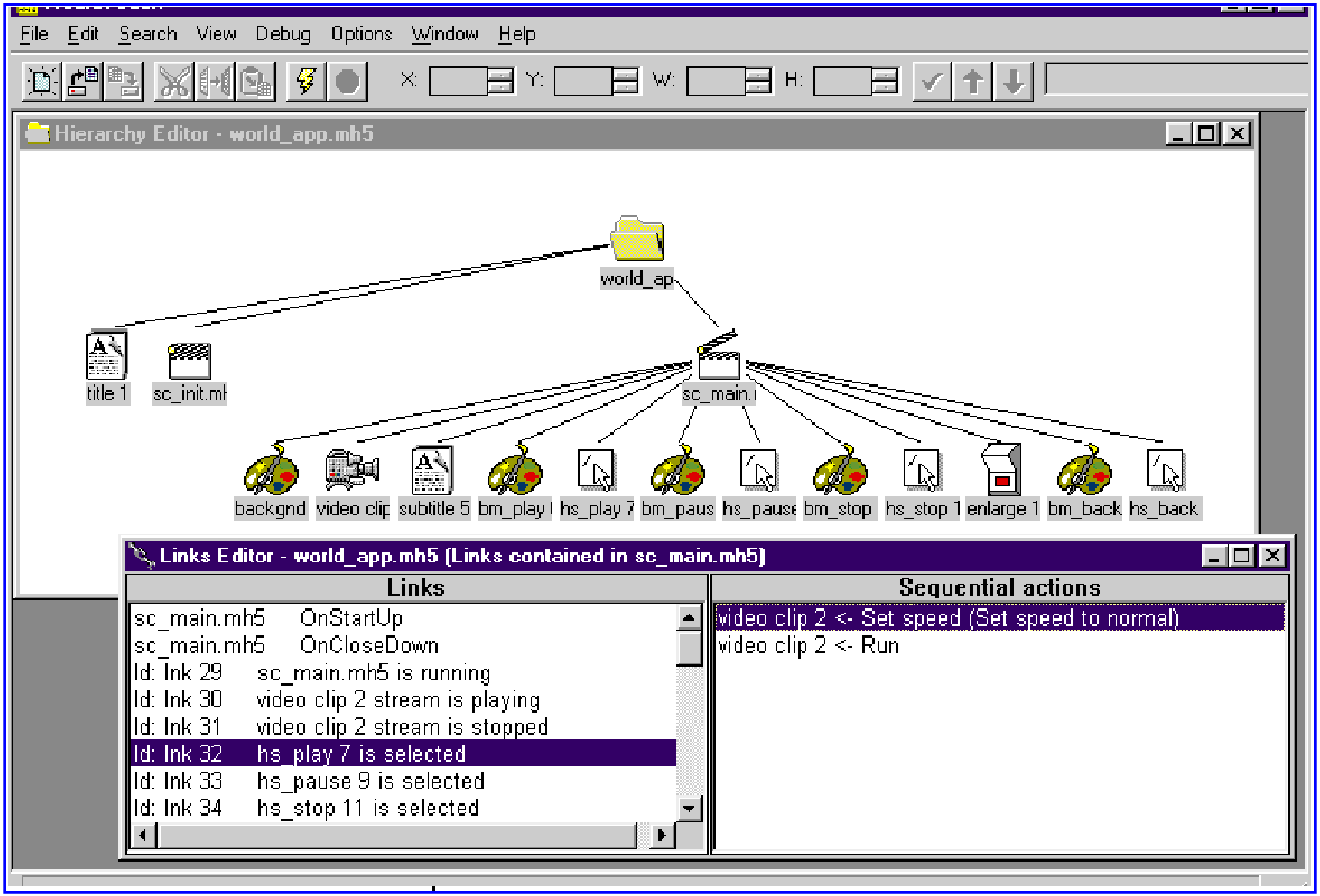
- bulk download for bitmaps, text, and MHEG-5 objects; and
- progressive download for audio and audiovisual streams.

The implementation of these mechanisms occurs via the DSMCC interface. The user has full interactive control of the data presentation, including playback of higher quality MPEG-2 streams delivered through an ATM network. Object and content access requests can also be issued to the Web via HTTP. However, this wide-spread protocol still suffers from limitations when high-quality video clips are progressively down-loaded, since the underlying networks may not yet provide adequate quality-of-service (QoS). When accessing the broadcast service, the Access module requires the DVB channel selection component to select the program referred to by a Stream object. To achieve this, you set the Con-tentHook to a value that means ***Switched DVB*** and the OriginalContent to the identifier of the channel to select. The MPEG-2 stream transport-ed by the selected channel is displayed in the active Scene's presentation space.

MHEG Authoring Tool: MediaTouch

--

The availability of an adequate authoring tool is mandatory to create the MHEG applications. The MediaTouch (Figure 8.15) application is one example developed for the Armida Sysytem
(http://drogo.cselt.stet.it/ufv /ArmidaIS/home_en.htm). It is a visual-based Hierarchical Iconic authoring tool, similar to Authorware in many approaches.



```
sc_main.mh5    OnStartUp
sc_main.mh5    OnCloseDown
Id: lnk 29     sc_main.mh5 is running
Id: lnk 30     video clip 2 stream is playing
Id: lnk 31     video clip 2 stream is stopped
Id: lnk 32     hs_play 7 is selected
Id: lnk 33     hs_pause 9 is selected
Id: lnk 34     hs_stop 11 is selected
```

```
video clip 2 <- Set speed (Set speed to normal)
video clip 2 <- Run
```

MediaTouch MHEG Authoring Tool (Hierarchy and Links Editor windows)

MediaTouch is based on the native approach, which lets the author operate at the level of MHEG-5 objects.

MediaTouch provides authors with the following functions:

Hierarchy Editor

-- Supports the interactive creation of the structure of Applications and Scenes objects. The author operates by adding, moving, and deleting Ingredients on the tree that represents the hierarchy's current status. Several Scenes from different Applications can be edited at one time and you can copy and move Ingredients between Scenes and Applications.

Properties Editor

-- The author sets the properties of Presentable Ingredients via an input form that is specific to each object class. The author does not need to type a value directly, since the system adopts a menu-based input interface that aims to minimize errors and inconsistencies.

Layout Editor

-- A Scene's layout can be set by interactively adjusting the position and size of the bounding box for every Visible Ingredient. An Ingredient's content data, when available, is also displayed within its bounding box to make the layout match the actual presentation.

Links Editor

-- This function lets authors add, modify, and delete links and actions to Application and Scene objects. The author sets the link conditions, actions, and referenced objects via a menu. Links and actions can also be copied between different scenes and applications.

Figure 8.15) shows a screen shot from MediaTouch where an Application composed of two Scenes and one shared Text is open within the Hierarchy Editor window. One of these Scenes shows its Presentable Ingredients, and the author is editing its Links. The author can then launch the RTE from within MediaTouch to check the outcome of his work.

MHEG Autoring Tool: MHEGDitor

-- MHEGDitor is an MHEG-5 authoring tool based on Macromedia Director. MHEGDitor is composed of an Authoring Xtra to edit applications and of a Converter Xtra to convert resulting movies into MHEG-5 applications. The two MHEGDitor Xtras work separately. With MHEGDitor Authoring Xtra, create and edit your application within Macromedia Director 6. It opens a window to set preferences and links a specific external script castLib to your movie for you to create specific MHEG behaviours rapidly. You can test your application on the spot within Director, as if it were played by MHEGPlayer, the MHEG interpreter companion of MHEGDitor. With MHEGDitor Converter Xtra, convert Macromedia Director 6 movies (edited with MHEGDitor Authoring Xtra) into a folder containing all necessary items for an MHEG-5 application. Associated with Macromedia Director 6, MHEGDitor is the easiest way to author multimedia applications for Interactive TV for both Macintosh and Windows. This tool makes it possible for all system operators or content providers to deliver developed interactive programs to the widest range of potential users.

The Current Release is 1.4 which performs the following:

- Director 6.5 to MHEG-5
- Available on both Macintosh and Windows 95/NT platforms
- Advanced dynamic lists support, new Converter Xtra settings (text data exported as external data...), converted applications optimisation, and more.

MHEG writing tool:MHEG Write

-- An editor to create and manipulate MHEG-5 applications by hand This editor is based on the free software "VIM", which is available via Internet from various sites for virtually all operating systems, including DOS, Windows, UNIX, etc. The MHEGWrite extension supports only the MHEG-5 textual notation. MHEGWrite provides macros for object templates, syntax highlighting and syntax error detection. Using MHEGWrite enables you to access every MHEG-5 features to edit your applications, since this editor is able to read native MHEG-5 code. It also gives good support for detailed examination, profile adaptation, and error correction of MHEG-5 applications produced with arbitrary other tools. By purchasing MHEGWrite, the MHEG Centre will also provide you with a set of MHEG-5 application samples, illustrating several MHEG-5 features, such as ListGroup together with cloning, using variables, sharing objects, etc.

Playing MHEG files

-- There are a few ways to ***play*** MHEG files:

- **MHEGPlayer** is an MHEG-5 interpreter which is able to execute MHEG-5 applications developed with MHEGDitorTM or any other authoring tool. It implements the MHEG-5 International Standard according to the additional requirements of the DAVIC application domain. The available content types are aligned with the underlying operating system. MHEGPlayer consists of a core interpreter with presentation system for Windows 95/NT. It is pre-configured to utilise the local file system to access MHEG objects and content data.

  Supported Formats include:

  - Bitmap: PNG, BMP, TIFF, JPEG, PSD
  - Video: MPEG-1, AVI, Quicktime (via MCI)
  - Audio: Wave, AIFF_C (via MCI)
  - MHEG-5: ASN.1 (DER) or Textual Notation
- MHEG Java Engine -- Java Source code exists to compile a platform-independent MHEG player (***http://enterprise.prz.tu-berlin.de/imw/***)
- MHEG plug-ins for Netscape Browsers and Internet Explorer have been developed,

---

Dave Marshall
10/4/2001

MHEG Future

Several companies and research institutes are currently developing MHEG tools and applications and conducting interoperability experiments for international projects and consortia. The MHEG Support Center is a European project that hopes to implement and operate an MHEG support and conformance testing environment for developers of multimedia systems and applications. Its partners include CCETT, Cril Ingenierie (France), De teBerkom, and GMD Fokus (Germany). IPA, the Information-technology Promotion Agency of Japan, is developing an MHEG-5 system with an authoring tool and viewer. The project members consist of ASCII, KDD, NEC, NTT Soft-ware, the Oki Electric Industry, Pioneer, Sony, and Toppan Printing. The purpose of this project is to do a profiling study and conduct interoperability testing of MHEG-5 systems.

Also, the following European achievements are worth mentioning:

- a Java implementation of an MHEG-5 engine, available from Philips; (See below)
- MhegDitor by CCETT, an authoring tool based on Macromedia Director; and
- the results of the Jupiter project that addresses usability, performance, and interoperability of Davic-compliant services.

At CSELT, a partner of Jupiter, we're making more complete versions of MediaTouch and Armi-da. We've demonstrated these systems at a number of events on behalf of Telecom Italia and showed such applications as movies-on-demand, news-on-demand, tourist information, and interactive TV.

A list containing the above applications and other initiatives is maintained by the MHEG Users Group (MUG), an unofficial forum begun by people from the standardization body to dissemi-nate information and enable discussion on MHEG (see http://www.fokus.gmd.de/ovma/mug).

It's evident that the initial interest in MHEG-1 and MHEG-3 has recently decreased due to the incoming ITV/VoD profile, which is based on MHEG-5 and MHEG-6. The specification of this ***complete solution***, urged and endorsed by Davic, was finalized by April 1998 when MHEG-6 is scheduled to become an International Standard. Further efforts will be devoted to MHEG-7, which is expected to provide the final specification of conformance and interoperability aspects for MHEG-5 by January 1999.

Dave Marshall
10/4/2001

Further Reading/Information

The above notes on MHEG are base largely on an article ***MHEG-5 -- Aim, Concepts, and Implementation Issues***, by M. Echiffre, C. Marchisio, P. Marchisio, P. Panicciari and S. Del Rossi, IEEE Multimedia, Winter 1998, pp 84-91. Also available at URL: http://www.fokus.gmd.de/ovma/mug/archives/doc/u1084.pdf

Other good reference on MHEG and related multimedia standards are:

- T. Meyer-Boudnik and W. Effelsberg, ***MHEG Explained***, IEEE Multimedia, Spring 1995, Vol. 2, No. 1, pp. 26-38.
- ISO 13522-5, Information Technology‹Coding of Multimedia and Hypermedia Information, Part 5: Support for Base-Level Interactive Applications, Int'l Org. for Standardization/Int'l Electronics Comm., Geneva, 1996.
- ISO Draft International Standard 13522-6, Information Technology‹Coding of Multimedia and Hypermedia Information, Part 6: Support for Enhanced Interactive Applications, Int'l Org. for Standardization/ Int'l Electronics Comm., Geneva, 1997.
- Davic 1.0 Specifications, Revision 5.0, Imprimeries de Versoix, Versoix, Switzerland, Dec. 1995.
- ETSI 300 468 2d ed.: Digital Broadcasting Systems for Television, Sound, and Data Services, Specification for Service Information (SI) in Digital Video Broadcasting (DVB) Systems, ETSI publications dept., Sophia Antipolis, France, 1997, http://www.etsi.fr.
- ISO 13818-6, Information Technology‹Generic Coding of Moving Pictures and Associated Audio Informa-tion, Part 6: Digital Storage Media Command and Control, Int'l Org. for Standardization/Int'l Electronics Comm., Geneva, 1996.
- S. Dal Lago et al., ***Armida: Multimedia Applications Across ATM-Based Networks Accessed via Internet Navigation***, Multimedia Tools and Applications, Vol. 5, No. 2, Sept. 1997.

Resources on the Web include:

- **The MHEG Centre** --- http://www.mhegcentre.com/
- **MHEG-5 User Group** --- www.fokus.gmd.de/ovma/mug
- **MHEG and the WWW** --- http://www.prz.tu-berlin.de/~joe/mheg/mheg_intro.html
- **Java MHEG Engine** --- http://thunder.informatik.uni-

kl.de:8080/MHEG5Engine/ , http://enterprise.prz.tu-berlin.de/imw/ (JAVA ENGINE SOURCE)

- **MediaTouch MHEG authoring ---** http://drogo.cselt.stet.it/ufv/mediatouch/index.htm
- **MediaTouch Free RTE ---** http://drogo.cselt.stet.it/ufv/mediatouch/rte.htm
- **Moving Pictures Expert Group ---** www.cselt.it/mpeg, www.mpeg.org
- **Digital Audio Visual Council ---** www.davic.org
- **Digital Video Broadcasting ---** www.dvb.org
- **CSELT, Multimedia and Video Services ---** www.cselt.it/ufv
- **Example MHEG Applications ---** http://drogo.cselt.stet.it/ufv/mediatouch/applications.htm

---

Dave Marshall
10/4/2001

Multimedia Programming

Dave Marshall
10/4/2001

QuickTime for Java

To conclude this course we will look at lower level Multimedia programming issues.

Multimedia Programming is NOT Multimedia Authoring

We will look at one example:

- Quicktime for Java

Dave Marshall
10/4/2001

The Java for Quicktime SDK, Example Programs and Online Documentation

The examples in this section are taken from the Java for Quicktime SDK. We will study a few of these programs in detail, some others are referred to with excerpts of code given here. At the end of this Chapter a full list of the example programs are given. You are encouraged to explore these further.

The Java for Quicktime SDK is available from Apple at

- www.apple.com/quicktime/qtjava

Further Documentation and examples are availble from this URL also.

The example programs that come with the SDK are also available locally:

- Java for Quicktime SDK Demos (Individual fil*** es) --- Local Students ONLY
- Zipped Java for Quicktime SDK Demos -- - Local Students ONLY

Finally, JavaDoc information about the Java for Quicktime class library is available from:

- www.cs.cf.ac.uk/applications/javaqt/

Dave Marshall
10/4/2001

Using QuickTime in Multimedia Production

As we have already discussed with our studies on the Quicktime architecture and format:

- QuickTime provides some powerful constructions that can simplify the production of applications or applets that contain or require multimedia content.
- A multimedia presentation can be very complex.
- The Quicktime architecture and its API (C or Java) can help to simplify this process.

Multimedia presentation often includes many different facets:

- some animation, some video, and
- a desire for complex presentation and interactive delivery of this content.
- The spaces and controllers architecture, and
- the services that it provides to build animations, provides the ability to deliver presentations that can be both complex in their construction and powerful in the interactive capabilities that they provide to the user.

The media requirements for such presentations can also be complex.

They may include ``standard'' digital video, animated characters, and customized musical instruments. QuickTime's ability to reference movies that exist on local and remote servers provides a great deal of flexibility in the delivery of digital content.

A movie can also be used to contain the media for animated characters and/or customized musical instruments. For example, a cell-based sprite animation can be built where the images that make up the character are retrieved from a movie that is built specifically for that purpose. In another scenario, a movie can be constructed that contains both custom instruments and a description of instruments to be used from QuickTime's built-in Software Synthesizer to play a tune.

In both cases we see a QuickTime movie used to contain media and transport this media around. Your application then uses this media to recreate its presentation. The movie in these cases is not meant to be played but is used solely as a media container. This movie can be stored locally or remotely and retrieved by the application when it is actually viewed. Of course, the same technique can be applied to any of the media types that QuickTime supports. The sprite images and custom instruments are only two possible applications of this technique.

A further interesting use of QuickTime in this production space is the ability of a QuickTime movie to contain the media data that it presents as well as to hold a reference to external media data. For example, this enables both an artist to be working on the images for an animated character and a programmer to be building the animation using these same images. This can save time, as the production house does not need to keep importing the character images, building intermediate data containers, and so on. As the artist enhances the characters, the programmer can immediately see these in his or her animation, because the animation references the same images.

Next Up Previous

Next: The QuickTime for Java **Up:** QuickTime for Java **Previous:** The Java for Quicktime

Dave Marshall
10/4/2001

The QuickTime for Java API

If you're a Java or QuickTime programmer and want to harness the power of QuickTime's multimedia engine, you'll find a number of important advantages to using the QuickTime for Java API. A C Quicktime API also exists. But we focus only on the Java API in this course.

For one thing, the API lets you access QuickTime's native runtime libraries and, additionally, provides you with a feature rich Application Framework that enables you to integrate QuickTime capabilities into Java software.

Aside from representation in Java, QuickTime for Java also provides a set of packages that forms the basis for the Application Framework found in the quicktime.app group. The focus of these packages is to present different kinds of media. The framework uses the interfaces in the `quicktime.app` packages to abstract and express common functionality that exists between different QuickTime objects.

As such, the services that the QuickTime for Java Application Framework renders to the developer can be viewed as belonging to the following categories:

- creation of objects that present different forms of media, using `QTFactory.makeDrawable()` methods
- various utilities (classes and methods) for dealing with single images as well as groups of related images
- spaces and controllers architecture, which enables you to deal with complex data generation or presentation requirements
- composition services that allow the complex layering and blending of different image sources
- timing services that enable you to schedule and control time-related activities
- video and audio media capture from external sources
- exposure of the QuickTime visual effects architecture

All of these are built on top of the services that QuickTime provides. They provided interfaces and classes in the `quicktime.app` packages can be used as a basis for developers to build on and extend in new ways, not just as a set of utility classes you can use.

Dave Marshall
10/4/2001

An Overview of QuickTime for Java

QuickTime for Java enables Java and QuickTime programmers to have direct access to QuickTime's multimedia capabilities.

- At its simplest level, QuickTime for Java lets you write a Java applet and run that applet on a variety of platforms. Java can be used in an applet, for example, to make Web pages more interactive so that users will be able to interact with those pages in ways they haven't before.
- At an advanced level, you can write applications that composite images, capture music and audio, create special effects, and use sprites for animation.

Dave Marshall
10/4/2001

The Quicktime API: A Set of Java Classes with 2 Layers

The QuickTime API is implemented as a set of Java classes in QuickTime for Java: These classes offer equivalent APIs for using QuickTime functionality on both Mac OS and Windows platforms.

The QuickTime for Java API consists of two layers:

Core layer
: -- provides the ability to access the QuickTime native runtime libraries (its API) and

Application Framework layer
: -- integrate QuickTime capabilities into their Java software.

The Application Framework layer includes:

- the integration of QuickTime with the Java runtime, which includes sharing display space between Java and QuickTime and passing events from Java into QuickTime
- a set of classes that provides a number of services that simplify the authoring of QuickTime content and operation

The QuickTime for Java classes are grouped into a set of packages on the basis of common functionality, common usage, and their organization in the standard QuickTime header files. The packages provide both an object model for the QuickTime API and a logical translation or binding of the native function calls into Java method calls.

Dave Marshall
10/4/2001

Next Up Previous
Next: The QuickTime for Java **Up:** The Quicktime API: A **Previous:** The Quicktime API: A

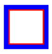
The `QTSimpleApplet` Example

A good starting point for understanding QuickTime for Java is the `QTSimpleApplet` program from the QuickTime for Java SDK. The code listing for `QTSimpleApplet.java` is given below:

```
/*
 * QuickTime for Java SDK Sample Code

   Usage subject to restrictions in SDK License Agreement
 * Copyright: © 1996-1999 Apple Computer, Inc.

 */
import java.applet.Applet;
import java.awt.*;
import java.io.*;

import quicktime.*;
import quicktime.io.QTFile;

import quicktime.app.QTFactory;
import quicktime.app.display.*;
import quicktime.app.image.ImageDrawer; // for exceptions

public class QTSimpleApplet extends Applet {
        private Drawable myQTContent;
        private QTCanvas myQTCanvas;

        public void init () {
                try {
                        // this is a workaround required by a bug in the
loading
                        // mechanism of applets using the JavaPlugin on
Netscape on Win
                        if (QTSession.isInitialized() == false)
                                QTSession.open();

                                // set up a QTCanvas which will disply its
content
                                // at its original size of smaller and
centered in the space given
                                // to the QTCanvas when the applet is
layed out
                        setLayout (new BorderLayout());
                        myQTCanvas = new QTCanvas (QTCanvas.kInitialSize,
0.5F, 0.5F);
                        add (myQTCanvas, "Center");

                        QTFile file = new QTFile (getCodeBase().getFile()
+ getParameter("file"));
                        myQTContent = QTFactory.makeDrawable (file);
                } catch (QTException qtE) {
                                // something wrong with the content but QT
itself is OK
```

```
                        if (QTSession.isInitialized())
                                myQTContent = ImageDrawer.getQTLogo();
                        else
                                throw new RuntimeException
(qtE.getMessage());
                } catch (FileNotFoundException fnfE) {
                        myQTContent = ImageDrawer.getQTLogo();

                } catch (IOException ioE) {
                                // the IOException would occur as a
violation of
                                // java security settings on the user
machine - throw it again
                        throw new SecurityException (ioE.getMessage());
                }
        }

        public void start () {
                try { // if QT was not successfully initialized the
QTCanvas will be null
                        if (myQTCanvas != null)
                                myQTCanvas.setClient (myQTContent, true);
                } catch (QTException e) {
                        e.printStackTrace();
                }
        }

        public void stop () {
                if (myQTCanvas != null)
                        myQTCanvas.removeClient();
        }

        public void destroy () {
                QTSession.close();
        }
}
```

The out of `QTSimpleApplet` running in an applet viewer is shown in Fig. 9.1.



{\tt QTSimpleApplet} Output

The things to note about this program are:

- We can display multimedia content (On Internet or locally) in less than a dozen lines of Java code.
- The media may include video, audio, text, timecode, music/ MIDI, sprite animation, tween, MPEG, or even movies that let you use Apple's QuickTime VR and QuickDraw 3D capabilities.
- You can control with the standard QuickTime movie controller buttons, uses the media as specified in the applet tag -- in this example, `sample.mov`:

  ```
  <applet code="QTSimpleApplet.class" width=200 height=100>
                  <param name="file" value="sample.mov">
          </applet>
  ```

The basic operation of the code is as follows:

- The `QTCanvas`, `QTDrawable`, and `QTFactory` classes that belong to the `quicktime.app` package and are discussed in greater detail. These are typically used in all Quicktime presentations.
- `QTCanvas`, `QTDrawable` create objects where media can be presented:
    - `QTCanvas` is the object responsible for handling the integration from the `java.awt` side between Java and QuickTime.
    - The `QTCanvas` also has parameters that let you control the resizing of the client that it presents -- here we tell the canvas to center the client within the space given by the applet's layout manager.This ensures that the client is only as big as its initial size (or smaller if you make the canvas smaller).
    - The `QTSession.open()` call performs a gestalt check to make sure that QuickTime is present and is initialized. Note that this is a required call before any QuickTime for Java classes can be used.
    - To set up a `QTCanvas` object that displays its content at its original size or smaller and is centered in the space given to the `QTCanvas` when the applet is laid out, we do the following:

```
setLayout (new BorderLayout());
myQTCanvas =
  new  QTCanvas (QTCanvas.kInitialSize, 0.5F,
                 0.5F);
add (myQTCanvas, "Center");
QTFile file = new QTFile (getCodeBase().getFile() +
getParameter("file"));
myQTContent = QTFactory.makeDrawable (file);
} catch (Exception e) {
e.printStackTrace();
...
}
}
```

- `QTFactory()` methods are used to manufacture a wide variety of multimedia objects that QuickTime supports.
    - The `QTFactory.makeDrawable()` method is used to create an appropriate QuickTime object for the media that is specified in the `<PARAM>` tag in the call from the HTML file.
    - If a `QTException` is thrown in the `init()` method, an appropriate action should be taken by the applet, depending on the error reported. In the start() method shown in the next code snippet, you set the client of the `QTCanvas`. This `QTCanvas`.client is the QuickTime object (***i.e.***, an object that implements the `QTDrawable` interface) that draws to the area of the screen that the `QTCanvas` occupies. This is the QuickTime side of the integration between Java and QuickTime:

```
public void start () {
try { my QTCanvas.setClient (myQTContent, true);
} catch (Exception e) {
e.printStackTrace();
}
}
```

    - You use the `stop()` method to remove the client from the `QTCanvas`. It will be reset in the `start()` method if the applet is restarted. `destroy()` is used to close the `QTSession`. This protocol enables the applet to be reloaded, suspended, and resumed -- for example, if the user is leaving and returning to the page with the applet. The `init()/de-stroy()`, and `start()/stop()` methods are reciprocal in their activities.

```
public void stop () {
my QTCanvas.removeClient();
}
public void destroy () {
QTSession.close();
}
```

   - You need to call `QTSession.close()` if you have previously called `QTSession.open()` in order to shut down QuickTime properly.
- The `init()` method may throw exceptions because the required file was not found or the applet does not have permission from Java's security manager to read that file. Alternatively, the required version of QuickTime may not be installed. The applet should deal with these issues appropriately.

Dave Marshall
10/4/2001

The QuickTime for Java Core Packages

As one would expect,The QuickTime for Java classes are grouped into a set of packages. The grouping is based on common functionality and usage. A number of packages also have subpackages that group together smaller sets of functionality.

The major packages generally have a constants interface that presents all of the constants that relate to this general grouping and an exception class that all errors that derive from a call in this package group will throw. The packages, with descriptions of their principal classes and interfaces are now summarised:

Package	Principal classes	Description
	and interfaces	
quicktime	`QTSession`,	The `QTSession` class has calls that set up and
	`QTException`	initialise the QuickTime engine, such as
		initialize, gestalt, and enter Movies.
quicktime.io	OpenFile, `QTFile`,	Contains calls that deal with file I/O. These
	OpenMovieFile,	calls are derived from the Movies.h file.
	QTIOException	
quicktime.qd	QDGraphics,	Contains classes that represent the
	PixMap, Region,	QuickDraw data structures that are required
	QDRect, QDColor,	for the rest of the QuickTime API. These calls
	QDConstants,	are derived from the QuickDraw.h and
	QDException	QDOffscreen.h files. The QuickTime API
		expects data structures that belong to
		QuickDraw, such as graphics ports, GWorlds,
		rectangles, and points.
quicktime.qd3d	CameraData,	Contains classes that represent the
	Q3Point, Q3Vector	QuickDraw 3D data structures that are
		required for the rest of the QuickTime API,

		predominantly the tweener and 3D media
		services.
quicktime.sound	SndChannel, Sound,	Contains classes that represent the Sound
	SPBDevice,	Manager API. These calls are derived from
	SoundConstants,	the Sound.h file. While some basic sound
	SoundException,	recording services are provided, for more
		demanding sound input and output the
		sequence grabber components and movie
		playback services should be used.

Package	Principal classes	Description
	and interfaces	
quicktime.std	StdQTConstants,	The original QuickTime interfaces on the
	StdQTException	Mac OS are contained in a collection of eight
		header files that describe the standard
		QuickTime API. As such, nearly all of the
		functions defined in these files are to be
		found in classes in the quicktime.std group
		of packages.
quicktime.std.anim	Sprite,	Classes that provide support for animation.
	`SpriteWorld`	QuickTime can be used as a real time
		rendering system for animation, distinct from
		a data format-that is, the movie. Thus, you
		can create a graphics space (`SpriteWorld`)
		within which characters (Sprite objects) can
		be manipulated.
quicktime.std.clocks	Clock,	Contains classes that provide timing services,
	TimeBase,	including support for the creation of
	QTCallback	hierarchical dependencies between time
	and subclasses	hierarchical dependencies between time
		bases, the usage of callbacks for user
		scheduling of events or notification, and the

		capability of instantiating the system clocks
		that provide the timing services.
quicktime.std.com	Component,	QuickTime is a component-based
	Component-Description	architecture, with much of its functionality
		being provided through the creation and
		implementation of a particular component's
		API. This package contains classes that
		provide basic support for this component
		architecture; a full implementation is
		forthcoming.
quicktime.std.image	CodecComponent,	Contains classes that present the Image
	QTImage,	Compression Manager. These classes provide
	CSequence, Matrix	control for the compression and
		decompression of both single images and
		sequences of images. It also contains the
		Matrix class, which (like the Region class in
		the qd package) is used generally throughout
		QuickTime to alter and control the rendering
		of 2D images.

Package	Principal classes	Description
	and interfaces	
quicktime.std.movies	AtomContainer,	Contains the principal data structures of
	Movie,	QuickTime, including classes that represent
	`MovieController`,	QuickTime atom containers, movies, movie
	Track	controllers, and tracks-all essential for
		creating and manipulating QuickTime
		movies. A movie containing one or more
		tracks is the primary way that data is
		organized and managed in QuickTime. A
		Movie object can be created from a file or from
		memory and can be saved to a file. The

		`MovieController` class provides the standard
		way that QuickTime data (movies) are
		presented and controlled. AtomContainer
		objects are the standard data structures used
		to store and retrieve data in QuickTime.
quicktime.std.movies.media	DataRef, Media	A Track object is a media neutral structure,
	and subclasses,	but it contains a single Media type that
	MediaHandler	defines the kind of data that a Track is
	and subclasses,	representing. The Media, MediaHandler and
	Sample	SampleDescription subclasses describe the
	Description and	various media types that QuickTime can
	subclasses	present. Media classes control references to
		data that comprise the raw media data.

Package	Principal classes	Description
	and interfaces	
quicktime.std.music	AtomicInstrument,	Contains classes that deal with the general
	NoteChannel,	music architecture provided by QuickTime.
	NoteAllocator	This architecture can be used to capture and
		generate music (MIDI) events in real time,
		customize and create instruments, and
		eventually provide your own algorithmic
		synthesis engines.
quicktime.std.qtcomponets	MovieExporter,	Contains classes that interface with some of
	MovieImporter,	the components that are provided to supply
	TimeCoder	different services. The import and export
		components are supported, as are tween and
		timecode media components.
quicktime.std.sg	SequenceGrabber,	Contains classes that implement the
	SGVideoChannel,	sequence grabber component for capturing
	SGSoundChannel	video and audio media data.
quicktime.util	QTHandle,	Contains classes that represent utility

	QTByteObject,	functionality required by the general
	QTPointer,	QuickTime API. The most commonly used
	UtilException	feature of this package is a set of classes for
		memory management from Memory.h. These
		classes typically form the base class for actual
		QuickTime objects.
quicktime.vr	QTVRConstants,	Contains classes that represent the
	QTVRInstance,	QuickTime Virtual Reality API. The package
	QTVRException	contains all the QuickTime VR interface
		constants, the QTVRInstance class and some
		QTVR callbacks for presentation of QTVR
		content.

Next Up Previous

Next: The Quicktime for Java **Up:** QuickTime for Java **Previous:** The QTSimpleApplet Example

Dave Marshall
10/4/2001

The Quicktime for Java Application Framework Packages

The classes in the QuickTime for Java Application Framework are built entirely on top of QuickTime for Java native binding classes. The Application Framework classes are designed to simplify the usage of the QuickTime for Java API and to provide a close integration with Java's display and event distribution system. They offer a set of services that are commonly used by QuickTime programs. In addition, they provide useful abstractions and capabilities that make the use of these services simpler and easier for the developer. The Framework itself is also designed with reusability and extensibility of classes in mind. It uses Java interfaces to express some of the functionality that can be shared or is common among different classes. You can also implement your own versions of these interfaces, or extend existing implementations, to more specifically meet a particular requirement, and in so doing, use these custom classes with other classes of the Framework itself. The Following Table describes the various Framework packages and their principal classes.

Package	Description
`quicktime.app`	Provides a set of ``factory" methods for creating classes that
	you can use to present media that QuickTime can import. In
	addition, it provides some utility methods for finding
	directories and files in the local file system.
`quicktime.app.action`	Contains a large number of useful controller classes for mouse
	drags and for handling mouse events. It also contains action
	classes that can be used to apply actions to target objects.
`quicktime.app.anim`	Contains classes that present all of the functionality of the
	Sprite and `SpriteWorld`.
`quicktime.app.audio`	Contains a number of interfaces and classes that deal
	specifically with the audio capabilities of QuickTime.
`quicktime.app.display`	Contains a number of classes that are important for using the
	QuickTime for Java API. `QTCanvas` and `QTDrawable` negotiate
	with java.awt classes to allow the presentation of QuickTime
	content within a Java window or display space.
`quicktime.app.image`	Handles the presentation and manipulation of images.
	Included are utility classes for setting transparent colors in
	images, applying visual effects, creating objects for handling

	sequences of images, and `QTDrawable` objects that read image
	data from a file or load the data into memory.
`quicktime.app.players`	`QTPlayer` and `MoviePlayer` define a set of useful methods that
	enables you to present QuickTime movies, using `QTCanvas`
	objects and the `QTDrawable` interface.
`quicktime.app.sg`	Contains a single class, `SGDrawer`.
`quicktime.app.spaces`	Interfaces in this package provide a uniform means of dealing
	with a collection of objects in QuickTime for Java.
`quicktime.app`.time	Provides a set of useful classes to handle timing services used
	to schedule regular tasks that need to be performed on an
	ongoing basis.

Next Up Previous

Next: Quicktime for Java Classes **Up:** QuickTime for Java **Previous:** The QuickTime for Java

Dave Marshall
10/4/2001

Quicktime for Java Classes and Interfaces

This section discusses the usage of two principal classes and one principal interface in the QuickTime for Java API.

These are

`QTCanvas`
: As we saw in the our `QTSimpleApplet`, the `QTCanvas` is a subclass of the `java.awt.Canvas` and represents primarily a way to gain access to the underlying native graphics structure of the platform. QuickTime requires this in order to draw to the screen.

`QTDrawable`
:

an interface that expresses this requirement of QuickTime objects to draw to this native graphics structure.

`QTFactory`
: a class that uses the importing capabilities of QuickTime to ``manufacture'' `QTDrawable` objects to present that media.

In this section, we discuss how to

- use the `QTCanvas` class to interact with the Java display and event system
- use the `QTDrawable` interface to encapsulate common operations that can be applied to a QuickTime drawing object
- present media that QuickTime can import by using ``factory'' methods provided by the `QTFactory` class

The section also introduces briefly an abstraction known as a space, which is part of the QuickTime for Java spaces and controllers architecture. A space defines and organizes the behavior of objects and allows for the complex representation of disparate media types.

The QTCanvas Class

To present QuickTime content within Java, you need a mechanism for interacting with the Java display and event system. This is provided through the services of the `QTCanvas` class and its client, an object that implements the Drawable interface. `QTCanvas` is a specialized canvas that supplies access to the native graphics environment and offers expanded display functionality.

`QTCanvas` encapsulates much of the display and presentation functionality of the QuickTime for Java API. It is responsible for ``punching a hole" within the Java display surface and telling its client that it can draw to this display surface and receive events that occur therein. Its clients are generally some kind of QuickTime object, and the events it receives may be mouse or key events. An instance of a `QTCanvas` object can display any object that implements the Drawable interface.

The client of a `QTCanvas` object is called Drawable because QuickTime generally uses the word ``draw" -- for example, `MCDraw`. Java uses the word ``paint," so this enables us to make a distinction between Java objects that ``paint" and QuickTime objects that ``draw."

Your code can interact with the `QTCanvas` methods as with those of any java.awt.Component. The `QTCanvas` delegates calls as appropriate to its drawing client. A client essentially draws itself in the display area of a `QTCanvas`. If the client is a `QTDrawable`, the `QTCanvas` also gets the graphics structure (QDGraphics) of the native implementation of the canvas's peer and sets this QDGraphics as the destination QDGraphics of such a client. All `QTDrawable` objects require a destination QDGraphics in which to draw.

Using the `setClient()` method, you can associate a new client, a Drawable object, with this `QTCanvas`. The flag determines if awt performs a layout and how the client is integrated with the canvas. If the flag is false, the new client takes on the current size and position of the canvas. If the flag is true, then awt lays out the canvas again, using the initial size of the client and the resize flags to resize the canvas and its client. The `getClient()` method returns the Drawable object currently associated with this `QTCanvas`.

- Interacting With Java Layout Managers

Next Up Previous

Next: Interacting With Java Layout **Up:** Quicktime for Java Classes **Previous:** Quicktime for Java Classes

Dave Marshall
10/4/2001

Interacting With Java Layout Managers

QuickTime media content is often sensitive to the size and proportions of screen space that it uses -- for instance, a movie that is made to display at 320 x 240 pixels can look bad or have serious performance repercussions if drawn at 600 x 60 pixels on the screen.

The `QTCanvas` provides flags for controlling how the Java layout managers allocate space to it (`resizeFlags`) and where, within that space, the `QTCanvas` client actually draws (`alignmentFlags`). You can also set the minimum, preferred, and maximum sizes of the `QTCanvas`, as with any other java.awt.Component. The preferred size is automatically set for you as the initial size of the `QTCanvas` client if your application does not specifically set the preferred size itself.

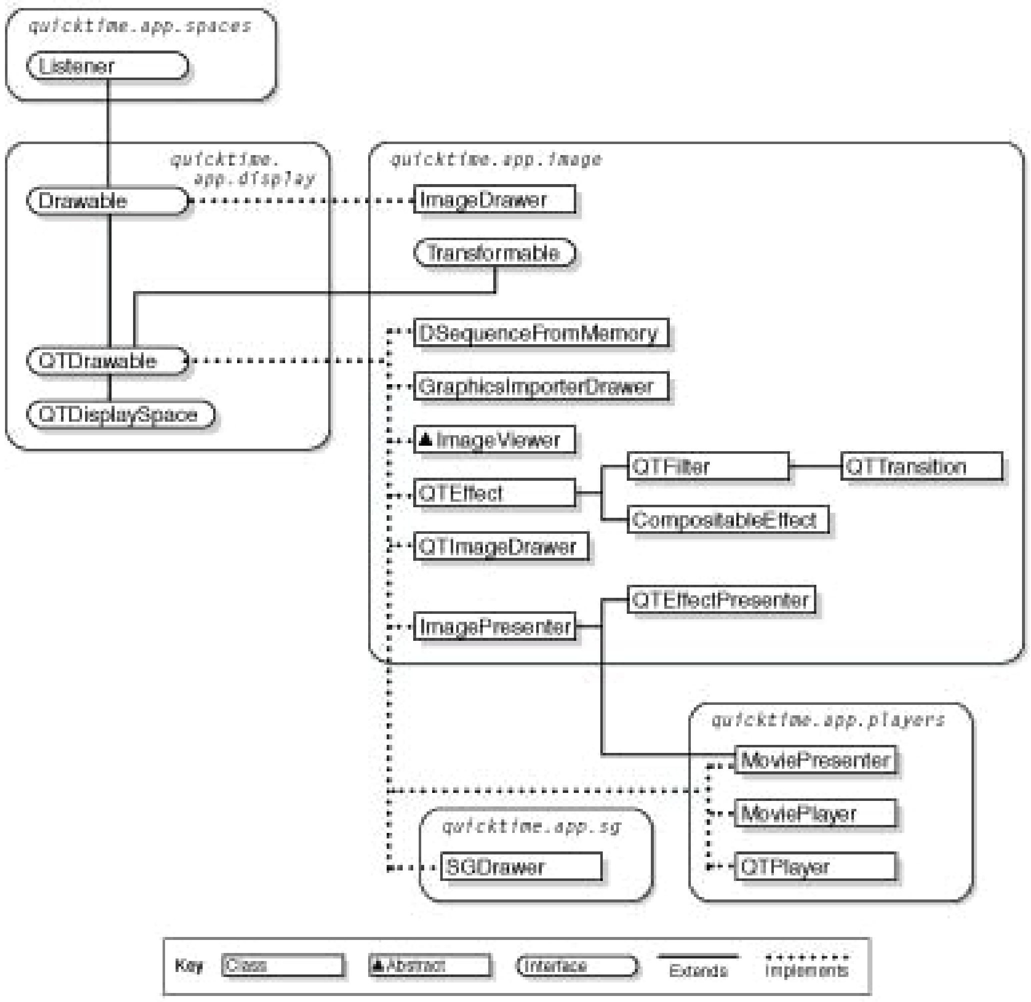
Dave Marshall
10/4/2001

The `QTDrawable` Interface

The `QTDrawable` interface encapsulates the common operations that can be applied to a QuickTime drawing object. It also presents an interface that expresses the required methods that the `QTCanvas` needs to call upon its drawing client.

Objects that implement the `QTDrawable` interface draw into the `QDGraphics` object presented to the client by its `QTCanvas`. As a consequence, this is not an appropriate interface for objects that are not QuickTime-based drawing objects. `QTExceptions` can be thrown by any of the methods in this interface and would indicate that either the graphics environment has changed in some unexpected way or that the media object itself is in some unexpected state.

Fig 9.2 represents the `QTDrawable` interfaces and the classes in QuickTime for Java that implement this interface.



QTDrawable interfaces and classes

- Working with the `QTDrawable` interface

Dave Marshall
10/4/2001

Working with the `QTDrawable` interface

As we've seen, the `QTDrawable` interface is used to handle the negotiation between the `QTCanvas` and one of several QuickTime objects. The QuickTime objects that implement the `QTDrawable` interface are extensive and include the following:

- the `QTPlayer` class, which presents the movie controller
- the `MoviePlayer` or `QTImageProducer` classes, which present the Movie data type
- the `GraphicsImporteDrawer` class, which wraps the graphics importer to present images from a data source, typically image files
- the `ImagePresenter` class, which presents the `DSequence` but for a single image only, allowing image data to be loaded and kept in memory, providing a faster redraw than the `GraphicsImporter` object
- the `SWCompositor` class, which presents the sprite world
- the `SGDrawer` class, which presents the sequence grabber and sequence grabber channel (video) components
- the `GroupDrawable` class, which groups `QTDrawable` objects into the same canvas.
- the `QTEffect` class, which wraps the visual effects architecture of QuickTime
- the `QTImageDrawer` class, which allows the results of Java painting into a java.awt.Image to be captured and drawn by QuickTime

The `QTDrawable` interface is designed to work hand-in-hand with a `QTCanvas` object. The class that implements this interface draws into the supplied QDGraphics object. The `QTCanvas` will call the methods of its client (setting its destination QDGraphics, setting display bounds, and redrawing it) as required.

These methods are used by the `QTCanvas` to notify the client object that it has been added to or removed from the `QTCanvas`. Various clients require this notification:

- `QTPlayer`, so it can declare its interest in mouse and key events
- `QTImageDrawer`, so that it can create an java.awt.Image object to paint into
- `QTDisplaySpaces` , to allow the attachment of controllers that are interested in receiving events whose source is the `QTCanvas`
- `setDisplayBounds()`, getDisplayBounds()

  These methods are used to get the current size of a `QTCanvas` client and set its size. The `QTCanvas` determines the size of itself and its client based on a complex interaction between the size the `QTCanvas` parent container allocates to the `QTCanvas`, the initial (or best) size of its client, and the setting of the resize and alignment flags of the `QTCanvas` itself.

Once the `QTCanvas` has determined the correct size of itself and its client, it uses the `setDisplayBounds()` method to resize and locate its client. The size and location of a `QTCanvas` and its client is always the same.

- `redraw()` This method is used to tell the client to redraw itself. Both the `setDisplayBounds()` and `redraw()` methods of a client are only called by the `QTCanvas` `paint()` method, with the `setDisplayBounds()` call, if required, being always called before a redraw() call.
- `getGWorld(), setGWorld()` These methods are used to set the `QTDrawable` client's destination QDGraphics if the client of a `QTCanvas` is a `QTDrawable`, as is normally the case. All `QTDrawable` objects must have a destination QDGraphics at all times. As such, if a `QTDrawable` client is removed from a `QTCanvas` or the `QTCanvas` has been hidden `myQTCanvas.setVisible` (false);

  then a special `QDGraphics`, `QDGraphics.scratch()`, is used to indicate to the `QTDrawable` that it is invisible and basically disabled. While the following methods are not used specifically in the relationship between `QTCanvas` and its client, they are presented here for the sake of completeness and also express capabilities that all `QTDrawable` objects possess:
- `getClip(), setClip()` All `QTDrawable` objects have the ability to clip their drawn pixels to a specified region. The `getClip()` and `setClip()` methods are used to get and set a clipping region. null can be used and returned, and it indicates that the `QTDrawable` currently has no clipping region set.

  The `DirectGroup` display space enables one or more `QTDrawable` objects to draw into the same `QTCanvas` destination QDGraphics. It provides the capability to layer the objects in such a group, so that the objects do not draw over each other. This layering is achieved through the use of clipping regions. The `DirectGroup` clips its members so that members that are behind others cannot draw into the area where those in front are positioned. As this group by definition draws directly to the screen, this is the only means that can be used to avoid the flickering that would occur if the `QTDrawable` objects could not be clipped.
- `getMatrix(), setMatrix(), getInitialSize()`

The `QTDrawable` interface also extends the `Transformable` interface.

The `Transformable` interface expresses the ability of QuickTime drawers to have a matrix applied that will map the source pixels of a drawing object to some transformed destination appearance. A matrix allows the following transformations to be applied in the rendering process:

- translation. Drawing pixels from an x or y location
- scaling. Scaling the image by an x or y scaling factor
- rotation. Rotating the image a specified number of degrees
- skew. Skewing the image by a specified amount
- perspective. Applying an appearance of perspective to the image

Your application can freely mix and match these different transformations at its own discretion.

All `QTDrawable` objects can have a matrix applied to them that will transform their visual appearance. A Sprite in a `SpriteWorld` can also have matrix transformations applied to it, as can a 3D model. The `TwoDSprite` also implements the `Transformable` interface.

The `Transformable` interface allows for applications to define behaviors that can be applied to any of these objects. For example, the `quicktime.app.actions.Dragger`, which will position objects in response to a `mouseDragged` event, is defined totally in terms of the `Transformable` interface. Thus, the `Dragger` can reposition any `TwoDSprite` in a `SWCompositor` or any `QTDrawable` in a `GroupDrawable`. Using interfaces to express common functionality is a powerful concept and is relied upon extensively in the QuickTime for Java Application Framework.

Next Up Previous

Next: The QTFactory Class **Up:** The QTDrawable Interface **Previous:** The QTDrawable Interface

Dave Marshall
10/4/2001

The `QTFactory` Class

The `QTFactory` class provides ``factory'' methods for creating classes used to present media that QuickTime can import. The `makeDrawable()` methods of `QTFactory` use the QuickTime importers to return an appropriate QuickTime object that can present any of a wide range of media types (images, movies, sounds, MIDI, and so on) that QuickTime can import.

Given a file, a Java `InputStream` object, or a URL, the `makeDrawable()` methods that belong to the `QTFactory` class examine the contained media and return an object that best presents that kind of media. For example:

- movies -- a `QTPlayer` object
- images -- a `GraphicsImporteDrawer` object
- sound media -- a `QTPlayer` object
- MIDI media -- a `QTPlayer` object

Once you have the `QTDrawable` object, you merely add it to the canvas and the visual component of the media is presented in the canvas' display space.

There are three versions of the `makeDrawable()` method. The first two methods deal with a file, either local or remote:

- `QTFactory.makeDrawable( QTFile, . . .)`
- `QTFactory.makeDrawable( String URL, . . .)`

  The `QTFile` version is a local file, whereas the URL version can use any of the protocols known to QuickTime:

  - file: for a local file
  - HTTP: for a remote file
  - FTP: for a remote file
  - RTSP: for a movie with streaming content

- `QTFactory.makeDrawable (InputStream`, . . .)

  In this method, the media data can be derived from any source -- for example, from a `ZipEntry`, from a local or remote file, or from memory.

The `readBytes()` method of the input stream is used to read all of the source object's bytes into memory. Then this memory is used to create the appropriate `QTDrawable`.

In the first two cases, QuickTime can use details about the file to determine the type of media that the file contains. Many media formats are not self-describing. That is, they don't have information in the data that describes what they are. Usually, the file type describes to QuickTime this important detail.

In the case of the `InputStream`, however, because the data can come from an unknown or arbitrary source, your application must describe to QuickTime the format of the source data so that it can import it successfully. As such, these methods require the provision of a `hintString` and `hintType`. You can specify that the hint is either the file extension, Mac OS file type, or ***MIMEType*** of the source data.

Once the `makeDrawable`() methods determine the media, they create either a Movie or a `GraphicsImporter` as the QuickTime object to present that media. They then pass off this Movie or `GraphicsImporter` to a `QTDrawableMaker` to return a `QTDrawable` object to present that Movie or `GraphicsImporter`.

Your application uses the default `QTDrawableMaker` if you do not specify one. This returns a `QTPlayer` that creates a `MovieController` for the Movie or a `GraphicsImporteDrawer` for the `GraphicsImporter`. An application can also specify a custom `QTDrawableMaker` that will create a required object for one or both of these two cases.

The `QTFactory` also provides methods to locate files in the known directories of the Java application when it executes. In order for QuickTime to open a local file, it must have the absolute path and name of the file. Your application may want to open files where it can know the relative location of the file from where it is executing. For instance, many of the code samples in the QuickTime for Java SDK will look for files in the media directory that is in the SDK directory. This media directory is added to the class path when the application is launched. The `QTFactory.findAbsolutePath()` method is used by the application to find the absolute path of the file, and to find those files, at runtime.

- `addDirectory()` Allows your application to specify a directory that is added to the known directories that are used in searching for files
- `removeDirectory`() Allows your application to remove directories where it doesn't want files to be found. This can shorten the search process, as the application can remove the specified directory from the search paths that the find()... methods will search for specified files in.
- `findAbsolutePath`() Given a relative file (or directory), this method will search for this specified path in

    - application-specified directories
    - user.dir, which is a directory that is one of the system properties of the Java runtime
    - any directory known in the class path directories. class.path is also a system property of the Java runtime.

  This method searches for the specified path as this path is appended to these known directories. It returns the first occurrence of the file (or directory) that exists in these locations. If the specified file is not found, a `FileNotFoundException` is thrown.
- `findInSystemPaths()` Whereas the `findAbsolutePath()` will look for the specified file by appending that file to the known directories, this method will do a recursive search for the specified file or subdirectory of all of the known or registered directories and their parents. This can be a time-consuming search, and typically your application will only use this method to find a file that the user has misplaced or is not found in the `findAbsolutePath()` method. If the specified file is not found, a `FileNotFoundException` is thrown.

Next Up Previous

Next: The Spaces and Controllers **Up:** Quicktime for Java Classes **Previous:** Working with the QTDrawable

Dave Marshall
10/4/2001

The Spaces and Controllers Architecture

The QuickTime for Java API introduces an abstraction known as a space. A space defines and organizes the behavior of objects and allows for the complex representation of disparate media types. Spaces provide a powerful and useful means to assemble and control a complex presentation.

You use a space to create a ``world'' that can be controlled dynamically from Java at runtime. This world could have certain characteristics that define how the objects exist and interact with others. Your application assembles the space with its objects or members and uses Java's event model to allow for user interaction with those objects. Because it is a dynamic environment, decisions about behavior and even which members belong to the space can be deferred until runtime. The Space interface, which is part of the `quicktime.app`.spaces package, provides the standard API that all spaces support.

QuickTime for Java controllers manipulate Java objects in spaces. These controllers can define the standard behavior for a group of objects by defining the behavior of objects over time, by monitoring objects, and by responding to user events. Controllers provide a uniform way of enforcing the same behavior on a group of objects. The behavior of a controller depends upon the support protocols defined by a space and by a controller itself.

The QuickTime for Java API provides a Controller interface and defines a protocol for the interaction between spaces and controllers. While the focus in this release of QuickTime for Java is in presentation, the architecture is general enough to be applied in the model space for generating data.

Dave Marshall
10/4/2001

Displaying and Streaming Movies

QuickTime provides a set of APIs that enable you to stream multimedia content over a network in real time, as opposed to downloading that content and storing it locally prior to presentation. With streaming, the timing and speed of transmission as well as the display of data are determined by the nature of the content rather than the speed of the network, server, or the client. Thus, a one minute long QuickTime movie is streamed over a network so that it can be displayed or presented in one minute of real time.

In the QuickTime streaming architecture, a stream is, simply, a track in a movie. QuickTime lets you stream a broad variety of content -- audio, video, text, and MIDI; the output of any audio or video codec supported in QuickTime can be streamed, in fact. If your application is QuickTime-savvy, you can automatically take advantage of this multimedia streaming capability.

In addition to demonstrating some useful techniques for displaying QuickTime movies, this section shows you how to play a streaming movie from a URL. It builds on the concepts and examples discussed in the previous sections and shows you how to

- select and then play a QuickTime movie with its controller detached
- display a QuickTime movie within a window and add callbacks that are triggered at some specific time during movie playback

---

Dave Marshall

Playing a Streaming Movie

This example builds on the `QTSimpleApplet` code discussed in ``The `QTSimpleApplet` Code''. This applet enables you to play a steaming movie from a URL.

You define the instance variables for the applet:

```
private Drawable myQTContent;
private QTCanvas myQTCanvas;
```

Just as with the `QTSimpleApplet` code sample, you can use the standard `init(), start(), stop()`, and `destroy()` methods to initialize, execute, and terminate the applet. Likewise, you call `QTSession.open()`in order to make sure that QuickTime is present and initialized. Again, this is a required call before any QuickTime for Java classes can be used. It is called first in the `init()` method. In order to shut down QuickTime properly, you also need to call `QTSession.close()` if you have previously called `QTSession`.open(). This is called in the `destroy()` method.

`QTCanvas`, as we've seen, is a display space into which QuickTime can draw and receive events. `QTCanvas` provides the output destination for QuickTime drawing. You set up a `QTCanvas` to display its content at its original size or smaller and centered in the space given to the `QTCanvas` when the applet is laid out. The `QTCanvas` is initialized to display its client up to as large as that client's initial size. And 0.5F flags are used to position the canvas at the center of the space allocated to it by its parent container's layout manager:

```
setLayout (new BorderLayout());
myQTCanvas = new QTCanvas (QTCanvas.kInitialSize, 0.5F, 0.5F);
add (myQTCanvas, "Center");
```

You need to set the client as a Drawable object that can display into the canvas. The QuickTime logo is displayed when there is no movie to display. Thus, `ImageDrawer` is set up as the initial client of `QTCanvas`.

```
myQTContent = ImageDrawer.getQTLogo();
```

You enter the URL to a QuickTime movie to be displayed in a text field:

```
final TextField urlTextField = new TextField (
"file:///... Enter an URL to a movie",
30);
```

You set the font and font size in the text field for the URL. The initial string is displayed in the text field. You add an `ActionListener` so that the events taking place on the text field are captured and executed. `tf.getText()` returns the URL that the user has entered:

```
urlTextField.setFont (new Font ("Dialog", Font.PLAIN, 10));
urlTextField.setEditable (true);
urlTextField.addActionListener (new ActionListener () {
TextField tf = urlTextField;
public void actionPerformed (ActionEvent ae) {
```

```
myQTContent = QTFactory.makeDrawable (tf.getText());
myQTCanvas.setClient (myQTContent, true);
}
});
```

The URL can support the usual protocols:

- file -- a local file on the user's computer
- HTTP -- HyperText Transfer Protocol
- FTP -- File Transfer Protocol
- RTSP -- Real Time Streaming Protocol

The URL can also point to any media file (movies, images, and so on) that QuickTime can present. The single `makeDrawable()` method will return the appropriate QuickTime object to present the specified media. Once created, this QuickTime object is set as the client of the `QTCanvas` and can then be viewed and/or played by the user.

Note that no error handling is done in the code in this example to keep things simple.

The full code for the `QTStreamingApplet.java` is given below:

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

import quicktime.*;
import quicktime.io.QTFile;

import quicktime.app.QTFactory;
import quicktime.app.display.*;
import quicktime.app.image.ImageDrawer;

public class QTStreamingApplet extends Applet {

        private Drawable myQTContent;
        private QTCanvas myQTCanvas;
    private Vector urlTable;
    private TextField urlTextField;
    private PopupMenu pm;

        public void init () {
                try {
                        QTSession.open();
                                // set up a QTCanvas which will disply its
content
                                // at its original size of smaller and
centered in the space given
                                // to the QTCanvas when the applet is
layed out
                        setLayout (new BorderLayout());
                        urlTable = new Vector();
                        myQTCanvas = new QTCanvas (QTCanvas.kInitialSize,
0.5F, 0.5F);
                        add (myQTCanvas, "Center");

                        myQTContent = ImageDrawer.getQTLogo();
                        add (bottomPanel(), "South");
                        readURL();

                } catch (QTException qtE) {
                                //in this case the only QTException is in
QTSession.open()
                        throw new RuntimeException (qtE.getMessage());
                }
        }

        public void start () {
                try { // if QT was not successfully initialized the
QTCanvas will be null
                        if (myQTCanvas != null)
                                myQTCanvas.setClient (myQTContent, true);
                } catch (QTException e) {
                        e.printStackTrace();
                }
        }

        public void stop () {
                if (myQTCanvas != null)
                        myQTCanvas.removeClient();
        }

        public void destroy () {
                QTSession.close();
        }


        /**
         * creates drawable object from the URL the user has entered
         * or from the menu item chosen
         */
        private void createNewMovieFromURL(String selURL) throws
QTException {
                String url = urlTextField.getText();
                myQTContent = QTFactory.makeDrawable (url);
                myQTCanvas.setClient (myQTContent, true);
        }

        /**
         * the parameter list in the html file should
  * have a name = total having the value of the total
         * number of url's to be appended to the popup menu.
  * Each url should start with "url" + i , where i
         * is 1, 2.... in increasing order till the i = total.
         */
        private void readURL() throws QTException{
                for ( int i = 0; i <
Integer.parseInt(getParameter("total")); i++) {
                        String url = getParameter("url" + (i + 1));
                        if ( url != null)
                                appendItem(url);
                }
        }
        public Component bottomPanel () {
                Panel row2 =  new Panel();
                row2.setLayout(new FlowLayout(FlowLayout.CENTER));
                row2.setBackground(Color.white);

                urlTextField = new TextField ("file:///... Enter an URL to
a movie", 40);
         //Enter URL to movie here", 30);
                urlTextField.setFont (new Font ("Dialog", Font.PLAIN,
10));
                urlTextField.setEditable (true);
                urlTextField.addActionListener (new ActionListener () {
                        //TextField tf = urlTextField;

                        public void actionPerformed (ActionEvent ae) {
                                if (myQTCanvas != null) {
                                        try {

createNewMovieFromURL(urlTextField.getText());

appendItem(urlTextField.getText());
                                        } catch (QTException e) {
                                                        //probably a non-
fatal error that the Applet
                                                        //should deal with
more informatively
                                                e.printStackTrace();
                                        }
                                }
                        }
                });

                row2.add(urlTextField);

                pm = new PopupMenu();

                Image img1 = Toolkit.getDefaultToolkit().getImage(
         getCodeBase().getFile() + "images/p1.gif");
                Image img2 = Toolkit.getDefaultToolkit().getImage(
         getCodeBase().getFile() + "images/r1.gif");
                IconButton menuButton =
       new PopupMenuButton (img1, img2, pm);

                row2.add (menuButton);

                return row2;
        }

        /**
         * This appends the input movie url entered in the textfield or
         * selected from the Choose movie menu to the PopupMenu. It checks
         * for its duplicates and then appends it to the list in the
popupmenu
     * @param     urlItem  Item to be appended to the PopupMenu
         */
    private void appendItem(String urlItem) throws QTException {
        if (urlTable.contains(urlItem) == false) {
                MenuItem item = new MenuItem(urlItem);
                pm.add (item);
                item.addActionListener (new ActionListener () {
                        public void actionPerformed(ActionEvent event) {
                                try {
                                        MenuItem jm =
(MenuItem)event.getSource();
                                        String selURL = jm.getLabel();

                                        urlTextField.setText(selURL);
                                        createNewMovieFromURL(selURL);
                                        }catch (QTException ex) {
                                                ex.printStackTrace();
                                        }
                                }
                        });
                urlTable.addElement(urlItem);
        }
    }
}
```

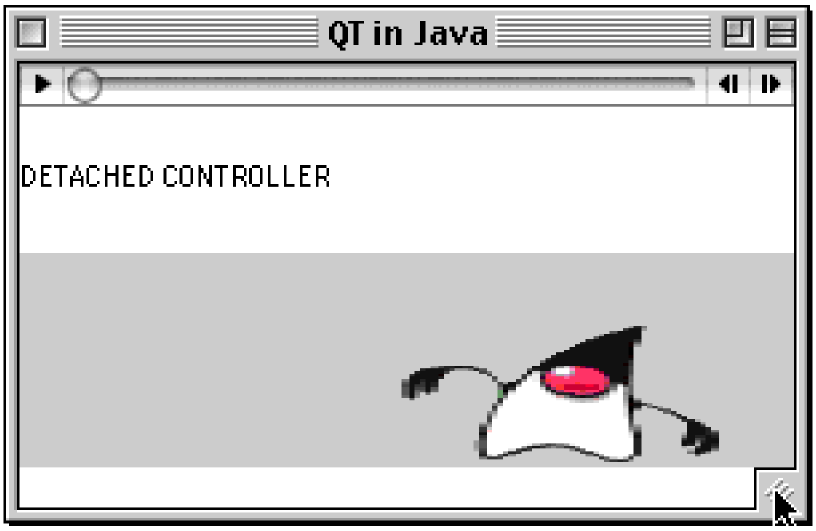
For supporting code required for this example (`IconButton.java` and `PopupMenuButton.java` refer to the sample code folder online for this example.

Dave Marshall
10/4/2001

Using The Detached Controller

The code sample in this section shows how to select and then play a QuickTime movie with its controller detached. The media required is a QuickTime movie of the user's choice.

You are prompted to select a movie file. If you make the selection, a window is constructed and the movie and its controller are presented, as shown in Figure 9.3



The Detached Controller

The movie and its controller are displayed separately in the same window, with the controller at the top and the actual movie at the bottom. They are separated by a label. Though the movie and its controller are shown here in the same window, they could also be displayed in different windows, even on different monitors.

You use a `QTPlayer` object to play the movie in its canvas, and you can use a `MoviePlayer` object to present the movie.

```
void setControllerCanvas(Movie mMovie) throws QTException {
QTCanvas controllerCanvas = new QTCanvas();
MovieController mController = new MovieController(mMovie,
mcScaleMovieToFit);
mController.setAttached(false);
QTPlayer qtPlayer = new QTPlayer(mController);
add(controllerCanvas, "North");
controllerCanvas.setClient(qtPlayer, true);
}
```

You can attach or detach the movie from its controller using this method:

```
mController.setAttached(false);
```

Once the controller is a client of its canvas, if the movie is reattached to the controller, you need to notify the canvas of this change in the client's display characteristics.

Now you create the canvas for the detached movie:

```
void setMovieCanvas(Movie mMovie) throws QTException{
QTCanvas movieCanvas = new QTCanvas();
MoviePlayer mPlayer = new MoviePlayer(mMovie);
add(movieCanvas, "South");
movieCanvas.setClient(mPlayer, true);
}
```

The following snippet shows the code that assembles the Detached-Controller window. Resizing the window resizes the movie and the width of the controller but not its height.

```
DetachedController(Movie mMovie) throws QTException {
super ("QT in Java");
setControllerCanvas(mMovie);
setMovieCanvas(mMovie);
add (new Label("DETACHED CONTROLLER"), "Center");
.
.
.
}
```

The full code listing for the `DetachedController.java` example can be found in the online code examples.

Dave Marshall
10/4/2001

Converting To Full Screen

The `FullScreen` class provides the capability for converting a screen to full-screen mode and back to normal mode. The QuickTime for Java API allows you to put the specified screen into full screen mode and then use a Java window to fill the screen.

To do this, you use the `FullScreenWindow` class, which is a subclass of the `java.awt.Window` object. The `FullScreenWindow` class internally manages a `FullScreen` object, and when the `show()` method is called, it puts the screen into full-screen mode and fills up the screen with an `awt.Window`. This is very useful because you can get the complete functionality of using a Java AWT Window but in full-screen mode.

The movie is created in a similar fashion as the `QTStreamingApplet`. The program also creates a menu that allows the user to select a movie and once opened provides a Present Movie menu item to present the movie in full screen mode.

The full code listing is available in the online code examples.

The main parts of the code are as follows:

You present the movie in full screen mode and use the current screen resolution and current movie. The `QTCanvas` is created using the performance `resize` flag. This ensures that the movie is displayed at its original size or at a multiple of 2:

```
FullScreenWindow w = new FullScreenWindow(new FullScreen(),
myPlayMovie);
MoviePlayer mp = new MoviePlayer (myPlayMovie.getMovie());
QTCanvas c = new QTCanvas (QTCanvas.kPerformanceResize, 0.5F, 0.5F);
w.add (c);
w.setBackground (Color.black);
```

You remove the movie from its current `QTCanvas` and put the movie into the new canvas of the `FullScreen`Window. You do this because a `QTDrawable` can only draw to a single destination `QDGraphics`:

```
myPlayMovie.getCanvas().removeClient();
c.setClient (mp, false);
```

`HideFSWindow` is a `MouseListener`. A `MouseListener` is installed on both the `QTCanvas` and the Window. The window is then shown, which will put the window into full-screen mode:

```
w.show();
HideFSWindow hw = new HideFSWindow (w, myPlayMovie, c);
w.addMouseListener (hw);
```

```
c.addMouseListener (hw);
```

As `MoviePlayer` object is used to present the movie, this shows just the movie and not a controller. So finally, you start the movie playing:

```
mp.setRate (1);
```

When the user presses the mouse, the movie is restored to its previous `QTCanvas` and the full-screen window is hidden:

```
public void mousePressed (MouseEvent me) {
try {
c.removeClient();
pm.getCanvas().setClient (pm.getPlayer(), false);
} catch (QTException e) {
e.printStackTrace();
} finally {
w.hide();
}
}
```

The user must explicitly press the mouse to hide the `FullScreenWindow`. However, your application could also define an `ExtremeCallBack` that would automatically hide the `FullScreenWindow` when the movie is finished playing. Callbacks are discussed in the next section.

Next Up Previous
Next: Using Movie Callbacks **Up:** Displaying and Streaming Movies **Previous:** Using The Detached Controller

Dave Marshall
10/4/2001

Using Movie Callbacks

This section explains how to display a QuickTime movie within a window and add ***callbacks***. The ***callbacks*** are QuickTime calling back into Java through the movie controller, movie, and QuickTime VR APIs. Callbacks can be used by an application to perform its own tasks when certain conditions occur within QuickTime itself. The callbacks used in the `MovieCallbacks` program are invoked when some condition having to do with the presentation of a movie is changed:

- In the movie, the `DrawingComplete` procedure is used to notify the Java program whenever QuickTime draws to the screen.
- `ActionFilter` procedures are used. This subclass overrides those actions that pass on no parameters and a float parameter.
- The movie contains QuickTime VR content, and a number of QTVR callbacks are installed for panning and tilting, for hot spots, and for entering and leaving nodes.

The `QTCallBack`, `ActionFilter`, or `DrawingComplete` callbacks are invoked through a direct or indirect call of the QuickTime `MoviesTask` function.

Many of the callback methods in QuickTime in Java are required to execute in place, in that QuickTime requires a result code in order to proceed. These callbacks provide meaningful feedback when their `execute()` method returns. The subclasses of `QTCallBack`, however, can execute asynchronously, in which case QuickTime does not require a result code in order to proceed. This is also true of any of the `execute()` methods with no return value.

The program just prints out details about the callback when it is invoked. The QuickTime API documentation provides examples and discussions on the usage of these.

To set up a movie drawing callback:

```
static class MovieDrawing implements MovieDrawingComplete {
public short execute (Movie m) {
System.out.println ("drawing:" + m);
return 0;
}
}
```

To set up an action filter:

```
static class PMFilter extends ActionFilter {
public boolean execute (MovieController mc, int action) {
System.out.println (mc + "," + "action:" + action);
return false;
}
public boolean execute (MovieController mc, int action, float value) {
System.out.println (mc + "," + "action:" + action + ",value=" + value);
return false;
}
}
```

The following code sets up callbacks for QuickTime VR content:

```
Track t = m.getQTVRTrack (1);
if (t != null) {
QTVRInstance vr = new QTVRInstance (t, mc);
vr.setEnteringNodeProc (new EnteringNode(), 0);
vr.setLeavingNodeProc (new LeavingNode(), 0);
vr.setMouseOverHotSpotProc (new HotSpot(), 0);
Interceptor ip = new Interceptor();
vr.installInterceptProc (QTVRConstants.kQTVRSetPanAngleSelector, ip, 0);
vr.installInterceptProc (QTVRConstants.kQTVRSetTiltAngleSelector, ip, 0);
vr.installInterceptProc (QTVRConstants.kQTVRSetFieldOfViewSelector, ip,
0);
vr.installInterceptProc (QTVRConstants.kQTVRSetViewCenterSelector, ip, 0);
vr.installInterceptProc (QTVRConstants.kQTVRTriggerHotSpotSelector, ip,
0);
vr.installInterceptProc (QTVRConstants.kQTVRGetHotSpotTypeSelector, ip,
0);
}
static class EnteringNode implements QTVREnteringNode {
public short execute (QTVRInstance vr, int nodeID) {
System.out.println (vr + ",entering:" + nodeID);
return 0;
}
}
static class LeavingNode implements QTVRLeavingNode {
public short execute (QTVRInstance vr, int fromNodeID, int toNodeID,
boolean[]
cancel) {
System.out.println (vr + ",leaving:" + fromNodeID + ",entering:" +
toNodeID);
// cancel[0] = true;
return 0;
}
}
static class HotSpot implements QTVRMouseOverHotSpot {
public short execute (QTVRInstance vr, int hotSpotID, int flags) {
System.out.println (vr + ",hotSpot:" + hotSpotID + ",flags=" + flags);
return 0;
}
}
```

The following code shows how to install a `QTRuntimeException` handler:

```
QTRuntimeException.registerHandler (new Handler());
```

Runtime exceptions can be thrown by many methods and are thrown where the method does not explicitly declare that it throws an exception. Using the `QTRuntimeHandler` allows your application to examine the exception and determine if it can be ignored or recovered from. An example of this is the `paint()` method of a Java component. Its signature is

```
public void paint (Graphics g);
```

It is declared not to throw any exceptions. However, the `QTDrawable redraw()` or `setDisplayBounds()` calls (which are both invoked on a `QTCanvas` client) are defined to throw `QTExceptions`. If the `QTCanvas` client does throw an exception in this case, it passes it off to a `QTRuntimeHandler` if the application has registered one, with details about the cause of the exception. The application can then either throw the exception or rectify the situation. If no runtime exception handler is registered, the exception is thrown and caught by the Java VM thread itself.

Next Up Previous

Next: Media AND Presenters **Up:** Displaying and Streaming Movies **Previous:** Converting To Full Screen

Dave Marshall
10/4/2001

Media AND Presenters

QuickTime understands media through the pairing of media data and a metadata object that describes the format and characteristics of that media data. This media data can exist either on some external storage device or in memory. The media data can further be stored in the movie itself or can exist in other files, either local or remote. The metadata object that describes the format of the media data is known as a `SampleDescription`. There are various extensions to the base-level `SampleDescription` that are customized to contain information about specific media types (***e.g.***, `ImageDescription` for image data).

This section discusses the following topics:

- retrieving media data and QuickTime's use of this complex and powerful capability, particularly with the introduction of multimedia streaming
- using presenters to express an object that renders media data loaded into or residing in memory
- the `ImagePresenter` class, which is the principal class in the QuickTime for Java API that presents such image data

---

Dave Marshall
10/4/2001

Media Data and Movies

With the introduction of QuickTime streaming, QuickTime has extended its ability to retrieve data. Traditionally, media data had to exist as local data. With streaming, media data can now exist on remote servers and retrieved using the HTTP or FTP Internet protocols, as well as the more traditional file protocol of previous releases. The location of media is contained in a structure called a `DataRef`. This tells QuickTime where media data is and how to retrieve it.

QuickTime streaming also adds support for broadcast media data, using the RTSP protocol. Media data of this format is delivered through network protocols, typically sourced through the broadcast of either live or stored video. It requires the creation of streaming data handlers to deal with the mechanics of retrieving media data using this protocol. The tracks in a single movie can have data in different locations. For example, one track's media data might be contained in the movie itself; a second track's media data might exist in another local file; and a third track's media data might exist in a remote file and retrieved through the FTP protocol. In these two cases, the movie references the media data that is stored elsewhere. A fourth track might retrieve broadcasted media data through a network using the RTSP protocol. Your application is free to mix and match these data references for tracks' media data as appropriate. QuickTime presents a complex and powerful capability to deal with media data; a complexity that provides the user of QuickTime with a powerful tool in both the development and delivery of media content.

Dealing With Media

Despite the complexity of the data retrieval semantics of QuickTime, the process of dealing with media in QuickTime is quite simple. In the `CreateMovie` sample code, the movie's data is constructed by inserting the media sample data and its description into the media object and then inserting the media into the track.

The following is a printout of the `ImageDescription` for the last frame of the image data that is added to the movie. As you can see, the `ImageDescription` describes the format, the size, and the dimensions of the image itself:

```
quicktime.std.image.ImageDescription[
      cType=rle ,
      temporalQuality=512,
      spatialQulity=512,
      width=330,
      height=140,
      dataSize=0,
      frameCount=1,
      name=Animation,
      depth=32]
```

The following is the `SoundDescription` of the sound track added in this sample code. It describes the sample, sample rate, size, and its format:

```
quicktime.std.movies.media.SoundDescription[
      format=twos,
      numChannels=1,
      sampleSize=8,
      sampleRate=22050.0]
```

If the data is not local to the movie itself, one inserts a sample description (as previously to describe the data) and a DataRef that describes to QuickTime both the location of the data and the means it should use to retrieve the data when it is required. Once assembled, QuickTime handles all of the mechanics of retrieving and displaying the media at runtime.

When media is displayed, the media handlers use the various rendering services of QuickTime. These rendering services are based on the same data model, in that the sample description is used to both describe the media and to instantiate the appropriate component responsible for rendering data of that specific format. These rendering components are also available to the application itself outside of movie playback.

For example, the `DSequence` object is used to render image data to a destination QDGraphics and used throughout QuickTime to render visual media. Your application can also use the `DSequence` object directly by providing both the image data and an `ImageDescription` that describes the format and other characteristics of this data and, of course, a destination `QDGraphics` where the data should be drawn. Like QuickTime itself, your application can apply matrix transformations, graphics modes, and clipping regions that should be applied in this rendering process.

Similar processes are used and available for all of the other media types (sound, music, and so on) that QuickTime supports, with each of these media types providing their own extended sample descriptions, media handlers, and rendering services.

* The `ImageSpec` Interface

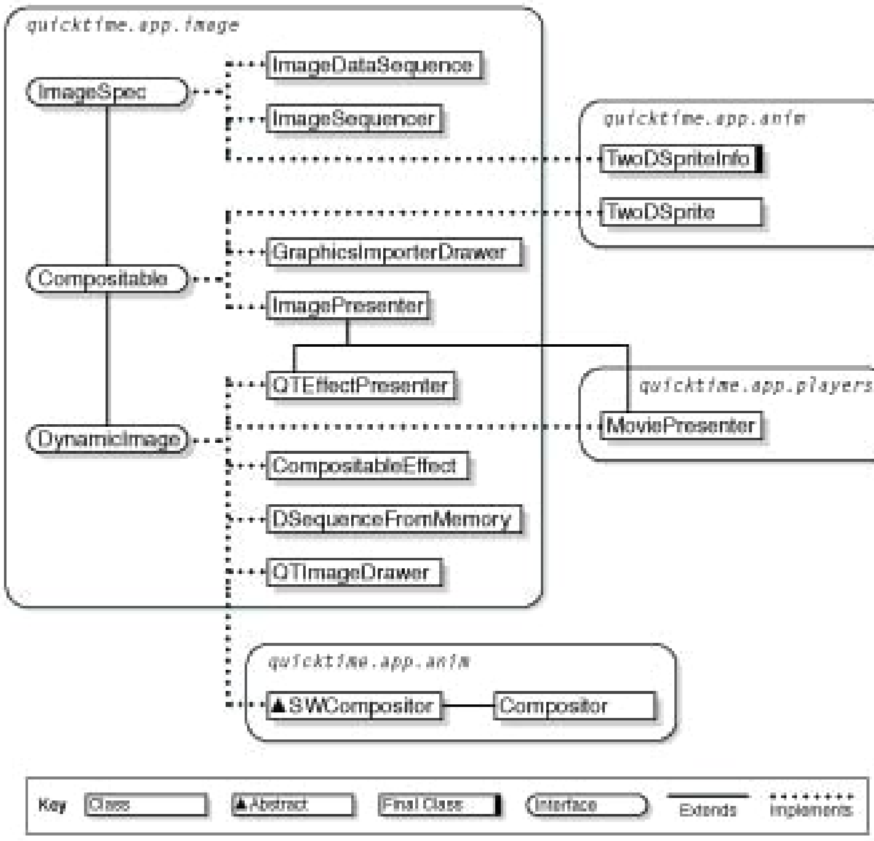
Next Up Previous

Next: The ImageSpec Interface **Up:** Media AND Presenters **Previous:** Media AND Presenters

Dave Marshall
10/4/2001

The ImageSpec Interface

The `ImageSpec` interface expresses the close relationship between image data and an `ImageDescription` that describes it. Figure 9.4 illustrates all of the `ImageSpec`-derived classes in the QuickTime for Java API.



The ImageSpec Interface

The image data itself is represented by an object that implements the `EncodedImage` interface. This interface allows image data to be stored in either a (Java) `int` or `byte` array. Any object that implements the `ImageSpec` interface can have its image rendered by either the `ImagePresenter` to a destination `QDGraphics` or by a `TwoDSprite` to its container `SWCompositor`.

The `ImageSpec` interface expresses the commonality of QuickTime's media model, specifically with regard to image data, and unifies the many possible constructions and imaging services that QuickTime provides. If your application requires a particular format for generating image data, then it can implement the `ImageSpec` interface and thus have QuickTime use this custom class wherever the QuickTime for Java API uses existing `ImageSpec` objects.

The `Compositable` and `DynamicImage` interfaces extend the `ImageSpec` interface. The `Compositable` interface captures the ability of image data to have a graphics mode applied to it when it is rendered. Graphics modes include rendering effects such as transparency, where any pixels of a particular color in the image data won't be drawn, and blending, where all of the drawn colors of an image are blended with a blend color to alter the rendered image.

The `DynamicImage` interface extends `Compositable` and expresses the fact that some pixel data may change. This interface is used by the `Compositor`, as a `TwoDSprite` must invalidate its Sprite if the pixel data changes.

Next Up Previous

Next: QuickTime For Java Presenters **Up:** Media Data and Movies **Previous:** Media Data and Movies

Dave Marshall
10/4/2001

QuickTime For Java Presenters

When QuickTime plays back a movie, it does not generally read all of its media data into memory but rather reads chunks of data as required. A movie can be constructed that requires the media data to be loaded into memory or other parameters that will require more memory to be used but will generally improve the quality of the rendered movie. The QuickTime API documentation covers these customisations that a movie's author can make.

Due to the usefulness of, and in some cases requirement for, loading media data into memory, QuickTime for Java provides presenters. A presenter is an object that renders media data that is loaded or resides in memory. A presenter also uses a QuickTime service to render this media data.

QuickTime for Java has a built in `ImagePresenter` for rendering image data. The `ImagePresenter` class implements the `QTDrawable` interface. The `ImagePresenter` uses a `DSequence` to perform the rendering of the image data. It is the primary object that is used in QuickTime for Java to render image data. The `TwoDSprite` is also a presenter that presents image data loaded into memory. However, its role is specific to the membership of its Sprite in a `SpriteWorld` (which is represented in QuickTime for Java by the `SWCompositor`).

Though only an `ImagePresenter` is provided in the current release of QT4Java, a similar design strategy could be employed with other media types. For example, let's consider the music media type. `MusicMedia` is rendered in QuickTime by the `TunePlayer` class. A `MusicPresenter` class could be created that used the `TunePlayer` to render the `MusicData`. A `MusicSpec` interface could be described that returns a `MusicDescription` and the raw `MusicData` of the tune events.

The `ImagePresenter` manages the varying state and conditions of use of QuickTime's `DSequence` renderer. It is a useful abstraction because it hides such details from the user, creating a robust and reusable class.

The `ImagePresenter` is able to render its data faster than its corollary `GraphicsImporterDrawer`, which also implements the `QTDrawable` interface, for two reasons. First, the data is kept in memory and so it is quicker to read. (The `GraphicsImporteDrawer` reads its image data from its `DataRef`, typically a file.) Second, the image data of an `ImagePresenter` can be (and often is) kept in a format that is optimized for rendering or decompression. However, a `GraphicsImporteDrawer` will use less memory and is often more than sufficient for presenting an image where no demanding rendering tasks are required, such as constant, time-sensitive redrawing.

An `ImagePresenter` object can be created from many sources. For example, from a file:

```
ImagePresenter myImage = ImagePresenter.fromFile(imageFile);
```

In this case, the `ImagePresenter` will create a `GraphicsImporter` to read and load into memory the image data from the file. It will then decompress the image if necessary to a format optimized for rendering.

You can also create an `ImagePresenter` from generated image data and an `ImageDescription` that describes it:

```
int width = 100;
int height = 100;
IntEncodedImage myImageData = new IntEncodedImage (width * height);
//...fill in pixel values using standard java ARGB ordering
ImageDescription myDescription =
ImageDescription.getJavaDefaultPixelDescription (width,
height);
myDescription.setDataSize (myImageData.getSize());
ImagePresenter myPresenter =  ImagePresenter.fromQTImage(myImageData,
myDescription);
```

You can also create an `ImagePresenter` object from an onscreen or offscreen `QDGraphics` that you have previously drawn into:

```
QDGraphics gWorld = ....
Rect rect = ....
int colorDepth = ....
int quality = ....
int codecType = ....
CodecComponent codec = ....
ImagePresenter myIP = ImagePresenter.fromGWorld (myGWorld,
          myGWorld.getBounds(),
          myGWorld.getPixMap().getPixelSize(),
          myDerivedCompressionQuality,
          myDerivedCompressionType,
          myDerivedCodecComponent);
```

- Subclasses of `ImagePresenter`

Next Up Previous

Next: Subclasses of ImagePresenter **Up:** QuickTime for Java **Previous:** The ImageSpec Interface

Dave Marshall
10/4/2001

Subclasses of `ImagePresenter`

There are two basic subclasses of `ImagePresenter`, which inherit from and build on the features of `ImagePresenter`: `QTImageProducer` and `QTEffectPresent`. Both of these subclasses work in a similar fashion. They render their target objects (a movie or an effect) into an offscreen `QDGraphics` object. The pixel data from this `QDGraphics` is then set as the `EncodedImage` data of the superclass. An `ImageDescription` is created that describes this raw pixel data and this description is given to the superclass.

Thus, retrieving the image data (`getImage()`) returns the raw pixel data that the movie has drawn into. The `ImagePresenter` superclass then blits this raw pixel data to its destination `QDGraphics`. If this presenter is added as a Sprite, the raw pixel data becomes that sprite's image data. `QTImageProducer` takes a Movie object and implements the same interfaces as `MoviePlayer`, but it has the extra capability of being a presenter and having a graphics mode set for its overall appearance. `QTEffectPresent` is similar to `QTImageProducer` but takes a `QTEffect` object. You could use this presenter to capture the results of applying a filter to a source image. You could then discard the filter object and just present the resultant image. You can also use this presenter to present a character in a sprite animation that could transition on and off the stage.

Dave Marshall
10/4/2001

Imaging and Effects

The QuickTime for Java Application Framework provides a number of classes that handle the presentation of images within a Java display space. These classes provide utility methods that you can use to set transparent colors in images, apply visual effects, or create objects for handling sequences of images (such as slide shows). Other methods en-able you to create `QTDrawable` objects that read image data from a file or load the data into memory.

This section shows you how to

- draw an image file using the `GraphicsImporteDrawer`
- create a java.awt.Image out of an image in QuickTime's format
- use the `QTImageDrawer` class to render Java-painted content in a QuickTime graphics space
- take advantage of QuickTime's visual effects architecture to apply transitions between two images

---

Dave Marshall
10/4/2001

Drawing An Image File

The `ImageFileDemo.java` code listing (below) shows how to import and draw an image from a file. This program works with the `GraphicsImporteDrawer` object to import and display a variety of image file formats. The media required for this sample code is any image file that can be imported using the `GraphicsImporterDrawer`. The `quicktime.app.image` package has two primary image display classes: `GraphicsImporteDrawer` and `ImagePresenter`. These classes implement both `ImageSpec` and `QTDrawable` interfaces.
`GraphicsImporterDrawer` uses `GraphicsImporter` to read, decompress, and display image files.

The `ImageFileDemo.java` code listing:

```
import java.awt.*;
import java.awt.event.*;

import quicktime.*;
import quicktime.std.StdQTConstants;
import quicktime.std.image.GraphicsImporter;
import quicktime.io.QTFile;

import quicktime.app.display.QTCanvas;
import quicktime.app.image.*;

/** QuickTime Graphics Importer and Codec demo */
public class ImageFileDemo extends Frame implements StdQTConstants {

        public static void main (String args[]) {
                try {
                        QTSession.open();

                        int[] fileTypes = { kQTFileTypeGIF,
kQTFileTypeJPEG, kQTFileTypePicture };
                        QTFile qtf = QTFile.standardGetFilePreview
(fileTypes);

                        ImageFileDemo ifd = new ImageFileDemo (qtf);
                        ifd.pack();
                        ifd.show();
                        ifd.toFront();
                } catch (QTException e) {
                        if (e.errorCode() != Errors.userCanceledErr)
                                e.printStackTrace();
                        QTSession.close();
                }
        }

        ImageFileDemo (QTFile qtf) throws QTException {
                super (qtf.getName());

                QTCanvas canv = new QTCanvas();
                add (canv, "Center");
                GraphicsImporterDrawer myImageFile = new
GraphicsImporterDrawer (qtf);
                canv.setClient (myImageFile, true);

                addWindowListener(new WindowAdapter () {
                        public void windowClosing (WindowEvent e) {
                                QTSession.close();
                                dispose();
                        }

                        public void windowClosed (WindowEvent e) {
                                System.exit(0);
                        }
                });
        }
}
```

Dave Marshall
10/4/2001

Quicktime to Java Imaging

The code in this section shows how to create a `java.awt.Image` from a QuickTime source. The `QTImageProducer` is used to produce this image's pixel data from the original QuickTime source.

The QuickTime image could come from any one of the following sources:

- an image file in a format that Java doesn't directly support but QuickTime does
- recording the drawing actions of a QDGraphics into a Pict. This can be written out to a file or presented by an `ImagePresenter` class to the `QTImageProducer` directly.
- using the services of QuickTime's sequence grabber component. A sequence grabber can be used to capture just one individual frame from a video source.

In the sample code, the user is prompted to open an image file from one of 20 or more formats that QuickTime's `GraphicsImporter` can import. The program then uses the `QTImageProducer` to create a `java.awt.Image` that is then drawn in the `paint()` method of the `Frame`. You prompt the user to select an image file and import that image into QuickTime. You then create a `GraphicsImporteDrawer` that uses the `GraphicsImporter` to draw.

This object produces pixels for the `QTImageProducer`:

```
QTFile imageFile =
QTFile.standardGetFilePreview(QTFile.kStandardQTFileTypes);
GraphicsImporter myGraphicsImporter =
     new GraphicsImporter (imageFile);
GraphicsImporterDrawer myDrawer =
     new GraphicsImporterDrawer(myGraphicsImporter);
```

You create a java.awt.Image from the pixels supplied to it by the `QTImageProducer`:

```
QDRect r = myDrawer.getDisplayBounds();
imageSize = new Dimension (r.getWidth(), r.getHeight());
QTImageProducer qtProducer = new QTImageProducer (myDrawer,
                                        imageSize);
javaImage =
   Toolkit.getDefaultToolkit().createImage(qtProducer);
```

Note that to do Java drawing, your application uses the `java.awt.Graphics.drawImage(...)` calls, which must be defined in a `paint()` method. When using a `QTCanvas` client, this detail is taken care of by the `QTCanvas` itself by establishing the `QTCanvas` `QTDrawable` client relationship, as the above code demonstrates.

In the `paint()` method of the frame, the image that we produced in the above code is drawn, using this `drawImage` call. This method will correctly resize the image to the size of the Frame.

```
public void paint (Graphics g) {
Insets i = getInsets();
Dimension d = getSize();
int width = d.width - i.left - i.right;
int height = d.height - i.top - i.bottom;
g.drawImage (javaImage, i.left, i.top, width, height, this);
}
```

This returns the size of the source image, so the `pack()` will correctly resize the frame:

```
public Dimension getPreferredSize () {
return imageSize;
}
```

Dave Marshall
10/4/2001

Image Producing

The code in this section shows how to display any QuickTime drawing object using Java's `ImageProducer` model.

The program works with the `QTImageProducer` and with Swing components. The `QTImageProducer` in this case is responsible for getting the QuickTime movie and producing the pixels for a
`java.awt.Image`. It draws the movie into its own `QDGraphics` world and then feeds the pixels to any `java.image.ImageConsumer` objects that are registered with it in a format that they are able to deal with. The Swing buttons control the movie playback.

The Swing `JComponent` is the `ImageConsumer` of the `QTImageProducer`. It will automatically repaint itself if the `QTImageProducer` source is multiframed media, such as a movie. This is a feature of the `ImageProducer` model of the Java API.

Placing a QuickTime movie within a Swing `JComponent` requires the usage of Java's image producing API. Swing is a framework that uses lightweight components. As such, a heavyweight component, such as a `QTCanvas`, is generally not added to the lightweight components. To put QuickTime content into a lightweight component
`java.awt.Image` is used to capture the pixel data generated by QuickTime. As the previous example showed, the Java image producing API is used, with the `QTImageProducer` implementing the `ImageProducer` interface used for this purpose.

We open the movie and set looping of its time base. We make a `MoviePlayer` out of the Movie to pass to the `QTImageProducer`, which takes any `QTDrawable` object as a source of pixel data. We pass in the original size of the movie to the `QTImageProducer`, which will notify the producer how big a QDGraphics it should create.

Once the `QTImageProducer` is made, we need to redraw it when each frame of the movie is drawn. To optimize this process, we install a Movie`DrawingComplete` callback. This callback notifies us when the movie draws a complete frame, and in the execute() method we redraw the `QTImageProducer`. The `QTImageProducer` redraw() method will pass on the changed pixel data to the registered `ImageConsumer` objects. The `ImageConsumer` is registered with the `QTImageProducer` when we create the `IPJComponent`, as shown in the code below.

```
OpenMovieFile openMovieFile = OpenMovieFile.asRead(movFile);
Movie m = Movie.fromFile (openMovieFile);
m.getTimeBase().setFlags (loopTimeBase);
MoviePlayer moviePlayer = new MoviePlayer (m);
QDRect r = moviePlayer.getDisplayBounds();
Dimension d = new Dimension (r.getWidth(), r.getHeight());
ip = new QTImageProducer (moviePlayer, d);

//this tells us that the movie has redrawn and
//we use this to redraw the QTImageProducer - which will
//supply more pixel data to its registered consumers

m.setDrawingCompleteProc (movieDrawingCallWhenChanged, this);
IPJComponent canv = new IPJComponent (d, ip);
pan.add("Center", canv);
```

The following is the `execute()` method of the `MovieDrawingComplete` interface. This `execute()` method is invoked whenever the movie draws. We use the `updateConsumers()` method of the `QTImageProducer` as the movie has already drawn when this callback is executed, so we only need to notify the image consumers and give them the new pixel data. If the movie hadn't drawn, then the `QTImageProducer` redraw() method would be called. This both redraws the `QTDrawable` source and updates the image consumers' pixel data:

```
public int execute (Movie m) {
try {
  ip.updateConsumers (null);
    } catch (QTException e) {

return e.errorCode();
}
return 0;
}
```

The following shows the construction of the `IPJComponent`. This `JComponent` is a Swing component and its paint() method will draw the image that is the consumer of the `QTImageProducer`. The constructor creates a java.awt.Image, the createImage() method establishing the `QTImageProducer` as the producer of pixel data for this Image. The `prepareImage()` call establishes the `IPJComponent` as the `ImageObserver` of this process. The `paint()` method uses the supplied `java.awt.Graphic drawImage()` call to draw the Image, also passing the `IPJComponent` as the `ImageObserver`. Each time the `QTImageProducer` is redrawn, it notifies its image consumers that it has more pixel data. This will, in turn, notify the `ImageObserver`, which is the `IPJComponent`, that it should repaint itself. The `paint()` method is consequently called and `drawImage()` will draw the new pixel data to the screen.

```
static class IPJComponent extends JComponent {
```

```
IPJComponent (Dimension prefSize, QTImageProducer ip) {
pSize = prefSize;
im = createImage (ip);
prepareImage (im, this);
}
private Dimension pSize;
private Image im;
public Dimension getPreferredSize() {
return pSize;
}
public void paint (Graphics g) {
g.drawImage (im, 0, 0, pSize.width, pSize.height, this);
}
// stops flicker as we have no background color to erase
public void update (Graphics g) {
paint (g);
}
}
```

Next Up Previous

Next: Using the QTImageDrawer Class **Up:** QuickTime for Java **Previous:** Quicktime to Java Imaging

Dave Marshall
10/4/2001

Using the `QTImageDrawer` Class

The `QTImageDrawer` class enables standard Java drawing commands and graphics objects to have their content rendered by QuickTime within a QuickTime graphics space (`QDGraphics`). Standard AWT `paint()` calls can be made on the Graphics object supplied to the `paint()` call`QTImageDrawer` implements the `ImageSpec` interface, you can add it to a `quicktime.app.display.Compositor` object and thus draw into the same display space as QuickTime-generated content. It also implements the `QTDrawable` interface and can thus also be a client of a `QTCanvas`. A `QTImageDrawer` object is used to convert the results of painting into a java.awt.Image object using a java.awt.Graphics object to a format that QuickTime can render.

When the java.awt.Image object is created and painted into (using the `QTImageDrawer` Paintable object) the `QTImageDrawer` uses Java's `PixelGrabber` to grab the raw pixel values that resulted from this painting. It then constructs an `ImageDescription` object that describes to Quick-Time the format, width, and height of this pixel data, and an `ImagePresenter` is used internally to render the image using QuickTime imaging services.

If the `QTImageDrawer` is used to grab the results of a single paint, then it can make optimizations on the amount of memory that it uses. A single frame is indicated by the `kSingleFrame` flag when constructing it. In this case, your application can grab the resultant image data (`getImage()`) and its description (`getDescription()`) and then discard the `QTImageDrawer` object completely. This is preferable in many situations where both performance and memory usage are a consideration:

- performance is enhanced because once the image is retrieved in this way and can be drawn very efficiently by QuickTime, it is already in a for-mat that is suitable.
- The extra memory and overhead of Java's imaging model and the translation to a format QuickTime can render can thus be discarded.

The `paint()` method of the `QTImageDrawer Paint`able object returns an array of rectangles that tells the drawer which areas of its drawing area were drawn. This optimizes the amount of copying and can greatly increase performance when composited.

Next: Visual Effects **Up:** QuickTime for Java **Previous:** Image Producing

Dave Marshall
10/4/2001

Visual Effects

QuickTime provides a wide range of sophisticated visual effects. QuickTime's visual effects architecture can be applied to visual media in a movie or can be used in real-time -- that is, applied to image sources that are not contained in a movie. The code listed here and the QuickTime for Java classes are designed with this second usage in mind.

Example code to apply effects to a movie can be in the QuickTime SDK Example code. Especially the `CompositedEffects, QTEffects` and `Transitions` demonstrations

The `QTEffect` class forms the base class for visual effects. Depending on the effect itself, you can apply visual effects over a length of time or once only.

- Effects can be applied to no source images (***i.e.***, the effect acts on a background QDGraphics object), such as the ripple effect.
- An effect can also be applied to a single source image (typically, color correction or embossing) using the `QTFilter` class.
- Finally, an effect can be applied to two sources (such as wipes and fades), using the `QTTransition` class that transitions from the source to the destination image.

The code snippets in this section show how to use QuickTime's visual effects architecture. The complete programs `CompositedEffects.java, PlayQTEffects.java` and `TransitionEffect.java` should be consulted for further examples of use of the code. The effects in the code are applied, in ***real time***, to two images. The rendering of the transitions is controlled by the user settings in the window's control panel.

Changes to any one of these three fields of the transition can affect the values of the other. Thus, changing the duration or frames per second (fps) will alter how many frames are rendered. The code updates the values in the other fields to reflect these dependencies. You set the duration of the transition if it is running using the time mode of the `QTTransition` class:

```
timeField.addActionListener (new ActionListener () {
public void actionPerformed (ActionEvent event) {
int output = ...  //get value from field
transition.setTime (output);
}
}
```

You set the number of frames the transition will take to render:

```
frameField.addActionListener (new ActionListener () {
public void actionPerformed (ActionEvent event) {
int output = ... //get value from field
transition.setFrames (output);
}
}
```

You set the number of frames per second that the transition should be rendered in:

```
fpsField.addActionListener (new ActionListener () {
public void actionPerformed (ActionEvent event) {
int output = ...//get value from field
transition.setFramesPerSecond (output);
}
}
```

You use this code snippet to make the transition run based on the settings. If the transition is profiling, some statistics about the transition are printed.

```
runEffectButton.addActionListener (new ActionListener () {
public void actionPerformed (ActionEvent event) {
try {
 transition.doTransition();
 if (transition.isProfiled()) {
    String profileString = "Transition Profile:"
        + "requestedDuration:" + transition.getTime()
        + ",actualDuration:" + transition.profileDuration()
        + ",requestedFrames:" + transition.getFrames()
        + ",framesRendered=" + transition.profileFramesRendered()
        + ",averageRenderTimePerFrame="
        + (transition.profileDuration()
        / transition.profileFramesRendered());
     System.out.println (profileString);
   }
 } catch (QTException e)
 {
 if (e.errorCode() != userCanceledErr)
       e.printStackTrace();
 }
}
}
```

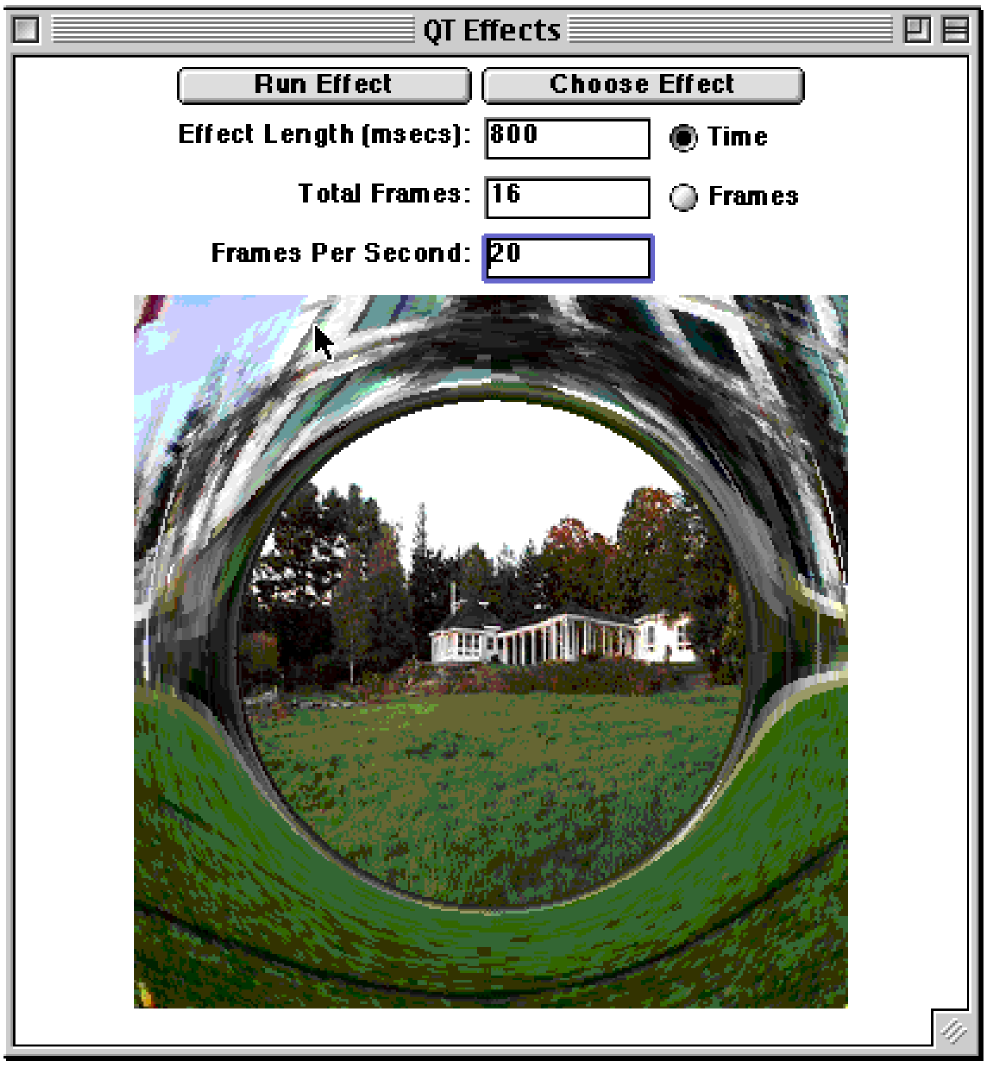
You use this code snippet to show QuickTime's Choose Effects dialog box:

```
chooseEffectButton.addActionListener (new ActionListener () {
public void actionPerformed (ActionEvent event) {
     PlayQTEffectApp.showDialog (transition);
}
}
```

The two buttons in the following code snippet can change the mode of the transition. If the transition renders using time mode, it will potentially drop frames in order to render itself as close to the specified duration as possible. If the transition is set to `doTime(false)`, it will render the currently specified number of frames as quickly as possible. This mode varies considerably from computer to computer based on the processing and video capabilities of the runtime environment.

```
frameButton.addItemListener (new ItemListener () {
public void itemStateChanged (ItemEvent event) {
transition.doTime (false);
}
});
timeButton.addItemListener (new ItemListener () {
public void itemStateChanged (ItemEvent event) {
transition.doTime (true);
}
}
```

Figs. 9.5 and 9.6 show two (of the many) effects from the program.

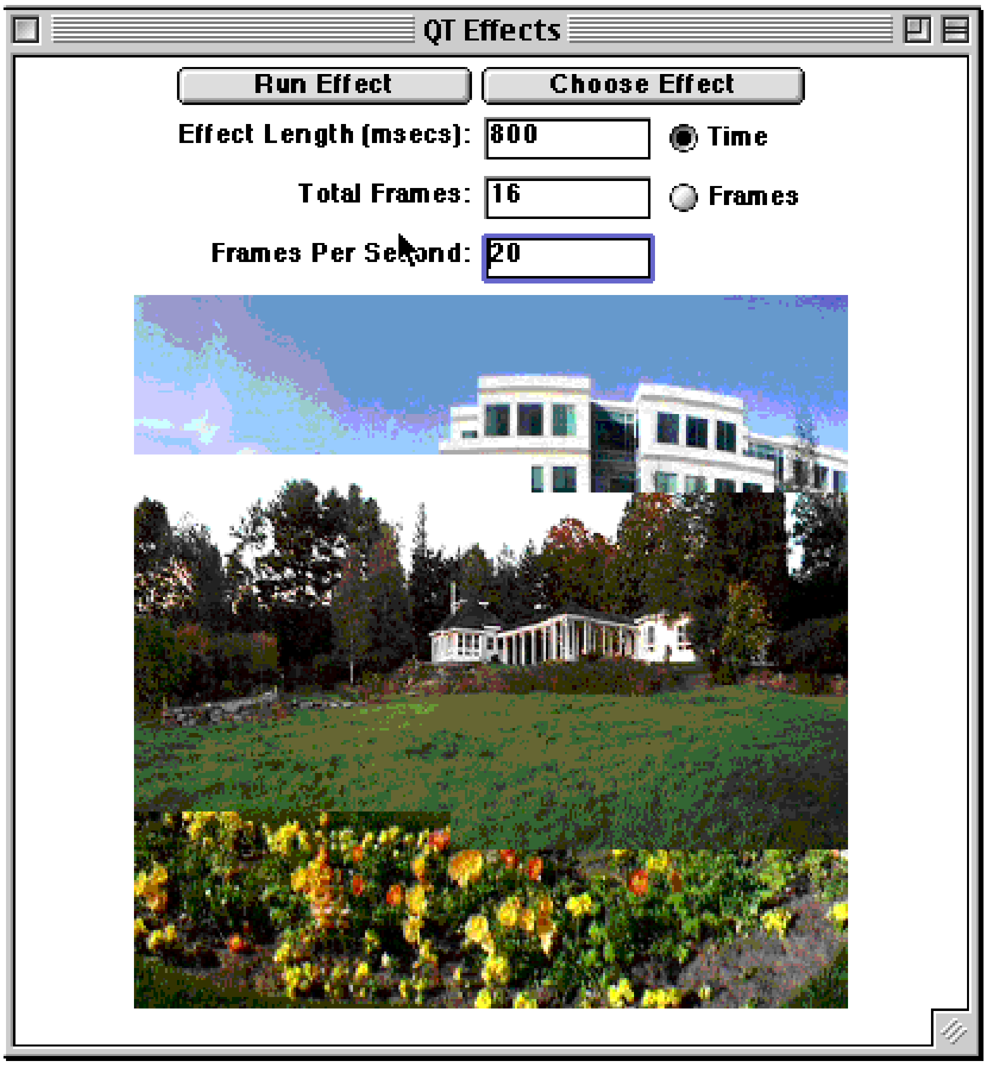


Effect Length (msecs): 800 [✓ Time] Frames

Total Frames: 16

Frames Per Second: 20

The explode effect frome the QT Effects program


QT Effects

Run Effect | Choose Effect

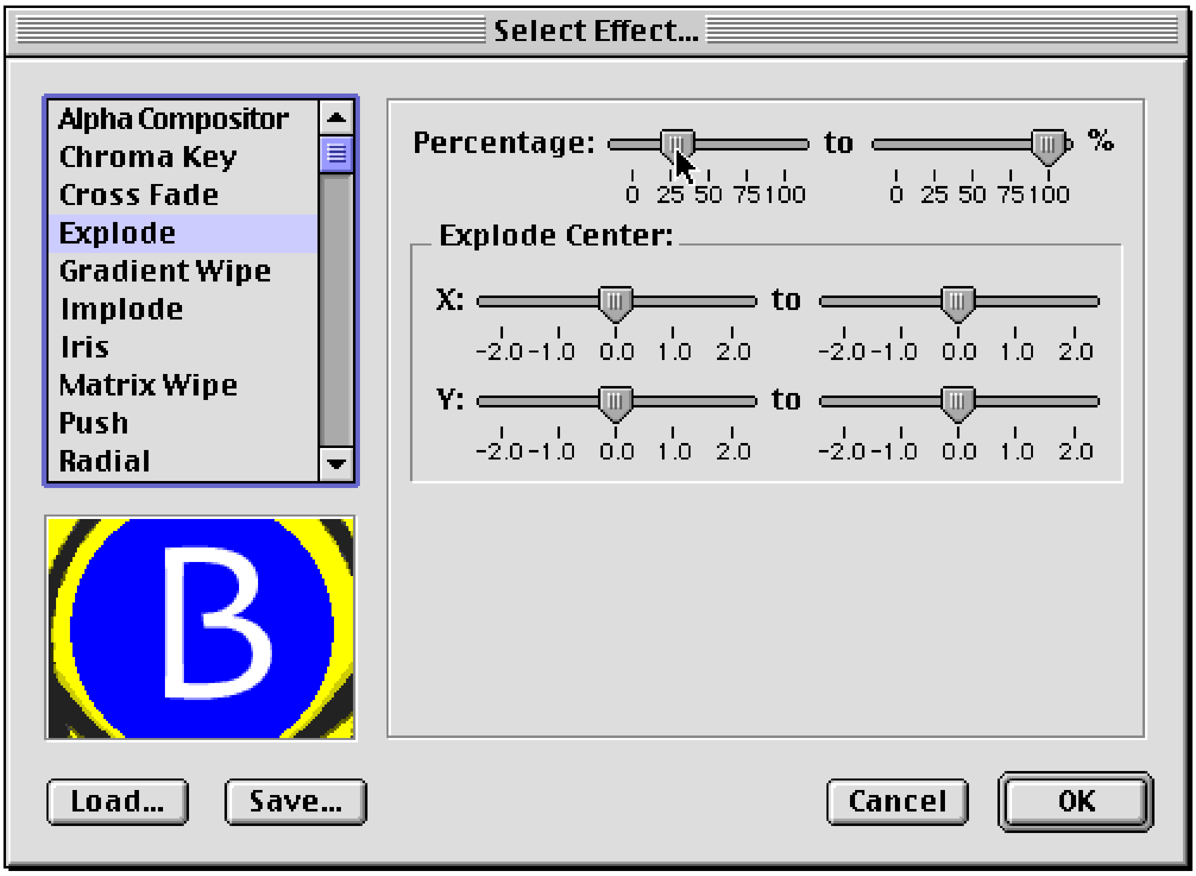
Effect Length (msecs): 800 ● Time

Total Frames: 16 ○ Frames

Frames Per Second: 20

The matrix effect frome the QT Effects program

To make an effect, you need to either build one in code or the user can choose an effect from QuickTime's Choose Effects dialog box, as shown in Fig. 9.7.



QuickTime's Choose Effects dialog box

You set up an atom container to use a SMPTE effect. Using SMPTE effects, you set the `WhatAtom` to SMPTE, ensure the endian order is `bigendian`, and use SMPTE effect number 74:

```
public AtomContainer createSMPTEEffect (int effectType,
int effectNumber) {
   AtomContainer effectSample = new AtomContainer();
  // We are using SMPTE Effects so set the what atom to smpt
  effectSample.insertChild (new Atom(kParentAtomIsContainer),
            kEffectWhatAtom,
            1,
            0,
    EndianOrder.flipNativeToBigEndian32(kWipeTransitionType));
  // We are using SMPTE effect number 74 - start at 0%, stop at 100%
  effectSample.insertChild (new Atom(kParentAtomIsContainer),
            effectType,
            1,
            0,
            EndianOrder.flipNativeToBigEndian32(effectNumber));
     return effectSample;
}
```

In this example, the dialog box is configured to show only effects that expect two sources, but it can be configured to show effects applied or requiring only a single source.

```
public static void showDialog (QTEffect ef) throws QTException {
   AtomContainer effectSample
       = ParameterDialog.showParameterDialog(new EffectsList
           (2, 2, 0), 0);
   ef.setEffect (effectSample);
}
```

You use the `doTransition()` method, which is part of the `QTTransition` class in the `quicktime.app`.image package, to set parameters to control the rendering behavior of the effect.

```
QTTransition ef = new QTTransition();
ef.setTime (800);
ef.setSourceImage (sourceImage);
ef.setDestinationImage (destImage);
ef.setEffect (createSMPTEEffect (kEffectWipe,
kRandomWipeTransitionType));
```

Your application can also directly control the rendering of each frame of an effect through setting the frame directly and redrawing the effect.

Animation and Compositing

With animation, you are working essentially with a time-based script of composited images, which simply means that the images will change over time. With compositing, you are using images from different sources and layering them and generating a composited image.

Dave Marshall
10/4/2001

The `SWCompositor` Class

The `SWCompositor` class provides the capability to compose a complex image from disparate image sources and then treat the result as a single image, which is presented to the user. It also provides a time base that controls the rendering cycle and allows your application to attach time-based behaviors or actions.

The `SWCompositor` uses the QuickTime `SpriteWorld` internally to perform its compositing tasks and its TimeBase for its timing services. All of the actual drawing of the members of a `SWCompositor` is handled through the interaction between the `SpriteWorld` and Sprite classes of QuickTime.

The `SpriteWorld` itself is wrapped by the `SWCompositor` class, and to represent the Sprite class it uses the `TwoDSprite`. The `TwoDSprite` is a presenter, that is it presents image information. The presentation of image information within the context of the `SWCompositor` `SpriteWorld` is determined by the matrix, graphics mode, layer, and visibility of the Sprite object.

To create a Sprite, you need a valid `SpriteWorld`. To create a `SpriteWorld`, you need a valid QDGraphics destination. Depending on whether a `SWCompositor` is visible, you may or may not have a valid destination QDGraphics. The interaction between the `SWCompositor` and its `TwoDSprite` presenters handles the saving and creating of `SpriteWorld` and Sprite objects -- your application does not need to deal specifically with this issue.

The `SWCompositor` object presents the functionality of the `SpriteWorld` within the context of the `QTDisplaySpace` interface. The `TwoDSprite` object wraps the QuickTime `Sprite` object, giving it a `Transformable` interface so that its visual characteristics can have matrix transformations applied to them. It also implements the `Compositable` interface, as graphics modes can be applied when a sprite's image data is rendered. You can save the current display state of an individual sprite and recreate a sprite from the information you saved; the `TwoDSpriteInfo` object is a helper class created for this purpose.

The `SWCompositor` supports two important characteristics of members. If the member implements a `DynamicImage`, this indicates to the `Compositor` that the image being presented by the member is apt to change. For example, the member is a `QTImageProducer`, in which case the member needs to create a special object -- an Invalidator -- that will invalidate the sprite that is presenting the member, so that the composite cycle will redraw that sprite, and you will see the changing image data of a movie as it plays back. This is discussed in more detail below. A further service that the `SWCompositor` renders to its new members is the handling of members that implement the Notifier interface. If a member implements this interface, the `Compositor` establishes the connection between the Notifier (the new member) and its `NotifyListener` (the new member's `TwoDSprite`). When the new member's image data is complete, it can then automatically notify its registered `NotifyListener` -- its `TwoDSprite`. This is used with `QTImageDrawer` members where the `QTImageDrawer` won't have valid image data until the Java offscreen image is created. It could also be used with images that may reside on a remote server, or interactively where the user may choose or draw an image that is part of an animation.

Next Up Previous

Next: The SWController **Up:** Animation and Compositing **Previous:** Animation and Compositing

Dave Marshall
10/4/2001

The `SWController`

The `SWController` deals with `SpriteWorldHitTest` calls on the `SpriteWorld` that are contained by the `SWCompositor`'s subclasses. By default, it performs hit-testing on the actual sprite image itself. However, your application can set the hit-test flags to support any mode of hit-testing appropriate. As a subclass of the `MouseController`, it will return any member of the `SWCompositor` that is hit only if `isWholespace()` is true. If `isWholespace`() is false, the hit sprite must be a member of the `SWController` itself.

Dave Marshall
10/4/2001

Next Up Previous
Next: Composited Effects **Up:** Animation and Compositing **Previous:** The SWController

The Compositor

The `Compositor` class is a subclass of `SWCompositor`, with its primary role being the relaxation of membership requirements of its members. The member object of a `Compositor` is only required to implement the `ImageSpec` interface, in which case, the `Compositor` creates the `TwoDSprite` that presents that image data in its display space. If a `TwoDSprite` itself is added to the `Compositor`, it is added directly as a member, since `TwoDSprite` also implements the `ImageSpec` interface.

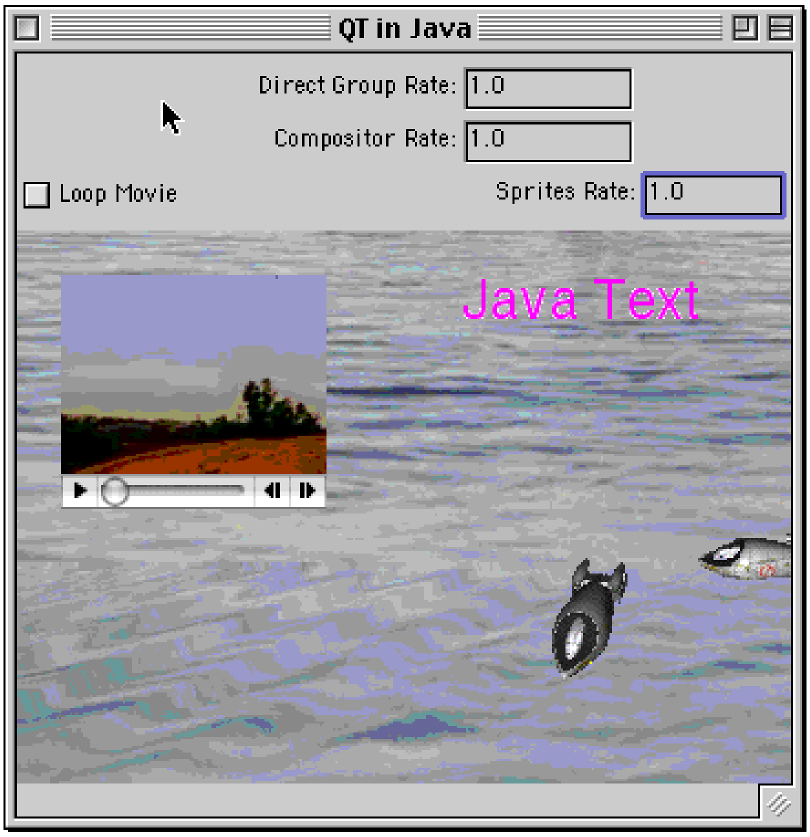
If the new member implements the `Transformable` interface, then its current matrix will be set on its `TwoDSprite` presenter. If the new member implements the `Compositable` interface, then its current `GraphicsMode` is applied to its `TwoDSprite` presenter. These actions simplify the adding of members to the `Compositor`. However, once added, your application will need to deal directly with the member's `TwoDSprite` presenter to alter the Matrix or `GraphicsMode` of the sprite.

Note that most of the functionality of the `Compositor` is within the `SWCompositor` abstract class. It is only in the membership requirements that the `Compositor` specializes the `SWCompositor` -- specifically in the creation, removal, and retrieval of a member's `TwoDSprite` presenter.

Dave Marshall
10/4/2001

Composited Effects

The example code in this section, `CompositedEffects.java` is one of demo programs in the SDK, shows the use of a `Compositor` to create a composited image out of effects, sprites, and Java text. It also shows (Fig 9.8) you how to use this as a backdrop for a QuickTime movie that draws directly to the screen. You construct a composited image containing the layering of an image file, a ripple effect, an animation and some Java text. Over this, you place a movie and its movie controller, which is drawn in front of the composited image. The `Compositor` is used to combine multiple-image objects, including a `CompositableEffect` (the ripple effect), into a single image that is then blitted on screen. Both the `Compositor` and the `QTPlayer` are added as members of the top-level `DirectGroup` client of a `QTCanvas`. The code also shows you how timing hierarchies can be established when spaces are contained within each other.



The {\tt CompositedEffects.java} Program Output

The media required for this sample code include (media files are found in the `media` folder in the SDK Demo files):

- ShipX.pct files (where X is a number indicative of a frame order)
- Water.pct
- jumps.mov

The compositing services of the `Compositor` (that is, transparent drawing, alpha blending, and so on) are not available with a `DirectGroup`. A `DirectGroup` does allow for its member objects to be layered. Thus, the movie can draw in front of the `Compositor` unheeded. It shows the embedding of a `Compositor` space in a parent `Compositor`, and then the embedding of this`Compositor` in a parent `DirectGroup` display space.

We create the parent Compositor that will contain the background image, ripple effect, Java text, and the spaceship compositor. The spaceship compositor is created similarly to preceding examples:

```
Dimension d = new Dimension (kWidth, kHeight);
QDRect r = new QDRect(d);
QDGraphics gw = new QDGraphics (r);
Compositor comp = new Compositor (gw, QDColor.green,
  new QDGraphics (r), 10, 1);
```

We add the background image, setting it to the same size as the Compositor. The ripple effect will ripple the pixels that this image draws:

```
QTFile bgFile = new QTFile(
  QTFactory.findAbsolutePath("pics/water.pct"));
GraphicsImporterDrawer if1 = new GraphicsImporterDrawer (bgFile);
if1.setDisplayBounds (r);
ImagePresenter background =
     ImagePresenter.fromGraphicsImporterDrawer (if1);
comp.addMember (background, Layerable.kBackMostLayer);
```

The ripple effect is layered to apply on top of the background image, and its bounds are set to only the top part of the compositor's display bounds. A ripple effect is applied only to what is behind it, not to sprites or text drawn in front of it. The QuickTime ripple effect codec works by moving pixels around on the destination QDGraphics that it is set to. By placing it in a Compositor, your application can control which part of an image it ripples -- in this case, the water picture that is be-hind it.

```
CompositableEffect e = new CompositableEffect ();
AtomContainer effectSample = new AtomContainer();
effectSample.insertChild (new Atom(kParentAtomIsContainer),
        kEffectWhatAtom,
        1,
        0,
  EndianOrder.flipNativeToBigEndian32(kWaterRippleCodecType));
e.setEffect (effectSample);
e.setDisplayBounds (new QDRect (0, kHeight - 100, kWidth, 100));
comp.addMember (e, 2);
```

We add the contained Compositor. Yellow is set as the background color, which is then not drawn, as we set the graphics mode of the Compositor to transparent with yellow as the transparent color.

We also add a Dragger so that members of this compositor can be dragged around when any modifier key is pressed when the mouse-Pressed event is generated. We also add a Dragger to the parent Compositor so that we can drag any of its top-level members when no modifier keys are pressed:

```
Compositor sh = new Compositor (
    new QDGraphics (new QDRect(160, 160)),
    QDColor.yellow, 8, 1);
addSprites (sh);
sh.setLocation (190, 90);
sh.setGraphicsMode (new GraphicsMode
     (transparent, QDColor.yellow));
sh.getTimer().setRate(1);
sh.addController(new SWController
    (new Dragger (
     MouseResponder.kAnyModifiersMask,
     MouseResponder.kAnyModifiers), true));
comp.addMember (sh, 1);
comp.addController(new SWController (
   new Dragger (MouseResponder.kNoModifiersMask),
   true));
```

You use the `QTImageDrawer` object using the Java-drawing APIs to draw the Java text, which is then given a transparency so that only the text characters themselves are displayed by QuickTime. Note that you set the background color to white, so the Java text appears transparent. White provides a reliable transparent background for different pixel depths.

```
myQTCanvas.setBackground (Color.white);
\begin{verbatim}

You add the Java text in front of the background image and ripples
and set its transparency to the background color of the {\tt QTCanvas}, so
that
only the text is seen.


\begin{verbatim}
QTImageDrawer qid =
  new QTImageDrawer (jt, new Dimension (110, 22),
      Redrawable.kSingleFrame);
Paintable jt = new JavaText ();
qid.setGraphicsMode
  (new GraphicsMode (transparent, QDColor.white));
qid.setLocation (200, 20);
comp.addMember (qid, 1);
```

The code here provides a good demonstration of the timing hierarchy that is built using the display spaces of the `Compositor` and `DirectGroup`. We make a `DirectGroup` as the top-level container space, adding both the containing `Compositor` as a member of this group and the movie jumps.mov. Finally, we set the rates of both the `Compositor` and the `DirectGroup` to 1 so that they are playing when the window is shown:

```
DirectGroup dg =
new DirectGroup (d, QDColor.white);
dg.addMember (comp, 2);
QTFile movieFile =
   new QTFile (
   QTFactory.findAbsolutePath ("jumps.mov"));
QTDrawable mov = QTFactory.makeDrawable (movieFile);
mov.setDisplayBounds (new QDRect(20, 20, 120, 106));
dg.addMember (mov, 1);
myQTCanvas.setClient (dg, true);
comp.getTimer().setRate(1);
dg.getTimer().setRate(1);
```

The top `DirectGroup` is the master time base for all of its members; the rate at which its time base is set (the top text box) determines the overall rate of its members. The members can have their own rates that become offset based on the rates of their parent groups. To start the program, you set the top rate to 1.

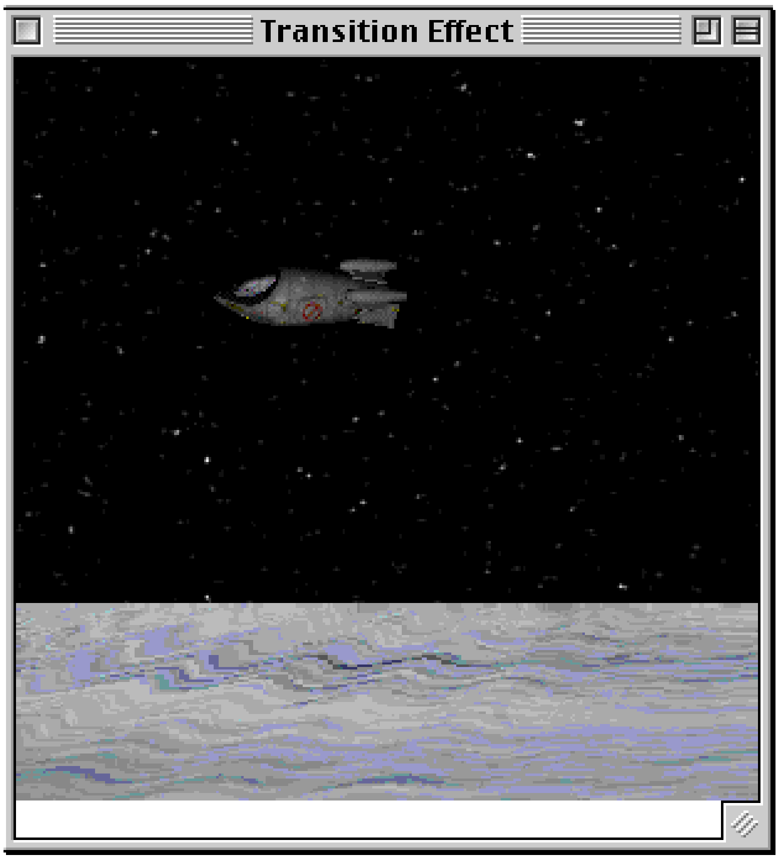
Next Up Previous

Dave Marshall
10/4/2001

Transition Effects

We end with a program, `TransitionEffect.java` that demonstrates how to use the QuickTime effects architecture and apply it to a character in an animation scene.

The code in this section shows how you can build a transition effect and apply it to a character in a realistic animation (Fig. 9.9):



Transition Effect Example

In the program, a fading effect is applied to transition to the spaceship image. The image fades in and out as per the rate and the number of frames set for the transition.

It shows the usage of effects and effects presenters. The `kCrossFadeTransitionType` effect is applied to the source and the destination images, which that makes them fade as per the number of frames set for the transition.

The `QTEffectPresent` is used to embed the `QTTransition` effect and present it to the `Compositor`, which draws it to the canvas. Note that the `QTTransition` effect cannot be directly added to the Compositor; instead, it is given to the `QTEffectPresent`, which is added to the `Compositor`. If a filter were applied, it would have the same limitations as a transition when added to a `Compositor`. It must be added using the `QTEffectPresent`.

A ripple effect is applied to the water image (in front of the water image taking up the same location), using the `CompositableEffect` class. Zero sourced effects, such as the ripple effect, can be added directly to a `Compositor`.

```
class Fader implements StdQTConstants {
Fader() throws Exception {
File file = QTFactory.findAbsolutePath ("pics/Ship.pct");
QTFile f = new QTFile (file.getAbsolutePath());
QDGraphics g = new QDGraphics (new QDRect (78, 29));
g.setBackColor (QDColor.black);
g.eraseRect(null);
ImagePresenter srcImage = ImagePresenter.fromGWorld(g);
Compositable destImage = new GraphicsImporterDrawer (f);
ef = new QTTransition ();
ef.setRedrawing(true);
ef.setSourceImage (srcImage);
ef.setDestinationImage (destImage);
ef.setDisplayBounds (new QDRect(78, 29));
ef.setEffect (createFadeEffect (kEffectBlendMode,
kCrossFadeTransitionType));
ef.setFrames(60);
ef.setCurrentFrame(0);
}
private QTTransition ef;
public QTEffectPresenter makePresenter() throws QTException {
QTEffectPresenter efPresenter = new QTEffectPresenter (ef);
return efPresenter;
}
public QTTransition getTransition () {
return ef;
AtomContainer createFadeEffect (int effectType, int effectNumber)
throws QTException {
AtomContainer effectSample = new AtomContainer();
effectSample.insertChild (new Atom(kParentAtomIsContainer),
     kEffectWhatAtom,
     1,
     0,
  EndianOrder.flipNativeToBigEndian32(kCrossFadeTransitionType));
effectSample.insertChild (new Atom(kParentAtomIsContainer),
effectType,
1,
0,
EndianOrder.flipNativeToBigEndian32(effectNumber));
return effectSample;
}
}
```

We then create the `QTEffectPresent` for the transition and add it as a member of the `Compositor`:

```
Fader fader = new Fader();
QTEffectPresenter efp = fader.makePresenter();
efp.setGraphicsMode (new GraphicsMode (blend, QDColor.gray));
efp.setLocation(80, 80);
comp.addMember (efp, 1);
comp.addController(new TransitionControl (20, 1,
fader.getTransition()));
```

The controller object implements the `TicklishController` and subclasses the `PeriodicAction` class that has a `doAction`() method, which gets invoked on every tickle call.

```
class TransitionControl extends PeriodicAction
implements TicklishController {
   TransitionControl (int scale, int period, QTTransition t) {
      super (scale , period);
      this.t = t;
   }
```

The `doAction()` call is overridden to set the current frame and redraw the `TransitionEffect`. The source and the destination images of the transition effect are swapped when the number of set frames is reached. The transition's controller then rests for a few seconds before it is awakened again and reapplied. The incoming time values to the `doAction` method (called by `PeriodicAction.tickle()`) are used to calculate the rest and transition, ensuring that if the rate of playback changes, the transition controller will react to these changes. When the transition is quiescent, we set the redrawing state of the `QTEffectPresent` to false. This ensures that when the `Compositor` invalidates this presenter it will not invalidate the sprite, as we are not currently drawing into the `QTEffectPresent`. When the transition is being applied, that is when the current frame is set, then the `isRedrawing` method will return true. The Invalidator for the `QTEffectPresenter` will then redraw the effect and invalidate its sprite presenter. Thus, this controller is also able to control the redrawing of both itself and its sprite through the use of the redrawing state and ensure that the `Compositor` only renders the sprite that presents this `QTEffectPresent` when it actually changes its pixel data -- that is, the image data of the effect's presenter.

```
protected void doAction (float er, int tm) throws QTException {
if (waiting) {
if ((er > 0 && ((startWaitTime + waitForMsecs) <= tm))
|| (er < 0 && ((startWaitTime - waitForMsecs) >= tm))) {
   waiting = false;
   t.setRedrawing(true);
} else
   return;
}

int curr_frm = t.getCurrentFrame();
curr_frm++;
t.setCurrentFrame(curr_frm);
if (curr_frm > t.getFrames()) {
  curr_frm = 0;
  t.setRedrawing(false);
  t.setCurrentFrame(curr_frm);
  t.swapImages();
  waiting = true;
  startWaitTime = tm;
}
```

Next Up Previous

Next: The Java for Quicktime **Up:** QuickTime for Java **Previous:** Composited Effects

Dave Marshall
10/4/2001

The Java for Quicktime Example Programs

Many more example programs are supplied with the Java for Quicktime. Full code listings for some of the example programs are also available here.

You are positively encouraged (***i.e*** this stuff is examinable) to investigate and implement these programs. Each example has a good Readme file that explains how it works.

The full list of example programs is as follows:

QTTestApplet
: This will do a simple test and catch any exceptions to ensure that both QuickTime and QTJava are installed properly

Music
: Simple demonstration application of the QTMusic toolbox - good first step to see that everything is installed properly. If this doesn't work there is a problem with your Java installation, your QuickTime installation or the QTJava installation, depending on the problem.

ImageFile
: use of the GraphicsImporter to display a QT image

PlayMovie
: use of the MovieController to play a QT Movie - the movie can be a local file or a URL specified by the user

DukeMovie
: Present any QT content within a Java frame with a Java rendered duke animation on top

`QTSimpleApplet`
: present ANY QT readable content with just a few lines of code. A browser version using the Java plugin and an appletviewer html page are provided

QTStreamingApplet
: An applet that lets you enter a URL to play a movie and maintain a list in the popup menu

JISApplet

: An example of reading QT data from an input stream and creating a QTCanvas client from it

DraggingSpritesApplet

: using the SpriteToolbox and java defined DragControllers to drag sprites around which is created using a movie's track

QTtoJavaImage

: Uses the QTImageProducer to produce a java.awt.Image

ImageProducing

: Display a QuickTime movie in a JComponent using the QTImageProducer

SlideShow

: This demo program shows how to use the ImageViewer object to present a sequence of images one at a time using mouse clicks from the user.

GroupDrawing

: using the group drawable object DirectGroup to present multiple QT content in a single QTCanvas - on screen. Shows the timing hierarchy that is built out of the members of the DirectGroup.

MediaPresenter

: using a DirectGroup to provide a clip region and buttons to either rotate or scale any QT presentable media.

ImageCompositing

: using the off screen Compositor to present multiple QT content in a single

CompositedEffects

: Using the compositor to composite the use of the Ripple Effect. The effect is applied to any objects it overlaps that are behind it in the compositor. The spaces ships and text are in front of it so the rippling is not applied to them. This resulting image is then added behind a movie in a DirectGroup

QTEffects

: using the QT Visual effects architecture

Bouncing Sprites

: shows using the SpriteToolbox to move sprites around and bounce them to different frames

SoundMeter

: Shows you how to meter a given band of Frequencies with the SoundMediaHandler calls. These values can be used to provide input to a spectrum display.

SoundRecord
: Using the SG classes to create a movie through recorded sound

SoundMemRecord
: Using the Sound Manager classes to record and playback sound into memory

Dragging Sprites
: using the SpriteToolbox and java defined DragControllers to drag sprites around

Create Movies
: Create a movie in Java

CurvesDemo
: Uses the Apple Curve Codec API calls that provide the functionality of QTVectors Graphics. Draws shapes and curves to demonstrate the usage of the Curve API calls. The QTVector Demo does the same thing by manipulating the raw data.

TimeCode
: Add and remove a TimeCode track to a movie - also shows how to notify QTCanvas when the application changes the size of its client

Play Tune
: Uses the TunePlayer to play a tune, save the tune to a movie and rebuild a tune from the instruments in a movie

DetachedController
: Detaching a movie from its controller - putting the movie in one QTCanvas and the MovieController in another

Custom Media
: Using the GenericMedia as a base class for an unknown to QTJava Media type Requires the user to open a VR file

JavaSprites
: "Capture" the results of drawing in java and make a QT Sprite out of it.

CreatePictFile
: Opens a file, draws it into an offscreen QDGraphics capturing the results of this drawing into a Pict. This pict is then written out as a file and can be opened and viewed in any application that reads Pict Files.

GraphicsExport

: Opens a image file, draws it into an offscreen QDGraphics , capturing the results of this drawing into a Pict file. It also provides you with a settings box where you can set the output image settings.

TweenCamera

: Adding a tween track to an existing movie with a 3D track and using the tween track to add an animated camera action.

JScriptApplet

: Using JavaScript to control playback of a QT movie. Only works in IE on Windows.

TimeSlaving

: Shows firstly scrolling java text that is scrolled based on the time value of its TimeBase. Press the movie button and the text's TimeBase is slaved to the Movie's TimeBase. The text will then scroll based on the changing time values of the movie.

Transitions

: Putting a character in an animation that fades in and out. A TicklishController is used to control the fade.

KeyboardController

: Putting a movie into the Compositor and controlling the playback of the movie with the space bar (toggle play/stop) and the arrow key - (left rate == -1, right rate = 1, up time to end of movie, down time to start of movie)

MusicMixer

: Dissecting a movie and creating a mixer representation out of the sound objects in that movie. It give you control over individual track sound and also builds a sub mixer for music instruments found in a music track.

`MovieCallbacks`

: Using the capability of QuickTime calling back into Java to notify a Java application of specific events that occur when presenting a movie

ImportExport

: An example demonstrating how to export a movie , import a media , and reference a media in a movie.

TextDemo

: An example demonstrating how to use the Text rendering capabilities of QuickDraw.

SGCapture
: Simple demonstration to display live video.

SGCapture2Disk
: Create a movie out of video and audio capture

Next Up Previous

Next: About this document ... **Up:** QuickTime for Java **Previous:** Transition Effects

Dave Marshall
10/4/2001

About this document ...

Multimedia
Module No: CM0340

The command line arguments were:
latex2html `BSC_MM_CALLER.tex`.

The translation was initiated by Dave Marshall on 10/4/2001

Dave Marshall
10/4/2001

Next Up Previous
Next: Introduction to Multimedia

Timetable for Lectures and Tutorials

- HTML version
- PDF Version

Past Exam Papers for Multimedia (CM0340) and Multimedia Systems (CMP632)

- Past Exam Papers

BSc Multimedia (CM0340) Assessed Exercises

- Assessed Exercises

Download Quicktime Java Demos

- Java for Quicktime SDK Demos (Individual files) --- Local Students ONLY
- Zipped Java for Quicktime SDK Demos --- Local Students ONLY

Java for Quicktime JavaDocs

- Java for Quicktime JavaDocs

Dave Marshall
10/4/2001

Next Up Previous
Next: About this document ...

CM0340 Multimedia Lecture and Tutorial Timetable Semester 1 2001

Dr David Marshall

Week Number	**Date/Time**	**Lecture/Tutorial**
1	Oct 1 11.10	Lecture
2	Oct 8 11.10	**Tutorial**
	Oct 9 10.00,11.00	Tutorials
3	Oct 15 11.10	Lecture
4	Oct 22 11.10	Lecture
	Oct 23 10.00,11.00	Tutorials
5	Oct 29 11.10	Lecture
6	Nov 5 11.10	Lecture
	Nov 6 10.00,11.00	Tutorials
7	Nov 12 11.10	Lecture
8	Nov 19 11.10	Lecture
	Nov 20 10.00,11.00	Tutorials
9	Nov 26 11.10	Lecture
10	Dec 3 11.10	Lecture
	Dec 4 10.00,11.00	Tutorials
11	Dec 10 11.10	Lecture

All lectures in C 2.07
All tutorials in S3.03

Dave Marshall
11/15/2001

About this document ...

CM0340 Multimedia
Lecture and Tutorial Timetable
Semester 1 2001

The command line arguments were:
latex2html `MM_Timetable2001.tex`.

The translation was initiated by Dave Marshall on 11/15/2001

Dave Marshall
11/15/2001

CM0340 Multimedia
Lecture and Tutorial Timeteble
Semester 1 2001

Dr David Marshall

Week Number	**Date/Time**	**Lecture/Tutorial**
1	Oct 1 11.10	Lecture
2	Oct 8 11.10	**Tutorial**
	Oct 9 10.00,11.00	Tutorials
3	Oct 15 11.10	Lecture
4	Oct 22 11.10	Lecture
	Oct 23 10.00,11.00	Tutorials
5	Oct 29 11.10	Lecture
6	Nov 5 11.10	Lecture
	Nov 6 10.00,11.00	Tutorials
7	Nov 12 11.10	Lecture
8	Nov 19 11.10	Lecture
	Nov 20 10.00,11.00	Tutorials
9	Nov 26 11.10	Lecture
10	Dec 3 11.10	Lecture
	Dec 4 10.00,11.00	Tutorials
11	Dec 10 11.10	Lecture

All lectures in C 2.07
All tutorials in S3.03

Multimedia Past Exam Papers

Below are past papers for the BSc Mutltimedia (CM0340) module and the MSc IGDS Elective Multimedia Systems (CMP632) Module.

Parts of the Syllabus are the same so you may wish to look at past papers from either exam. Clearly if the topic of a question is not in your respective module syllabus this can be ignored in you case.

All paper are PDF format

BSc Mutltimedia (CM0340)

- Multimedia MOCK EXAM 1999 Paper
- Multimedia BSC EXAM Academic Year 1999-2000
  - Multimedia BSC EXAM Paper 2000
  - Multimedia BSC Solutions 2000
- Smil Exam Crib Sheet

- Multimedia BSC EXAM Academic Year 2000-2001
  - Multimedia BSC EXAM Paper 2001
  - Multimedia BSC Solutions 2001

MSc IGDS Multimedia Systems (CMP632)

- Multimedia Systems Exam 1999
  - Exam paper 1999
  - Exam Solutions 1999
- Multimedia Systems Exam 2000
  - Exam paper 2000
  - Exam Solutions 2000

Multimedia
Mock Exam Questions Dec 1999

Exam paper format:

- Time Allowed: 2 Hours
- *Answer 3 Questions out of 4*
- Each Question Carries 27 Marks

1. (a) What is meant by the terms *Multimedia* and *Hypermedia*? Distinguish between these two concepts. [2]

(b) What is meant by the terms *static* media and *dynamic* media? Give two examples of each type of media. [4]

(c) What major factors affect the integration of multimedia in a multimedia presentation? [8]

(d) Describe giving suitable code fragments how you would effectively combine a video clip and an audio clip in an MHEG application. You may assume that both clips are of the same duration and must start at the same instant. [13]

2. (a) Why is file or data compression necessary for Multimedia activities?
[3]

(b) Briefly explain how the Discrete Cosine Transform Operates, and why is it so important in data compression in Multimedia applications
[10]

(c) A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

(b) Take top left pixel as the base value for the block, pixel *A*.
(c) Calculate three other transformed values by taking the difference between these (respective) pixels and pixel *A*, *i.e. B-A, C-A, D-A.*
(d) Store the base pixel and the differences as the values of the transform.

Given the above transform:

(i) What is the inverse transform? [2]
(ii) How may such a transform scheme be used to compress data?
[4]
(iii) Show how you would encode and compress the following image block:

10	20	20	25
15	25	15	20
20	25	10	20
15	20	15	25

[5]
(iv) Why is this scheme not very suitable for general image compression?
[3]

3 (a) What are the major factors to be taken into account when considering storage requirements for Multimedia Systems? [4]

(b) What is RAID technology and what advantages does it offer as a medium for the storage and delivery of large data? [4]

(c) Briefly explain the *eight* levels of RAID functionality. [8]

(d) A digital video file is 40 Mb in size. The disk subsystem has four drives and the controller is designed to support read and write onto each drive, concurrently. The digital video is stored using the *disk striping* concept. A block size of 8 Kb is used for each I/O operation.

(i) What is the performance improvement in *sequentially* reading the complete file when compared to a single drive subsystem in terms of the number of operations performed?

(ii) What is the percentage performance improvement for this system compared to a single drive system?

[11]

4 (a) Give a definition of a Multimedia Authoring System. What key features should such a system provide? [2]

(b) What Multimedia Authoring paradigms exist? Describe each paradigm briefly. [8]

(c) How would you facilitate the following application in Macromedia Director. Your answer should concentrate on the Lingo programming aspects rather than Interface issues.

Accept two numeric inputs from an appropriate cast members that enables another cast member to be moved to the inputted 2D coordinates specified by the input. The input should be checked to see if valid coordinates have been specified. [17]

CARDIFF UNIVERSITY
EXAMINATION PAPER

Academic Year:	1999/2000
Examination Period:	Autumn
Examination Paper Number:	CM0340
Examination Paper Title:	Multimedia
Duration:	2 Hours

Do not turn this page over until instructed to do so by the Senior Invigilator.

Structure of Examination Paper:

There are Three pages and Four questions.

Students to be provided with:

Answer Book(s)

Multimedia information sheets

Instructions to Students:

Attempt THREE questions.
Each question is worth 27 marks.

1. (a) What is meant by the terms *Multimedia* and *Hypermedia*?
Distinguish between these two concepts. [2]

(b) What is meant by the terms *static* media and *dynamic* media?
Give two examples of each type of media. [4]

(c) What types of functionality need to be provided in order to effectively use a wide variety of media in Multimedia applications? Your answer should briefly address how such functionality can be facilitated in general Multimedia applications.
[9]

(d) Different types of media will require different types of supporting operations to provide adequate levels of functionality. For the examples of static and dynamic media given in your answer to part 1(b) briefly discuss what operations are needed to support a wide range of multimedia applications. [12]

2. (a) Why is file or data compression highly desirable for Multimedia activities?
[2]

(b) Briefly explain, clearly identifying the differences between them, how entropy coding and transform coding techniques work for data compression. Illustrate your answer with a simple example of each type. [8]

(c) (i) Show how you would use *Huffman coding* to encode the following set of tokens:

```
BABACACADADABBCBABEBEDDABEEEBB
```

How is this message transmitted when encoded? [7]

(ii) How many bits are needed to transfer this coded message and what is its Entropy? [4]

(iii) What amendments are required to this coding technique if data is generated live or is otherwise not wholly available? Show how you could use this modified scheme by adding the tokens ADADA to the above message. [6]

3 (a) What are the major factors to be taken into account when considering storage requirements for Multimedia Systems? [4]

(b) What is RAID technology and what advantages does it offer as a medium for the storage and delivery of large data? [4]

(c) Briefly explain the *eight* levels of RAID functionality. [8]

(d) A digital video file is 40 Mb in size. The disk subsystem has four drives and the controller is designed to support read and write onto each drive, concurrently. The digital video is stored using the *disk striping* concept. A block size of 8 Kb is used for each I/O operation.

(i) What is the performance improvement in *sequentially* reading the complete file when compared to a single drive subsystem in terms of the number of operations performed?

(ii) What is the percentage performance improvement for this system compared to a single drive system? [11]

4 (a) Give a definition of a Multimedia Authoring System. What key features should such a system provide? [2]

(b) What Multimedia Authoring paradigms exist? Describe each paradigm briefly. [8]

(c) You have been asked to provide a Multimedia presentation that can support media in both English and French. You may assume that you have been given a sequence of 10 images and a single 50 second digitised audio soundtrack in both languages. Each image should be mapped over consecutive 5 second fragments of the audio. All images are of the same 500x500 pixel dimension.

Describe, giving suitable code fragments, how you would assemble such a presentation using SMIL. Your solution should cover all aspects of the SMIL presentation. [17]

Multimedia CM0340 Suplementary Exam Material Synchronized Multimedia Integration Language (SMIL) 1.0 Specification

Abstract

This document specifies version 1 of the Synchronized Multimedia Integration Language (SMIL 1.0, pronounced "smile"). SMIL allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can

1. describe the temporal behavior of the presentation
2. describe the layout of the presentation on a screen
3. associate hyperlinks with media objects

This specification is structured as follows:

- Section 1 presents the specification approach.
- Section 2 defines the "smil" element.
- Section 3 defines the elements that can be contained in the head part of a SMIL document.
- Section 4 defines the elements that can be contained in the body part of a SMIL document. In particular, this Section defines the time model used in SMIL.

Table of Contents

1 Specification Approach

SMIL documents are XML 1.0 documents [XML10]. The reader is expected to be familiar with the concepts and terms defined in XML 1.0.

This specification does not rely on particular features defined in URLs that cannot potentially be expressed using URNs. Therefore, the more generic term URI [URI] is used throughout the specification.

The syntax of SMIL documents is defined by the DTD in Section 5.2. The syntax of an attribute value that cannot

be defined using the DTD notation is defined together with the first element using an attribute that can contain the attribute value. The syntax of such attribute values is defined using the Extended Backus-Naur Form (EBNF) defined in the XML 1.0 specification.

An element definition is structured as follows: First, all attributes of the element are defined in alphabetical order. An attribute is defined in the following way: If the attribute is used by an element for the first time in the specification, the semantics of the attribute are defined. If the attribute has already been used by another element, the specification refers to the definition of the attribute in the first element that used it. The definition of element attributes is followed by the definition of any attribute values whose syntax cannot be defined using the DTD notation. The final section in an element definition specifies the element content.

2 The `smil` Element

Element Attributes

The "smil" element can have the following attribute:

id
: This attribute uniquely identifies an element within a document. Its value is an XML identifier.

Element Content

The "smil" element can contain the following children:

body
: Defined in Section 4.1

head
: Defined in Section 3.1

3 The Document Head

3.1 The `head` Element

The "head" element contains information that is not related to the temporal behavior of the presentation.

Element Attributes

The "head" element can have the following attribute:

id
: Defined in Section 2

Element Content

The "head" element can contain the following children:

layout
: Defined in Section 3.2

meta
: Defined in Section 3.4

switch
: Defined in Section 4.3

The "head" element may contain any number of "meta" elements and either a "layout" element or a "switch" element.

3.2 The `layout` Element

The "layout" element determines how the elements in the document's body are positioned on an abstract rendering surface (either visual or acoustic).

If a document contains no layout element, the positioning of the body elements is implementation-dependent.

A SMIL document can contain multiple alternative layouts by enclosing several layout elements within a "switch" element (defined in Section 4.3). This can be used for example to describe the document's layout using different layout languages.

The following example shows how CSS2 can be used as alternative to the SMIL basic layout language (defined in Section 3.3):

```
<smil>
  <head>
    <switch>
     <layout type="text/css">
        [region="r"] { top: 20px; left: 20px }
     </layout>
     <layout>
       <region id="r" top="20" left="20" />
     </layout>
   </switch>
   </head>
   <body>
     <seq>
       <img region="r" src="http://www.w3.org/test" dur="10s" />
     </seq>
   </body>
</smil>
```

(note that in this example, both layout alternatives result in the same layout)

Element Attributes

id
: Defined in Section 2

type
: This attribute specifies which layout language is used in the layout element. If the player does not understand this language, it must skip all content up until the next "</layout>" tag. The default value of the type attribute is "text/smil-basic-layout".

Element Content

If the type attribute of the layout element has the value "text/smil-basic-layout", it can contain the following elements:

region
: Defined in Section 3.3.1

root-layout
: Defined in Section 3.3.2

If the type attribute of the "layout" element has another value, the element contains character data.

3.3 SMIL Basic Layout Language

This section defines a basic layout language for SMIL. SMIL basic layout is consistent with the visual rendering model defined in CSS2, it reuses the formatting properties defined by the CSS2 specification, and newly introduces the "fit" attribute [CSS2]. The reader is expected to be familiar with the concepts and terms defined in CSS2.

SMIL basic layout only controls the layout of media object elements (defined in Section 4.2.3). It is illegal to use SMIL basic layout for other SMIL elements.

The type identifier for SMIL basic layout is "text/smil-basic-layout".

Fixed Property Values

The following stylesheet defines the values of the CSS2 properties "display" and "position" that are valid in SMIL basic layout. These property values are fixed:

```
a           {display:block}
anchor      {display:block}
animation   {display: block;
             position: absolute}
body        {display: block}
head        {display: none}
img         {display: block;
             position: absolute}
layout      {display: none}
meta        {display: none}
par         {display: block}
region      {display: none}
ref         {display: block;
             position: absolute}
root-layout {display: none}
seq         {display: block}
smil        {display: block}
switch      {display:block}
text        {display: block;
             position: absolute}
textstream  {display: block;
             position: absolute}
video       {display: block;
             position: absolute}
```

Note that as a result of these definitions, all absolutely positioned elements (animation, img, ref, text, textstream and video) are contained within a single containing block defined by the content content edge of the root element (smil).

Default Values

SMIL basic layout defines default values for all layout-related attributes. These are consistent with the initial values of the corresponding properties in CSS2.

If the author wants to select the default layout values for *all* media object elements in a document, the document must contain an empty layout element of type "text/smil-basic-layout" such as:

```
<layout type="text/smil-basic-layout"></layout>
```

3.3.1 The `region` Element

The region element controls the position, size and scaling of media object elements.

In the following example fragment, the position of a text element is set to a 5 pixel distance from the top border of the rendering window:

```
<smil>
  <head>
    <layout>
      <region id="a" top="5" />
    </layout>
  </head>
  <body>
    <text region="a" src="text.html" dur="10s" />
  </body>
</smil>
```

Element Attributes

The "region" element can have the following attributes:

background-color
: The use and definition of this attribute are identical to the "background-color" property in the CSS2 specification, except that SMIL basic layout does not require support for "system colors".
If the background-color attribute is absent, the background is transparent.

fit

This attribute specifies the behavior if the intrinsic height and width of a visual media object differ from the values specified by the height and width attributes in the "region" element. This attribute does not have a 1-1 mapping onto a CSS2 property, but can be simulated in CSS2.
This attribute can have the following values:

fill
: Scale the object's height and width independently so that the content just touches all edges of the box.

hidden
- If the intrinsic height (width) of the media object element is smaller than the height (width) defined in the "region" element, render the object starting from the top (left) edge and fill up the remaining height (width) with the background color.
- If the intrinsic height (width) of the media object element is greater than the height (width) defined in the "region" element, render the object starting from the top (left) edge until the height (width) defined in the "region" element is reached, and clip the parts of the object below (right of) the height (width).

meet
: Scale the visual media object while preserving its aspect ratio until its height or width is equal to the value specified by the height or width attributes, while none of the content is clipped. The object's left top corner is positioned at the top-left coordinates of the box, and empty space at the left or bottom is filled up with the background color.

scroll
: A scrolling mechanism should be invoked when the element's rendered contents exceed its bounds.

slice
: Scale the visual media object while preserving its aspect ratio so that its height or width are equal to the value specified by the height and width attributes while some of the content may get clipped. Depending on the exact situation, either a horizontal or a vertical slice of the visual media object is displayed. Overflow width is clipped from the right of the media object. Overflow height is clipped from the bottom of the media object.

The default value of "fill" is "hidden".

height
: The use and definition of this attribute are identical to the "height" property in the CSS2 specification. Attribute values can be "percentage" values, and a variation of the "length" values defined in CSS2. For "length" values, SMIL basic layout only supports pixel units as defined in CSS2. It allows to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2).

id
: Defined in Section 2
A region element is applied to a positionable element by setting the region attribute of the positionable element to the id value of the region.
The "id" attribute is required for "region" elements.

left
: The use and definition of this attribute are identical to the "left" property in the CSS2 specification. Attribute values have the same restrictions as the attribute values of the "height" attribute.
The default value is zero.

skip-content
: This attribute is introduced for future extensibility of SMIL (see Appendix). It is interpreted in the following two cases:
  - If a new element is introduced in a future version of SMIL, and this element allows SMIL 1.0 elements as element content, the "skip-content" attribute controls whether this content is processed by a SMIL 1.0 player.
  - If an empty element in SMIL version 1.0 becomes non-empty in a future SMIL version, the "skip-content" attribute controls whether this content is ignored by a SMIL 1.0 player, or results in a syntax error.

If the value of the "skip-content" attribute is "true", and one of the cases above apply, the content of the element is ignored. If the value is "false", the content of the element is processed.
The default value for "skip-content" is "true".

title
: This attribute offers advisory information about the element for which it is set. Values of the title attribute may be rendered by user agents in a variety of ways. For instance, visual browsers frequently display the title as a "tool tip" (a short message that appears when the pointing device pauses over an object).
It is strongly recommended that all "region" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document

without this attribute.

top
: The use and definition of this attribute are identical to the "top" property in the CSS2 specification. Attribute values have the same restrictions as the attribute values of the "height" attribute.
The default value is zero.

width
: The use and definition of this attribute are identical to the "width" property in the CSS2 specification. Attribute values have the same restrictions as the attribute values of the "height" attribute.

z-index
: The use and definition of this attribute are identical to the "z-index" property in the CSS2 specification, with the following exception:
  - If two boxes generated by elements A and B have the same stack level, then
    1. If the display of an element A starts later than the display of an element B, the box of A is stacked on top of the box of B (temporal order).
    2. If the display of the elements starts at the same time, and an element A occurs later in the SMIL document text than an element B, the box of A is stacked on top of the box of B (document tree order as defined in CSS2).

Element Content

"region" is an empty element.

3.3.2 The `root-layout` element

The "root-layout" element determines the value of the layout properties of the root element, which in turn determines the size of the viewport, e.g. the window in which the SMIL presentation is rendered.

If a document contains more than one "root-layout" element, this is an error, and the document should not be displayed.

Element Attributes

The "root-layout" element can have the following attributes:

background-color
: Defined in Section 3.3.1

height
: Defined in Section 3.3.1
Sets the height of the root element. Only length values are allowed.

id
: Defined in Section 2

skip-content
: Defined in Section 3.3.1

title
: Defined in Section 3.3.1

width
: Defined in Section 3.3.1
Sets the width of the root element. Only length values are allowed.

Element Content

"root-layout" is an empty element.

3.4 The `meta` Element

The "meta" element can be used to define properties of a document (e.g., author, expiration date, a list of key words, etc.) and assign values to those properties. Each "meta" element specifies a single property/value pair.

Element Attributes

The "meta" element can have the following attributes:

content

This attribute specifies the value of the property defined in the meta element.
The "content" attribute is required for "meta" elements.

id
: Defined in Section 2

name
: This attribute identifies the property defined in the meta element.
The "name" attribute is required for "meta" elements.

skip-content
: Defined in Section 3.3.1

The list of properties is open-ended. This specification defines the following properties:

base
: The value of this property determines the base URI for all relative URIs used in the document.

pics-label or PICS-Label
: The value of this property specifies a valid rating label for the document as defined by PICS [PICS].

title
: The value of this property contains the title of the presentation.

Element Content

"meta" is an empty element.

4 The Document Body

4.1 The body Element

The "body" element contains information that is related to the temporal and linking behavior of the document. It implicitly defines a "seq" element (defined in Section 4.2.2, see Section 4.2.4 for a definition of the temporal semantics of the "body" element).

Element Attributes

The "body" element can have the following attribute:

id
: Defined in Section 2

Element Content

The "body" element can contain the following children:

a
: Defined in Section 4.5.1

animation
: Defined in Section 4.2.3

audio
: Defined in Section 4.2.3

img
: Defined in Section 4.2.3

par
: Defined in Section 4.2.1

ref
: Defined in Section 4.2.3

seq
: Defined in Section 4.2.2

switch
: Defined in Section 4.3

text
: Defined in Section 4.2.3

textstream
: Defined in Section 4.2.3

video

Defined in Section 4.2.3

4.2 Synchronization Elements

4.2.1 The par Element

The children of a par element can overlap in time. The textual order of appearance of children in a par has no significance for the timing of their presentation.

Element Attributes

The "par" element can have the following attributes:

abstract
: A brief description of the content contained in the element.

author
: The name of the author of the content contained in the element.

begin
: This attribute specifies the time for the explicit begin of an element. See Section 4.2.4 for a definition of its semantics.
The attribute can contain the following two types of values:

delay-value
: A delay value is a clock-value measuring presentation time. Presentation time advances at the speed of the presentation. It behaves like the timecode shown on a counter of a tape-deck. It can be stopped, decreased or increased either by user actions, or by the player itself.
The semantics of a delay value depend on the element's first ancestor that is a synchronization element (i.e. ancestors that are "a" or "switch" elements are ignored):
- If this ancestor is a "par" element, the value defines a delay from the effective begin of that element (see Figure 4.1).
- If this ancestor is a "seq" element (defined in Section 4.2.2), the value defines a delay from the effective end of the first lexical predecessor that is a synchronization element (see Figure 4.2).

event-value
: The element begins when a certain event occurs (see Figure 4.3). Its value is an element-event (see Definition below).
The element generating the event must be "in scope". The set of "in scope" elements S is determined as follows:
1. Take all children from the element's first ancestor that is a synchronization element and add them to S.
2. Remove all "a" and "switch" elements from S. Add the children of all "a" elements to S, unless they are "switch" elements.

The resulting set S is the set of "in scope" elements.

copyright
: The copyright notice of the content contained in the element.

dur
: This attribute specifies the explicit duration of an element. See Section 4.2.4 for a definition of its semantics. The attribute value can be a clock value, or the string "indefinite".

end
: This attribute specifies the explicit end of an element. See Section 4.2.4 for a definition of its semantics.
The attribute can contain the same types of attribute values as the "begin" attribute.

endsync
: For a definition of the semantics of this attribute, see Section 4.2.4. The attribute can have the following values:
- first
  For a definition of the semantics of this value, see Section 4.2.4.
- id-ref
  This attribute value has the following syntax:

```
id-ref ::= "id(" id-value ")"
```
  where "id-value" must be a legal XML identifier.
  For a definition of the semantics of this value, see Section 4.2.4.
- last

For a definition of the semantics of this value, see Section 4.2.4.

The default value of "endsync" is "last".

id
    Defined in Section 2

region
    This attribute specifies an abstract rendering surface (either visual or acoustic) defined within the layout section of the document. Its value must be an XML identifier. If no rendering surface with this id is defined in the layout section, the values of the formatting properties of this element are determined by the default layout.
    The "region" attribute on "par" elements cannot be used by the basic layout language for SMIL defined in this specification. It is added for completeness, since it may be required by other layout languages.

repeat
    For a definition of the semantics of this attribute, see Section 4.2.4. The attribute value can be an integer, or the string "indefinite". The default value is 1.

system-bitrate
    Defined in Section 4.4

system-captions
    Defined in Section 4.4

system-language
    Defined in Section 4.4

system-overdub-or-caption
    Defined in Section 4.4

system-required
    Defined in Section 4.4

system-screen-size
    Defined in Section 4.4

system-screen-depth
    Defined in Section 4.4

title
    Defined in Section 3.3.1
    It is strongly recommended that all "par" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Note on Synchronization between Children

The accuracy of synchronization between the children in a parallel group is implementation-dependent. Take the example of synchronization in case of playback delays, i.e. the behavior when the "par" element contains two or more continuous media types such as audio or video, and one of them experiences a delay.
A player can show the following synchronization behaviors:

hard synchronization
    The player synchronizes the children in the "par" element to a common clock (see Figure 4.4 a)).

soft synchronization
    Each child of the "par" element has its own clock, which runs independently of the clocks of other children in the "par" element (see Figure 4.4 b)).

Attribute Values

clock value
    Clock values have the following syntax:

```
Clock-val            ::= Full-clock-val | Partial-clock-val | Timecount-val
Full-clock-val       ::= Hours ":" Minutes ":" Seconds ("." Fraction)?
Partial-clock-val    ::= Minutes ":" Seconds ("." Fraction)?
Timecount-val        ::= Timecount ("." Fraction)?
                         ("h" | "min" | "s" | "ms")? ; default is "s"
Hours                ::= 2DIGIT; any positive number
Minutes              ::= 2DIGIT; range from 00 to 59
Seconds              ::= 2DIGIT; range from 00 to 59
Fraction             ::= DIGIT+
Timecount            ::= DIGIT+
2DIGIT               ::= DIGIT DIGIT
```

```
DIGIT                ::= [0-9]
```

The following are examples of legal clock values:
- Full clock value: 02:30:03 = 2 hours, 30 minutes and 3 seconds
- Partial clock value: 02:33 = 2 minutes and 33 seconds
- Timecount values:
  3h = 3 hours
  45min = 45 minutes
  30s = 30 seconds
  5ms = 5 milliseconds

A fraction x with n digits represents the following value:

x * 1/10**n

Examples:

00.5s = 5 * 1/10 seconds = 500 milliseconds
00:00.005 = 5 * 1/1000 seconds = 5 milliseconds

element-event value
    An *element event* value specifies a particular event in a synchronization element.
    An element event has the following syntax:

```
Element-event     ::= "id(" Event-source ")(" Event ")"
Event-source      ::= Id-value
Event             ::= "begin" | Clock-val | "end"
```

The following events are defined:

begin
    This event is generated at an element's effective begin.
    Example use: `begin="id(x)(begin)"`

clock-val
    This event is generated when a clock associated with an element reaches a particular value. This clock starts at 0 at the element's effective begin. For "par" and "seq" elements, the clock gives the presentation time elapsed since the effective begin of the element. For media object elements, the semantics are implementation-dependent. The clock may either give presentation time elapsed since the effective begin, or it may give the media time of the object. The latter may differ from the presentation time that elapsed since the object's display was started e.g. due to rendering or network delays, and is the recommended approach.
    It is an error to use a clock value that exceeds the value of the effective duration of the element generating the event.

    Example use: `begin="id(x)(45s)"`

end
    This event is generated at the element's effective end.
    Example use: `begin="id(x)(end)"`

Element Content

The par element can contain the following children:

a
    Defined in Section 4.5.1

animation
    Defined in Section 4.2.3

audio
    Defined in Section 4.2.3

img
    Defined in Section 4.2.3

par

Defined in Section 4.2.1

ref
    Defined in Section 4.2.3

seq
    Defined in Section 4.2.2

switch
    Defined in Section 4.3

text
    Defined in Section 4.2.3

textstream
    Defined in Section 4.2.3

video
    Defined in Section 4.2.3

All of these elements may appear multiple times as direct children of a par element.

4.2.2 The `seq` Element

The children of a "seq" element form a temporal sequence.

Attributes

The seq element can have the following attributes:

abstract
    Defined in Section 4.2.1

author
    Defined in Section 4.2.1

begin
    Defined in Section 4.2.1

copyright
    Defined in Section 4.2.1

dur
    Defined in Section 4.2.1

end
    Defined in Section 4.2.1

id
    Defined in Section 2

region
    Defined in Section 4.2.1
    The region attribute on "seq" elements cannot be used by the basic layout language for SMIL defined in this specification. It is added for completeness, since it may be required by other layout languages.

repeat
    Defined in Section 4.2.1

system-bitrate
    Defined in Section 4.4

system-captions
    Defined in Section 4.4

system-language
    Defined in Section 4.4

system-overdub-or-caption
    Defined in Section 4.4

system-required
    Defined in Section 4.4

system-screen-size
    Defined in Section 4.4

system-screen-depth
    Defined in Section 4.4

title
    Defined in Section 3.3.1
    It is strongly recommended that all "seq" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

The seq element can contain the following children:

a
    Defined in Section 4.5.1

animation
    Defined in Section 4.2.3

audio
    Defined in Section 4.2.3

img
    Defined in Section 4.2.3

par
    Defined in Section 4.2.1

ref
    Defined in Section 4.2.3

seq
    Defined in Section 4.2.2

switch
    Defined in Section 4.3

text
    Defined in Section 4.2.3

textstream
    Defined in Section 4.2.3

video
    Defined in Section 4.2.3

4.2.3 Media Object Elements: The `ref`, `animation`, `audio`, `img`, `video`, `text` and `textstream` elements

The media object elements allow the inclusion of media objects into a SMIL presentation. Media objects are included by reference (using a URI).

There are two types of media objects: media objects with an intrinsic duration (e.g. video, audio) (also called "continuous media"), and media objects without intrinsic duration (e.g. text, image) (also called "discrete media").

Anchors and links can be attached to visual media objects, i.e. media objects rendered on a visual abstract rendering surface.

When playing back a media object, the player must not derive the exact type of the media object from the name of the media object element. Instead, it must rely solely on other sources about the type, such as type information contained in the "type" attribute, or the type information communicated by a server or the operating system.

Authors, however, should make sure that the group into which of the media object falls (animation, audio, img, video, text or textstream) is reflected in the element name. This is in order to increase the readability of the SMIL document. When in doubt about the group of a media object, authors should use the generic "ref" element.

Element Attributes

Media object elements can have the following attributes:

abstract
    Defined in Section 4.2.1

alt
    For user agents that cannot display a particular media-object, this attribute specifies alternate text. It is strongly recommended that all media object elements have an "alt" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

author
    Defined in Section 4.2.1

begin
    Defined in Section 4.2.1

clip-begin

The clip-begin attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object.
Values in the clip-begin attribute have the following syntax:

```
Clip-time-value    ::= Metric "=" ( Clock-val | Smpte-val )
Metric             ::= Smpte-type | "npt"
Smpte-type         ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val          ::= Hours ":" Minutes ":" Seconds
                       [ ":" Frames [ "." Subframes ]]
Hours              ::= 2DIGIT
Minutes            ::= 2DIGIT
Seconds            ::= 2DIGIT
Frames             ::= 2DIGIT
Subframes          ::= 2DIGIT
```

The value of this attribute consists of a metric specifier, followed by a time value whose syntax and semantics depend on the metric specifier. The following formats are allowed:

SMPTE Timestamp
: SMPTE time codes [SMPTE] can be used for frame-level access accuracy. The metric specifier can have the following values:

smpte

smpte-30-drop
: These values indicate the use of the "SMPTE 30 drop" format with 29.97 frames per second. The "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute.

smpte-25
: The "frames" field in the time specification can assume the values 0 through 24.

The time value has the format hours:minutes:seconds:frames.subframes. If the frame value is zero, it may be omitted. Subframes are measured in one-hundredth of a frame.
Examples:
`clip-begin="smpte=10:12:33:20"`

Normal Play Time
: Normal Play Time expresses time in terms of SMIL clock values. The metric specifier is "npt", and the syntax of the time value is identical to the syntax of SMIL clock values.
Examples:
`clip-begin="npt=123.45s"`
`clip-begin="npt=12:05:35.3"`

clip-end
: The clip-end attribute specifies the end of a sub-clip of a continuous media object (such as audio, video or another presentation) that should be played. It uses the same attribute value syntax as the clip-begin attribute.
If the value of the "clip-end" attribute exceeds the duration of the media object, the value is ignored, and the clip end is set equal to the effective end of the media object.

copyright
: Defined in Section 4.2.1

dur
: Defined in Section 4.2.1

end
: Defined in Section 4.2.1

fill
: For a definition of the semantics of this attribute, see Section 4.2.4. The attribute can have the values "remove" and "freeze".

id
: Defined in Section 2

longdesc
: This attribute specifies a link (URI) to a long description of a media object. This description should supplement the short description provided using the alt attribute. When the media-object has associated anchors, this attribute should provide information about the anchor's contents.

region
: Defined in Section 4.2.1

src
: The value of the src attribute is the URI of the media object.

system-bitrate

: Defined in Section 4.4

system-captions
: Defined in Section 4.4

system-language
: Defined in Section 4.4

system-overdub-or-caption
: Defined in Section 4.4

system-required
: Defined in Section 4.4

system-screen-size
: Defined in Section 4.4

system-screen-depth
: Defined in Section 4.4

title
: Defined in Section 3.3.1
It is strongly recommended that all media object elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

type
: MIME type of the media object referenced by the "src" attribute.

Element Content

Media Object Elements can contain the following element:

anchor
: Defined in Section 4.5.2

4.2.4 SMIL Time Model

4.2.4.1 Time Model Values

In the following discussion, the term "element" refers to synchronization elements only.

For each element we define the implicit, explicit, desired, and effective begin, duration, and end.

The effective begin/duration/end specify what the reader of the document will perceive.

The implicit, explicit, and desired values are auxiliary values used to define the effective values.

The rules for calculating each of these values for the elements defined in SMIL 1.0 are described in the next section.

1. Each element in SMIL has an *implicit begin* .
2. Each element can be assigned an *explicit begin* by adding a "begin" attribute to the element:

   `begin = "value of explicit-begin "`

   It is an error if the explicit begin is earlier than the implicit begin of the element.
3. Each element in SMIL has an *implicit end* .
4. Each element can be assigned an *explicit end* by adding an "end" attribute to the element:

   `end = "value of explicit-end "`

5. The *implicit duration* of an element is the difference between the implicit end and the implicit begin.
6. Each element in SMIL can be assigned an *explicit duration* by adding a "dur" attribute to the element:

   `dur = "value of explicit-duration "`

7. The *desired begin* of an element is equal to the explicit begin if one is given, otherwise the desired begin is equal to the implicit begin.
8. Each element has a *desired end* .
9. The *desired duration* of an element is the difference between the desired end and the desired begin.
10. Each element has an *effective begin* .

11. Each element has an *effective end* . (Note: the effective end of a child element can never be later than the effective end of its parent.)
12. The *effective duration* of an element is the difference between the effective end and the effective begin.

4.2.4.2 Determining Time Model Values for SMIL 1.0 Elements

This section defines how time model values are calculated for the synchronization elements of SMIL 1.0 in cases that are not covered by the rules in Section 4.2.4.1.

Determining the *implicit begin* of an element

- The implicit begin of the first child of the "body" element is when the document starts playing. When this is falls outside the scope of this document.
- The implicit begin of a child of a "par" element is equal to the effective begin of the "par" element.
- The implicit begin of the first child of a "seq" element is equal to the effective begin of the "seq" element.
- The implicit begin of any other child of a "seq" element is equal to the desired end time of the previous child of the "seq" element.

Determining the *implicit end* of an element

The first description that matches the element is the one that is to be used:

- An element with a "repeat" attribute with value "indefinite" has an implicit end immediately after its effective begin.
- An element with a "repeat" attribute with a value other than "indefinite" has an implicit end equal to the implicit end of a seq element with the stated number of copies of the element without "repeat" attribute as children.
- A media object element referring to a continuous media object has an implicit end equal to the sum of the effective begin of the element and the intrinsic duration of the media object.
- A media object element referring to a discrete media object such as text or image has an implicit end immediately after its effective begin.
- A "seq" element has an implicit end equal to the desired end of its last child.
- A "par" element has an implicit end that depends on the value of the "endsync" attribute. The implicit end is equal to the sum of the effective begin of the "par" element and the implicit duration which is derived as follows:
  - o If the value of the "endsync" attribute is "last", or if the "endsync" attribute is missing, the implicit duration of the "par" element is the maximum of the desired durations of its children.
  - o If the value of the "endsync" attribute is "first", the implicit duration of the "par" element is the minimum of the desired durations of its children.
  - o If the value of the "endsync" attribute is an id-ref, the implicit duration of the "par" element is equal to the desired duration of the child referenced by the "id-ref".

Determining the *desired end* of an element

- If the element has both an explicit duration and an explicit end, the desired end is the minimum of:
  - o the sum of the desired begin and the explicit duration; and
  - o the explicit end.
- If the element has an explicit duration but no explicit end, the desired end is the sum of the desired begin and the explicit duration.
- If the element has an explicit end but no explicit duration, the desired end is equal to the explicit end
- Otherwise, the desired end is equal to the implicit end.

Determining the *desired begin* of an element

The desired begin of an element is determined by using rule 7 in Section 4.2.4.1.

Determining the *effective begin* of an element

The *effective begin* of an element is equal to the desired begin of the element, unless the effective end of the parent element is earlier than this time, in which case the element is not shown at all.

Determining the *effective end* of an element

- The effective end of the last child of the body element is player-dependent. The effective end is at least as

late as the desired end, but whether it is any later is implementation-dependent.
- The effective end of the child of a "par" element can be derived as follows:
  - o If the child has a "fill" attribute, and the value of the "fill" attribute is "freeze", the effective end of the child element is equal to the effective end of the parent.
    The last state of the element is retained on the screen until the effective end of the element.
  - o If the child has a "fill" attribute, and the value of the "fill" attribute is "remove", the effective end of the child element is the minimum of the effective end of the parent and the desired end of the child element.
  - o If the child element has no "fill" attribute, the effective end of the child depends on whether or not the child has an explicit duration or end.
    - □ If the child has an explicit duration or end, the effective end is determined as if the element had a "fill" attribute with value "remove".
    - □ If the child has neither an explicit duration nor an explicit end, the effective end is determined as if the element had a "fill" attribute with value "freeze".
- The effective end of the last child of a "seq" element is derived in the same way as the effective end of a child of a "par" element.
- The effective end of any other child of a "seq" element can be derived as follows:
  - o If the child has a "fill" attribute, and the value of the "fill" attribute is "freeze", the effective end of the child element is equal to the effective begin of the next element
  - o If the child has a "fill" attribute, and the value of the "fill" attribute is "remove", the effective end of the child element is the minimum of the effective begin of the next element and the desired end of the next child element.
  - o If the child element has no "fill" attribute, the effective end of the child depends on whether or not the child has an explicit duration or end.
    - □ If the child has an explicit duration or end, the effective end is determined as if the element had a fill attribute with value "remove".
    - □ If the child has neither an explicit duration nor an explicit end, the effective end is determined as if the element had a fill attribute with value "freeze".

4.3 The `switch` Element

The switch element allows an author to specify a set of alternative elements from which only one acceptable element should be chosen. An element is acceptable if the element is a SMIL 1.0 element, the media-type can be decoded, and all of the test-attributes (see Section 4.4) of the element evaluate to "true".

An element is selected as follows: the player evaluates the elements in the order in which they occur in the switch element. The first acceptable element is selected at the exclusion of all other elements within the switch.

Thus, authors should order the alternatives from the most desirable to the least desirable. Furthermore, authors should place a relatively fail-safe alternative as the last item in the <switch> so that at least one item within the switch is chosen (unless this is explicitly not desired). Implementations should NOT arbitrarily pick an object within a <switch> when test-attributes for all fail.

Note that http URIs provide for content-negotiation, which may be an alternative to using the "switch" element in some cases.

Attributes

The switch element can have the following attributes:

id
: Defined in Section 2

title
: Defined in Section 3.3.1
: It is strongly recommended that all switch elements have a "title" attribute with a meaningful description Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

If the "switch" element is used as a direct or indirect child of a "body" element, it can contain the following children:

a

Defined in Section 4.5.1

animation
: Defined in Section 4.2.3

audio
: Defined in Section 4.2.3

img
: Defined in Section 4.2.3

par
: Defined in Section 4.2.1

ref
: Defined in Section 4.2.3

seq
: Defined in Section 4.2.2

switch
: Defined in Section 4.3

text
: Defined in Section 4.2.3

textstream
: Defined in Section 4.2.3

video
: Defined in Section 4.2.3

All of these elements may appear multiple times as children of a "switch" element.

If the "switch" element is used within a "head" element, it can contain the following child:

layout
: Defined in Section 3.2
: Multiple layout elements may occur within the switch element.

4.4 Test Attributes

This specification defines a list of test attributes that can be added to any synchronization element, and that test system capabilities and settings. Conceptually, these attributes represent boolean tests. When one of the test attributes specified for an element evaluates to "false", the element carrying this attribute is ignored.

Within the list below, the concept of "user preference" may show up. User preferences are usually set by the playback engine using a preferences dialog box, but this specification does not place any restrictions on how such preferences are communicated from the user to the SMIL player.

The following test attributes are defined in SMIL 1.0:

system-bitrate
: This attribute specifies the approximate bandwidth, in bits per second available to the system. The measurement of bandwidth is application specific, meaning that applications may use sophisticated measurement of end-to-end connectivity, or a simple static setting controlled by the user. In the latter case, this could for instance be used to make a choice based on the users connection to the network. Typical values for modem users would be 14400, 28800, 56000 bit/s etc. Evaluates to "true" if the available system bitrate is equal to or greater than the given value. Evaluates to "false" if the available system bitrate is less than the given value.
: The attribute can assume any integer value greater than 0. If the value exceeds an implementation-defined maximum bandwidth value, the attribute always evaluates to "false".

system-captions
: This attribute allows authors to distinguish between a redundant text equivalent of the audio portion of the presentation (intended for a audiences such as those with hearing disabilities or those learning to read who want or need this information) and text intended for a wide audience. The attribute can has the value "on" if the user has indicated a desire to see closed-captioning information, and it has the value "off" if the user has indicated that they don't wish to see such information. Evaluates to "true" if the value is "on", and evaluates to "false" if the value is "off".

system-language
: The attribute value is a comma-separated list of language names as defined in [RFC1766].

: Evaluates to "true" if one of the languages indicated by user preferences exactly equals one of the languages given in the value of this parameter, or if one of the languages indicated by

user preferences exactly equals a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-".

Evaluates to "false" otherwise.

Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags for which this tag is a prefix.

The prefix rule simply allows the use of prefix tags if this is the case.

Implementation note: When making the choice of linguistic preference available to the user, implementors should take into account the fact that users are not familiar with the details of language matching as described above, and should provide appropriate guidance. As an example, users may assume that on selecting "en-gb", they will be served any kind of English document if British English is not available. The user interface for setting user preferences should guide the user to add "en" to get the best matching behavior.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi", presented simultaneously in the original Maori and English versions, would call for:

```
<audio src="foo.rm" system-language="mi, en"/>
```

However, just because multiple languages are present within the object on which the system-language test attribute is placed, this does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin," which is clearly intended to be used by an English-literate audience. In this case, the system-language test attribute should only include "en".

Authoring note: Authors should realize that if several alternative language objects are enclosed in a "switch", and none of them matches, this may lead to situations such as a video being shown without any audio track. It is thus recommended to include a "catch-all" choice at the end of such a switch which is acceptable in all cases.

system-overdub-or-caption
: This attribute is a setting which determines if users prefer overdubbing or captioning when the option is available. The attribute can have the values "caption" and "overdub". Evaluates to "true" if the user preference matches this attribute value. Evaluates to "false" if they do not match.

system-required
: This attribute specifies the name of an extension. Evaluates to "true" if the extension is supported by the implementation, otherwise, this evaluates to "false". In a future version of SMIL, this attribute value will be an XML namespace [NAMESPACES].

system-screen-size
: Attribute values have the following syntax:
: `screen-size-val ::= screen-height"X"screen-width`
: Each of these is a pixel value, and must be an integer value greater than 0. Evaluates to "true" if the SMIL playback engine is capable of displaying a presentation of the given size. Evaluates to "false" if the SMIL playback engine is only capable of displaying a smaller presentation.

system-screen-depth
: This attribute specifies the depth of the screen color palette in bits required for displaying the element. The value must be greater than 0. Typical values are 1, 8, 24 .... Evaluates to "true" if the SMIL playback engine is capable of displaying images or video with the given color depth. Evaluates to "false" if the SMIL playback engine is only capable of displaying images or video with a smaller color depth.

4.5 Hyperlinking Elements

The link elements allows the description of navigational links between objects.

SMIL provides only for in-line link elements. Links are limited to uni-directional single-headed links (i.e. all links have exactly one source and one destination resource). All links in SMIL are actuated by the user.

Handling of Links in Embedded Documents

Due to its integrating nature, the presentation of a SMIL document may involve other (non-SMIL) applications or plug-ins. For example, a SMIL browser may use an HTML plug-in to display an embedded HTML page. Vice versa, an HTML browser may use a SMIL plug-in to display a SMIL document embedded in an HTML page.

In such presentations, links may be defined by documents at different levels and conflicts may arise. In this case, the link defined by the containing document should take precedence over the link defined by the embedded object. Note that since this might require communication between the browser and the plug-in, SMIL implementations may choose not to comply with this recommendation.

If a link is defined in an embedded SMIL document, traversal of the link affects only the embedded SMIL document.

If a link is defined in a non-SMIL document which is embedded in a SMIL document, link traversal can only affect the presentation of the embedded document and not the presentation of the containing SMIL document. This restriction may be released in future versions of SMIL.

Addressing

SMIL supports name fragment identifiers and the '#' connector. This means that SMIL supports locators as currently used in HTML (e.g. it uses locators of the form "http://foo.com/some/path#anchor1").

Linking to SMIL Fragments

A locator that points to a SMIL document may contain a fragment part (e.g. http://www.w3.org/test.smi#par1). The fragment part is an id value that identifies one of the elements within the referenced SMIL document. If a link containing a fragment part is followed, the presentation should start as if the user had fast-forwarded the presentation represented by the destination document to the effective begin of the element designated by the fragment.

The following special cases can occur:

1. The element addressed by the link has a "repeat" attribute.
    1. If the value of the "repeat" attribute is N, all N repetitions of the element are played.
    2. If the value of the "repeat" attribute is "indefinite", playback ends according to the rules defined for repeat value "indefinite".
2. The element addressed by the link is contained within another element that contains a "repeat" attribute.
    1. If the value of the "repeat" attribute is N, playback starts at the beginning of the element addressed by the link, followed by N-1 repetitions of the element containing the "repeat" attribute.
    2. If the value of the "repeat" attribute is "indefinite", playback starts at the beginning of the element addressed by the link. Playback ends according to the rules defined for repeat value "indefinite".
3. The element addressed by the link is content of a "switch" element: It is forbidden to link to elements that are the content of "switch" elements.

4.5.1 The `a` Element

The functionality of the "a" element is very similar to the functionality of the "a" element in HTML 4.0 [HTML40]. SMIL adds an attribute "show" that controls the temporal behavior of the source when the link is followed. For synchronization purposes, the "a" element is transparent, i.e. it does not influence the synchronization of its child elements. "a" elements may not be nested. An "a" element must have an href attribute.

Attributes

The "a" element can have the following attributes:

id
  Defined in Section 2
href
  This attribute contains the URI of the link's destination.
  The "href" attribute is required for "a" elements.
show
  This attribute controls the behavior of the source document containing the link when the link is followed. It can have one of the following values:
  - "replace": The current presentation is paused at its current state and is replaced by the destination

resource. If the player offers a history mechanism, the source presentation resumes from the state in which it was paused when the user returns to it.
  - "new": The presentation of the destination resource starts in a new context, not affecting the source resource.
  - "pause": The source presentation is paused at its current state, and the destination resource starts in a new context. When the display of the destination resource ends, the source presentation resumes from the state in which it was paused.

  The default value of "show" is "replace".
title
  Defined in Section 3.3.1
  It is strongly recommended that all "a" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

The "a" element can contain the following children:

animation
  Defined in Section 4.2.3
audio
  Defined in Section 4.2.3
img
  Defined in Section 4.2.3
par
  Defined in Section 4.2.1
ref
  Defined in Section 4.2.3
seq
  Defined in Section 4.2.2
switch
  Defined in Section 4.3
text
  Defined in Section 4.2.3
textstream
  Defined in Section 4.2.3
video
  Defined in Section 4.2.3

4.5.2 The `anchor` Element

The functionality of the "a" element is restricted in that it only allows associating a link with a complete media object. HTML image maps have demonstrated that it is useful to associate links with spatial subparts of an object. The anchor element realizes similar functionality for SMIL:

1. The anchor element allows associating a link destination to spatial and temporal subparts of a media object, using the "href" attribute (in contrast, the "a" element only allows associating a link with a complete media object).
2. The anchor element allows making a subpart of the media object the destination of a link, using the "id" attribute.
3. The anchor element allows breaking up an object into spatial subparts, using the "coords" attribute.
4. The anchor element allows breaking up an object into temporal subparts, using the "begin" and "end" attributes. The values of the begin and end attributes are relative to the beginning of the media object.

Attributes

The anchor element can have the following attributes:

begin
  Defined in Section 4.2.1
coords
  The value of this attribute specifies a rectangle within the display area of a visual media object. Syntax and semantics of this attribute are similar to the coords attribute in HTML image maps, when the link is

associated with a rectangular shape. The rectangle is specified by four length values: The first two values specify the coordinates of the upper left corner of the rectangle.The second two values specify the coordinates of the lower right corner of the rectangle. Coordinates are relative to the top left corner of the visual media object (see Figure 4.5). If a coordinate is specified as a percentage value, it is relative to the total width or height of the media object display area.
An attribute with an erroneous coords value is ignored (right-x smaller or equal to left-x, bottom-y smaller or equal to top-y). If the rectangle defined by the coords attribute exceeds the area covered by the media object, exceeding height and width are clipped at the borders of the media object.
Values of the coords attribute have the following syntax:

```
coords-value ::= left-x "," top-y "," right-x "," bottom-y
```

end
: Defined in Section 4.2.1

id
: Defined in Section 2

show
: Defined in Section 4.5.1

skip-content
: Defined in Section 3.3.1

title
: Defined in Section 3.3.1
It is strongly recommended that all anchor elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Multimedia BSc Exam 2000 SOLUTIONS

Setter: ADM
CheckerACJ

Additional Material: SMIL Language Description Sheet

Answer 3 Questions out of 4

1. (a) *What is meant by the terms Multimedia and Hypermedia? Distinguish between these two concepts.*

Multimedia ---- An Application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.

Hypermedia --- An application which uses associative relationships among information contained within multiple media data for the purpose of facilitating access to, and manipulation of, the information encapsulated by the data.

2 MARKS ---- BOOKWORK

(b) What is meant by the terms static media and dynamic media? Give two examples of each type of media.

Static Media – does not change over time, e.g. text, graphics

Dynamic Media --- Time dependent (Temporal), e.g. Video, sound, animation.

4 MARKS --- BOOKWORK

(c) What issues of functionality need to be provided in order to effectively use a wide variety of media in Multimedia applications? Your answer should briefly address how such functionality can facilitated in general Multimedia applications.

The following functionality should be provided:

- Digital Representation of Media --- Many formats for many media
- Capture: Digitisation of Media --- special Hardware/Software
- Creation and editing --- assemble media and alter it
- Storage Requirements --- significant for multimedia
- Compression --- related to above and below, ie can save on storage but can hinder retrieval
- Structuring and retrieval methods of media --- simple to advanced DataBase Storage

- Display or Playback methods --- effect of retrieval must view data
- Media Synchronisation --- display multimedia as it is intended

9 MARKS --- BOOKWORK

(d) Different types of media will require different types of supporting operations to provide adequate levels of functionality. For the examples of static and dynamic media given in your answer to part 1(b) briefly discuss what operations are need to support a wide range of multimedia applications.

A selection of the items below is reuired for good marks NOT ALL. Other Solns Possible?

Typical Range of operations required for common media

<u>Text</u>: Editing
Formatting
Sorting
Indexing
Searching
Encrypting
ABOVE REQUIRE: :
Character Manipulation
String Manipulation

<u>Audio</u>: Audio Editing
Synchronisation
Conversion/Translation
Filtering/ Sound Enhancing Operators
Compression
Searching
Indexing
ABOVE REQUIRE: :
Sample Manipulation
Waveform Manipulation

<u>Graphics</u>: Graphic primitive Editing
Shading
Mapping
Lighting
Viewing
Rendering
Searching
Indexing
ABOVE REQUIRE: :
Primitive Manipulation
Structural/Group Manipulation

Image:	Pixel operations
	Geometric Operations
	Filtering
	Conversion
	Indexing
	Compression
	Searching
Animation:	Primitive/Group Editing
	Structural Editing
	Rendering
	Synchronistaion
	Searching
	Indexing
Video:	Pixel Operations
	Frame Operations
	Editing
	Synchronisation
	Conversion
	Mixing
	Indexing
	Searching
	Video Effects/Filtering

12 MARKS --- UNSEEN

2. (a) Why is file or data compression necessary for Multimedia activities?

Multimedia files are very large therefore for storage, file transfer etc. file sizes need to be reduced. Text and other files may also be encoded/compressed for email and other applications.

2 MARKS --- BOOKWORK

(b) *Briefly explain, clearly identifying the differences between them, how entropy coding and transform coding techniques work for data compression. Illustrate your answer with a simple example of each type.*

Compression can be categorised in two broad ways:

Lossless Compression
-- where data is compressed and can be reconstituted (uncompressed) without loss of detail or information. These are referred to as bit-preserving or reversible compression systems also.
Lossy Compression
-- where the aim is to obtain the best possible fidelity for a given bit-rate or minimizing the bit-rate to achieve a given fidelity measure. Video and audio compression techniques are most suited to this form of compression.

Lossless compression frequently involves some form of entropy encoding and are based in information theoretic techniques

Lossy compression use source encoding techniques that may involve transform encoding, differential encoding or vector quantisation.

ENTROPY METHODS:

The entropy of an information source S is defined as:

$H(S) = SUM_I (P_I \, Log_2 \, (1/P_I)$

where P_I is the probability that symbol S_I in S will occur.

$Log_2 \, (1/P_I)$ indicates the amount of information contained in S_I, i.e., the number of bits needed to code S_I.

Encoding for the Shannon-Fano Algorithm:

A top-down approach

1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.

2. Recursively divide into two parts, each with approx. same number of counts.

(Huffman algorithm also valid indicated below)

A simple transform coding example

A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. Take top left pixel as the base value for the block, pixel A.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. B-A, C-A, D-A.
3. Store the base pixel and the differences as the values of the transform.

Given the above we can easily for the forward transform:

and the inverse transform is trivial

The above transform scheme may be used to compress data by exploiting redundancy in the data:

Any Redundancy in the data has been transformed to values, Xi. So We can compress the data by using fewer bits to represent the differences. I.e if we
use 8 bits per pixel then the 2x2 block uses 32 bits/ If we keep 8 bits for the base pixel, X0, and assign 4 bits for each difference then we only use 20 bits.
Which is better than an average 5 bits/pixel

8 MARKS --- BOOKWORK

(c) (*i) Show how you would use Huffman coding to encode the following set of tokens:*

BABACACADADABBCBABEBEDDABEEEBB

How is this message transmitted when encoded?

The Huffman algorithm is now briefly summarised:

1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).

2. Repeat until the OPEN list has only one node left:

(a) From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.

(b) Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.

(c) Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.

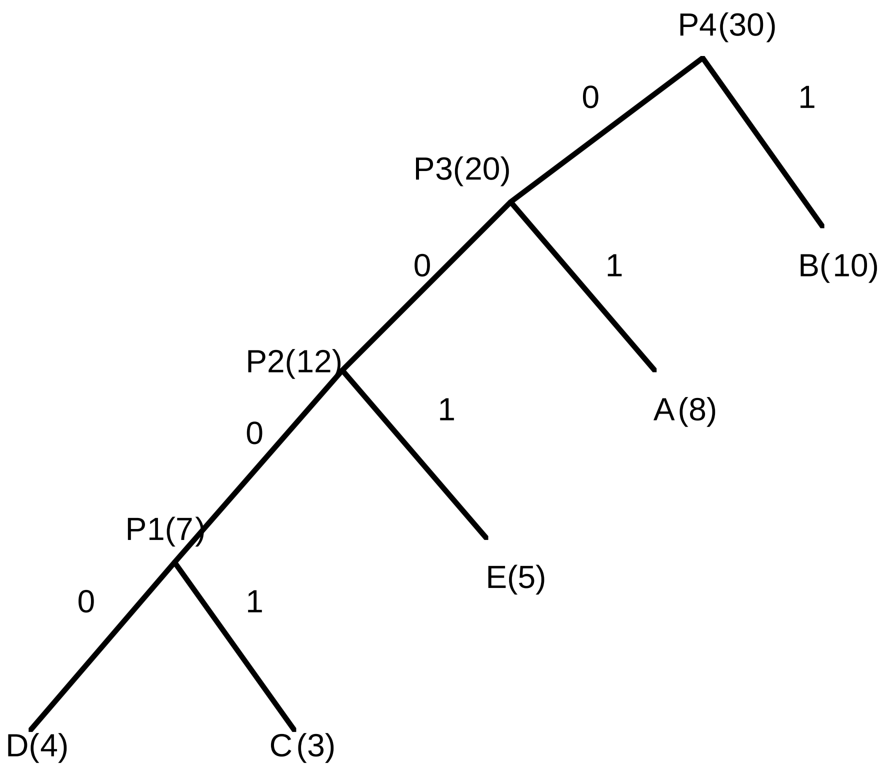
Symbol	Count	OPEN (1)	OPEN (2)	OPEN (3)
A	8			20
B	10			
C	3	7	12	-
D	4	-		
E	5		-	
Total	30			

- indicate merge node with other node with number in column

Finished Huffman Tree:

Symbol	Code
A	01
B	1
C	0001
D	0000
E	001

How is this message transmitted when encoded?



Send code book and then bit code for each symbol.

7 Marks --- UNSEEN

(ii) How many bits are needed transfer this coded message and what is its Entropy?

Symbol	Count	Subtotal # of bits
A	8	16
B	10	10
C	3	13
D	4	16
E	5	15

Total Number bits (excluding code book) = 70

Entropy = 70/30 = 2.3333

4 MARKS --- UNSEEN

(iii) *What amendments are required to this coding technique if data is generated live or is otherwise not wholly available? Show how you could use this modified scheme by adding the tokens* ADADA *to the above message.*

Adaptive method needed:

Basic idea (encoding)

```
Initialize_model();
 while ((c = getc (input)) != eof)
   {
     encode (c, output);
     update_model (c);
   }
```

So encode message as before:

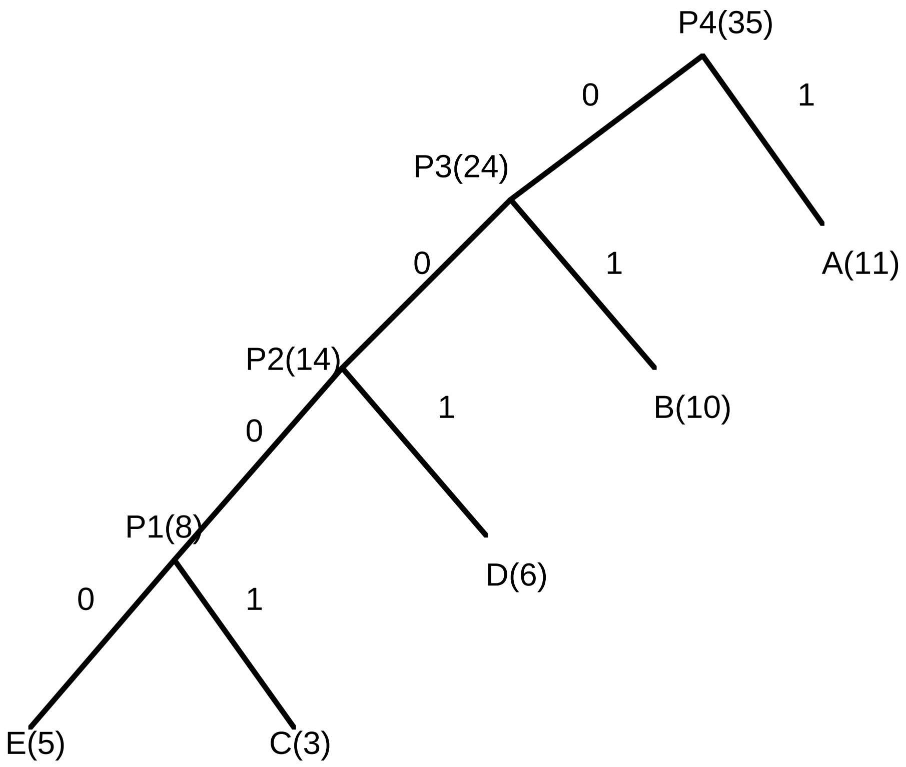
A= 01 D = 0000

So addd stream:

01000001000001

Modify Tree:

Symbol	Count	OPEN (1)	OPEN (2)	OPEN (3)
A	11			
B	10			24
C	3	8	14	-
D	6		-	
E	5	-		



6 Marks --- UNSEEN

3 (a) What are the major factors to be taken into account when considering what storage requirements are necessary for Multimedia Systems?

Major factors:

- Large volume of date
- Real time delivery
- Data format
- Storage Medium
- Retrieval mechanisms

4 MARKS --- Unseen/applied bookwork

(b) What is RAID technology and what advantages does it offer as a medium for the storage and delivery of large data?

RAID --- Redundant Array of Inexpensive Disks

Offers:

- Affordable alternative to mass storage
- High throughput and reliability

RAID System:

- Set of disk drives viewed by user as one or more logical drives
- Data may be distributed across drives
- Redundancy added in order to allow for disk failure

4 MARKS --- BOOKWORK

(c) Briefly explain the eight levels of RAID functionality .

Level 0 – Disk Striping --- distributing data across multiple drives
Level 1 – Disk Mirroring --- Fault tolerancing
Level 2 – Bit Interleaving and HEC Parity
Level 3 - Bit Interleaving with XOR Parity
Level 4 – Block Interleaving with XOR Parity
Level 5 - Block Interleaving with Parity Distribution
Level 6 – Fault Tolerant System --- Error recovery
Level 7 – Heterogeneuos System --- Fast access across whole system

8 MARKS --- BOOKWORK

(d) A digital video file is 40 Mb in size. The disk subsystem has four drives and the controller is designed to support read and write onto each drive, concurrently. The digital video stored using the disk striping concept. A block size of 8 Kb is used for each I/O operation.

(i) What is the performance improvement in sequentially reading the complete file when compared to a single drive subsystem in terms of the number of operations performed?

We have 5120 segments to write to RAID disks. Given 4 disks we have 1280 actual I/Os to perform

On 1 drive we clearly have 5120 operations to perform.

(ii) What is the percentage performance improvement expressed as the number of physical I/O operations to be executed in on the RAID and single drive systems?

The improvement is
(5120 – 1280)/1280*100 = 300%. Obvious given 4 concurrent drives and RAID!!

11 MARKS --- UNSEEN

4 (a) Give a definition of a Multimedia Authoring System. What key features should such a system provide?

An Authoring System is a program which has pre-programmed elements for the development of interactive multimedia software titles.
Authoring systems vary widely in orientation, capabilities, and learning curve.

There is no such thing (at this time) as a completely point-and-click automated authoring system; some knowledge of heuristic thinking and algorithm design is necessary.

Authoring is basically just a speeded-up form of programming --- VISUAL PROGRAMMING; you don't need to know the intricacies of a programming language, or worse, an API, but you do need to understand how programs work.

2 MARKS ---- BOOKWORK

(b) *What Multimedia Authoring paradigms exist? Describe each paradigm briefly.*

There are various paradigms, including:

Scripting Language

The Scripting paradigm is the authoring method closest in form to traditional programming. The paradigm is that of a programming language, which specifies (by filename) multimedia elements, sequencing, hotspots, synchronization, etc. A powerful, object-oriented scripting language is usually the centerpiece of such a system; in-program editing of elements (still graphics, video, audio, etc.) tends to be minimal or non-existent. Scripting languages do vary; check out how much the language is object-based or object-oriented. The scripting paradigm tends to be longer in development time (it takes longer to code an individual interaction), but generally more powerful interactivity is possible. Since most Scripting languages are interpreted, instead of compiled, the runtime speed gains over other authoring methods are minimal.

The media handling can vary widely; check out your system with your contributing package formats carefully. The Apple's HyperTalk for HyperCard, Assymetrix's

OpenScript for ToolBook and Lingo scripting language of Macromedia Director are examples of a Multimedia scripting language.

Here is an example lingo script to jump to a frame

```
global gNavSprite
on exitFrame
```

```
go the frame
play sprite gNavSprite
end
```

<u>Iconic/Flow Control</u>

This tends to be the speediest (in development time) authoring style; it is best suited for rapid prototyping and short-development time projects. Many of these tools are also optimized for developing Computer-Based Training (CBT). The core of the paradigm is the Icon Palette, containing the possible functions/interactions of a program, and the Flow Line, which shows the actual links between the icons. These programs tend to be the slowest runtimes, because each interaction carries with it all of its possible permutations; the higher end packages, such as Authorware or IconAuthor, are extremely powerful and suffer least from runtime speed problems.

<u>Frame</u>

The Frame paradigm is similar to the Iconic/Flow Control paradigm in that it usually incorporates an icon palette; however, the links drawn between icons are conceptual and do not always represent the actual flow of the program. This is a very fast development system, but requires a good auto-debugging function, as it is visually un-debuggable. The best of these have bundled compiled-language scripting, such as Quest (whose scripting language is C) or Apple Media Kit.

<u>Card/Scripting</u>

The Card/Scripting paradigm provides a great deal of power (via the incorporated scripting language) but suffers from the index-card structure. It is excellently suitedfor Hypertext applications, and supremely suited for navigation intensive (a la Cyan's "MYST" game) applications. Such programs are easily extensible via XCMDs andDLLs; they are widely used for shareware applications. The best applications allow all objects (including individual graphic elements) to be scripted; many entertainment applications are prototyped in a card/scripting system prior to compiled-language coding.

<u>Cast/Score/Scripting</u>

The Cast/Score/Scripting paradigm uses a music score as its primary authoring metaphor; the synchronous elements are shown in various horizontal tracks withsimultaneity shown via the vertical columns. The true power of this metaphor lies in the ability to script the behavior of each of the cast members. The most popularmember of this paradigm is Director, which is used in the creation of many commercial applications. These programs are best suited for animation-intensive orsynchronized media applications; they are easily extensible to handle other functions (such as hypertext) via XOBJs, XCMDs, and DLLs.

Macromedia Director uses this .

Hierarchical Object

The Hierarchical Object paradigm uses a object metaphor (like OOP) which is visually represented by embedded objects and iconic properties. Although the learning curve is non-trivial, the visual representation of objects can make very complicated constructions possible.

Hypermedia Linkage

The Hypermedia Linkage paradigm is similar to the Frame paradigm in that it shows conceptual links between elements; however, it lacks the Frame paradigm's visual linkage metaphor.

Tagging

The Tagging paradigm uses tags in text files (for instance, SGML/HTML, SMIL (Synchronised Media Integration Language), VRML, 3DML and WinHelp) to link pages, provide interactivity and integrate multimedia elements.

8 Marks --- BOOKWORK

(c) *You have been asked to provide a Multimedia presentation that can support media in both English and French. You may assume that you have been given a sequence of 10 images and a single 50 second digitised audio soundtrack in both languages. Each Image should be mapped over consecutive 5 second fragments of the audio. All Images are of the same 500x500 pixel dimension.*

Describe, giving suitable code fragments, how you would assemble such a presentation using SMIL. Your solution should cover all aspects of the SMIL presentation

```
<smil>
          <head>
           <layout>
            <root-layout height="500" width="500" background-
color="#000000" title="MultiLingual"/>
            <region id="image1" width="500" height="500" top="0"
left="0" background-color="#000000" z-index="1" />
            <region id="image2" width="500" height="500" top="0"
left="0" background-color="#000000" z-index="1" />
……….
          </layout>
           </head>

<body>
     <par>
           <switch>

            <!-- English only -->
```

```
               < audio system-language="en" src ="english.au" />

                  <!-- French only -->
                  <audio system-language="fr" src ="francais.au" />
              </switch>

<seq>
                <img src="image1.jpg" region="image1" begin="0.00s"
dur="5.00s" />
                <img src="image2.jpg" region="image2" begin="5.00s"
dur="5.00s" />

…….

</seq>
</par>

</body>
</smil>
```

17 Marks ---- UNSEEN

Multimedia Systems Suplementary Exam Material
Synchronized Multimedia Integration Language (SMIL) 1.0 Specification

Abstract

This document specifies version 1 of the Synchronized Multimedia Integration Language (SMIL 1.0, pronounced "smile"). SMIL allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can

1. describe the temporal behavior of the presentation
2. describe the layout of the presentation on a screen
3. associate hyperlinks with media objects

This specification is structured as follows:

- Section 1 presents the specification approach.
- Section 2 defines the "smil" element.
- Section 3 defines the elements that can be contained in the head part of a SMIL document.
- Section 4 defines the elements that can be contained in the body part of a SMIL document. In particular, this Section defines the time model used in SMIL.

Table of Contents

1 Specification Approach

SMIL documents are XML 1.0 documents [XML10]. The reader is expected to be familiar with the concepts and terms defined in XML 1.0.

This specification does not rely on particular features defined in URLs that cannot potentially be expressed using URNs. Therefore, the more generic term URI [URI] is used throughout the specification.

The syntax of SMIL documents is defined by the DTD in Section 5.2. The syntax of an attribute value that cannot be defined using the DTD notation is defined together with the first element using an attribute that can contain the attribute value. The syntax of such attribute values is defined using the Extended Backus-Naur Form (EBNF) defined in the XML 1.0 specification.

An element definition is structured as follows: First, all attributes of the element are defined in alphabetical order. An attribute is defined in the following way: If the attribute is used by an element for the first time in the specification, the semantics of the attribute are defined. If the attribute has already been used by another element, the specification refers to the definition of the attribute in the first element that used it. The definition of element attributes is followed by the definition of any attribute values whose syntax cannot be defined using the DTD notation. The final section in an element definition specifies the element content.

2 The `smil` Element

Element Attributes

The "smil" element can have the following attribute:

id
: This attribute uniquely identifies an element within a document. Its value is an XML identifier.

Element Content

The "smil" element can contain the following children:

body
: Defined in Section 4.1

head
: Defined in Section 3.1

3 The Document Head

3.1 The `head` Element

The "head" element contains information that is not related to the temporal behavior of the presentation.

Element Attributes

The "head" element can have the following attribute:

id
    Defined in Section 2

Element Content

The "head" element can contain the following children:

layout
    Defined in Section 3.2
meta
    Defined in Section 3.4
switch
    Defined in Section 4.3

The "head" element may contain any number of "meta" elements and either a "layout" element or a "switch" element.

3.2 The `layout` Element

The "layout" element determines how the elements in the document's body are positioned on an abstract rendering surface (either visual or acoustic).

If a document contains no layout element, the positioning of the body elements is implementation-dependent.

A SMIL document can contain multiple alternative layouts by enclosing several layout elements within a "switch" element (defined in Section 4.3). This can be used for example to describe the document's layout using different layout languages.

The following example shows how CSS2 can be used as alternative to the SMIL basic layout language (defined in Section 3.3):

```
<smil>
  <head>
    <switch>
     <layout type="text/css">
        [region="r"] { top: 20px; left: 20px }
     </layout>
     <layout>
```

```
          <region id="r" top="20" left="20" />
        </layout>
      </switch>
      </head>
      <body>
        <seq>
          <img region="r" src="http://www.w3.org/test" dur="10s" />
        </seq>
      </body>
</smil>
```

(note that in this example, both layout alternatives result in the same layout)

Element Attributes

id
: Defined in Section 2

type
: This attribute specifies which layout language is used in the layout element. If the player does not understand this language, it must skip all content up until the next "</layout>" tag. The default value of the type attribute is "text/smil-basic-layout".

Element Content

If the type attribute of the layout element has the value "text/smil-basic-layout", it can contain the following elements:

region
: Defined in Section 3.3.1

root-layout
: Defined in Section 3.3.2

If the type attribute of the "layout" element has another value, the element contains character data.

3.3 SMIL Basic Layout Language

This section defines a basic layout language for SMIL. SMIL basic layout is consistent with the visual rendering model defined in CSS2, it reuses the formatting properties defined by the CSS2 specification, and newly introduces the "fit" attribute [CSS2]. The reader is expected to be familiar with the concepts and terms defined in CSS2.

SMIL basic layout only controls the layout of media object elements (defined in Section 4.2.3). It is illegal to use SMIL basic layout for other SMIL elements.

The type identifier for SMIL basic layout is "text/smil-basic-layout".

Fixed Property Values

The following stylesheet defines the values of the CSS2 properties "display" and "position" that are valid in SMIL basic layout. These property values are fixed:

```
a            {display:block}
anchor       {display:block}
animation    {display: block;
              position: absolute}
body         {display: block}
head         {display: none}
img          {display: block;
              position: absolute}
layout       {display: none}
meta         {display: none}
par          {display: block}
region       {display: none}
ref     {display: block;
              position: absolute}
root-layout {display: none}
seq          {display: block}
smil         {display: block}
switch       {display:block}
text         {display: block;
              position: absolute}
textstream  {display: block;
              position: absolute}
video        {display: block;
              position: absolute}
```

Note that as a result of these definitions, all absolutely positioned elements (animation, img, ref, text, textstream and video) are contained within a single containing block defined by the content content edge of the root element (smil).

Default Values

SMIL basic layout defines default values for all layout-related attributes. These are consistent with the initial values of the corresponding properties in CSS2.

If the author wants to select the default layout values for *all* media object elements in a document, the document must contain an empty layout element of type "text/smil-basic-layout" such as:

```
<layout type="text/smil-basic-layout"></layout>
```

3.3.1 The `region` Element

The region element controls the position, size and scaling of media object elements.

In the following example fragment, the position of a text element is set to a 5 pixel distance from the top border of the rendering window:

```
<smil>
  <head>
    <layout>
      <region id="a" top="5" />
    </layout>
  </head>
  <body>
    <text region="a" src="text.html" dur="10s" />
  </body>
</smil>
```

Element Attributes

The "region" element can have the following attributes:

background-color
: The use and definition of this attribute are identical to the "background-color" property in the CSS2 specification, except that SMIL basic layout does not require support for "system colors".
If the background-color attribute is absent, the background is transparent.

fit
: This attribute specifies the behavior if the intrinsic height and width of a visual media object differ from the values specified by the height and width attributes in the "region" element. This attribute does not have a 1-1 mapping onto a CSS2 property, but can be simulated in CSS2.
This attribute can have the following values:

    fill
    : Scale the object's height and width independently so that the content just touches all edges of the box.

    hidden
    - If the intrinsic height (width) of the media object element is smaller than the height (width) defined in the "region" element, render the object starting from the top (left) edge and fill up the remaining height (width) with the background color.
    - If the intrinsic height (width) of the media object element is greater than the height (width) defined in the "region" element, render the object starting from the top (left) edge until the height (width) defined in the "region" element is reached, and clip the parts of the object below (right of) the height (width).

    meet
    : Scale the visual media object while preserving its aspect ratio until its height or width is equal to the value specified by the height or width attributes, while none of the content is clipped. The object's left top corner is positioned at the top-left coordinates of the box, and empty space at the left or bottom is filled up with the background color.

    scroll
    : A scrolling mechanism should be invoked when the element's rendered contents exceed its bounds.

    slice
    : Scale the visual media object while preserving its aspect ratio so that its height or width are equal to the value specified by the height and width attributes while some of the content may get clipped. Depending on the exact situation, either a horizontal or a vertical slice of the visual media object is displayed. Overflow width is clipped from the right of the media object. Overflow height is clipped from the bottom of the media object.

    The default value of "fill" is "hidden".

height
: The use and definition of this attribute are identical to the "height" property in the CSS2 specification. Attribute values can be "percentage" values, and a variation of the "length" values defined in CSS2. For

"length" values, SMIL basic layout only supports pixel units as defined in CSS2. It allows to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2).

id

Defined in Section 2
A region element is applied to a positionable element by setting the region attribute of the positionable element to the id value of the region.
The "id" attribute is required for "region" elements.

left

The use and definition of this attribute are identical to the "left" property in the CSS2 specification. Attribute values have the same restrictions as the attribute values of the "height" attribute.
The default value is zero.

skip-content

This attribute is introduced for future extensibility of SMIL (see Appendix). It is interpreted in the following two cases:

- If a new element is introduced in a future version of SMIL, and this element allows SMIL 1.0 elements as element content, the "skip-content" attribute controls whether this content is processed by a SMIL 1.0 player.
- If an empty element in SMIL version 1.0 becomes non-empty in a future SMIL version, the "skip-content" attribute controls whether this content is ignored by a SMIL 1.0 player, or results in a syntax error.

If the value of the "skip-content" attribute is "true", and one of the cases above apply, the content of the element is ignored. If the value is "false", the content of the element is processed.
The default value for "skip-content" is "true".

title

This attribute offers advisory information about the element for which it is set. Values of the title attribute may be rendered by user agents in a variety of ways. For instance, visual browsers frequently display the title as a "tool tip" (a short message that appears when the pointing device pauses over an object).
It is strongly recommended that all "region" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

top

The use and definition of this attribute are identical to the "top" property in the CSS2 specification. Attribute values have the same restrictions as the attribute values of the "height" attribute.
The default value is zero.

width

The use and definition of this attribute are identical to the "width" property in the CSS2 specification. Attribute values have the same restrictions as the attribute values of the "height" attribute.

z-index

The use and definition of this attribute are identical to the "z-index" property in the CSS2 specification, with the following exception:

- If two boxes generated by elements A and B have the same stack level, then
  1. If the display of an element A starts later than the display of an element B, the box of A is stacked on top of the box of B (temporal order).
  2. If the display of the elements starts at the same time, and an element A occurs later in the SMIL document text than an element B, the box of A is stacked on top of the box of B (document tree order as defined in CSS2).

Element Content

"region" is an empty element.

3.3.2 The `root-layout` element

The "root-layout" element determines the value of the layout properties of the root element, which in turn determines the size of the viewport, e.g. the window in which the SMIL presentation is rendered.

If a document contains more than one "root-layout" element, this is an error, and the document should not be displayed.

Element Attributes

The "root-layout" element can have the following attributes:

background-color
    Defined in Section 3.3.1
height
    Defined in Section 3.3.1
    Sets the height of the root element. Only length values are allowed.
id
    Defined in Section 2
skip-content
    Defined in Section 3.3.1
title
    Defined in Section 3.3.1
width
    Defined in Section 3.3.1
    Sets the width of the root element. Only length values are allowed.

Element Content

"root-layout" is an empty element.

3.4 The `meta` Element

The "meta" element can be used to define properties of a document (e.g., author, expiration date, a list of key words, etc.) and assign values to those properties. Each "meta" element specifies a single property/value pair.

Element Attributes

The "meta" element can have the following attributes:

content
    This attribute specifies the value of the property defined in the meta element.
    The "content" attribute is required for "meta" elements.

id
    Defined in Section 2
name
    This attribute identifies the property defined in the meta element.
    The "name" attribute is required for "meta" elements.
skip-content
    Defined in Section 3.3.1

The list of properties is open-ended. This specification defines the following properties:

base
    The value of this property determines the base URI for all relative URIs used in the document.
pics-label or PICS-Label
    The value of this property specifies a valid rating label for the document as defined by PICS [PICS].
title
    The value of this property contains the title of the presentation.

Element Content

"meta" is an empty element.

4 The Document Body

4.1 The body Element

The "body" element contains information that is related to the temporal and linking behavior of the document. It implicitly defines a "seq" element (defined in Section 4.2.2, see Section 4.2.4 for a definition of the temporal semantics of the "body" element).

Element Attributes

The "body" element can have the following attribute:

id
    Defined in Section 2

Element Content

The "body" element can contain the following children:

a
    Defined in Section 4.5.1
animation
    Defined in Section 4.2.3
audio
    Defined in Section 4.2.3
img

Defined in Section 4.2.3

par
: Defined in Section 4.2.1

ref
: Defined in Section 4.2.3

seq
: Defined in Section 4.2.2

switch
: Defined in Section 4.3

text
: Defined in Section 4.2.3

textstream
: Defined in Section 4.2.3

video
: Defined in Section 4.2.3

4.2 Synchronization Elements

4.2.1 The `par` Element

The children of a par element can overlap in time. The textual order of appearance of children in a par has no significance for the timing of their presentation.

Element Attributes

The "par" element can have the following attributes:

abstract
: A brief description of the content contained in the element.

author
: The name of the author of the content contained in the element.

begin
: This attribute specifies the time for the explicit begin of an element. See Section 4.2.4 for a definition of its semantics.
The attribute can contain the following two types of values:

  delay-value
  : A delay value is a clock-value measuring presentation time. Presentation time advances at the speed of the presentation. It behaves like the timecode shown on a counter of a tape-deck. It can be stopped, decreased or increased either by user actions, or by the player itself.
  The semantics of a delay value depend on the element's first ancestor that is a synchronization element (i.e. ancestors that are "a" or "switch" elements are ignored):
    - If this ancestor is a "par" element, the value defines a delay from the effective begin of that element (see Figure 4.1).
    - If this ancestor is a "seq" element (defined in Section 4.2.2), the value defines a delay from the effective end of the first lexical predecessor that is a synchronization element (see Figure 4.2).

  event-value

The element begins when a certain event occurs (see Figure 4.3). Its value is an element-event (see Definition below).
The element generating the event must be "in scope". The set of "in scope" elements S is determined as follows:

1. Take all children from the element's first ancestor that is a synchronization element and add them to S.
2. Remove all "a" and "switch" elements from S. Add the children of all "a" elements to S, unless they are "switch" elements.

The resulting set S is the set of "in scope" elements.

copyright
: The copyright notice of the content contained in the element.

dur
: This attribute specifies the explicit duration of an element. See Section 4.2.4 for a definition of its semantics. The attribute value can be a clock value, or the string "indefinite".

end
: This attribute specifies the explicit end of an element. See Section 4.2.4 for a definition of its semantics. The attribute can contain the same types of attribute values as the "begin" attribute.

endsync
: For a definition of the semantics of this attribute, see Section 4.2.4. The attribute can have the following values:

- first
  For a definition of the semantics of this value, see Section 4.2.4.
- id-ref
  This attribute value has the following syntax:

```
id-ref ::= "id(" id-value ")"
```
  where "id-value" must be a legal XML identifier.
  For a definition of the semantics of this value, see Section 4.2.4.
- last
  For a definition of the semantics of this value, see Section 4.2.4.

The default value of "endsync" is "last".

id
: Defined in Section 2

region
: This attribute specifies an abstract rendering surface (either visual or acoustic) defined within the layout section of the document. Its value must be an XML identifier. If no rendering surface with this id is defined in the layout section, the values of the formatting properties of this element are determined by the default layout.
The "region" attribute on "par" elements cannot be used by the basic layout language for SMIL defined in this specification. It is added for completeness, since it may be required by other layout languages.

repeat
: For a definition of the semantics of this attribute, see Section 4.2.4. The attribute value can be an integer, or the string "indefinite". The default value is 1.

system-bitrate
: Defined in Section 4.4

system-captions
    Defined in Section 4.4
system-language
    Defined in Section 4.4
system-overdub-or-caption
    Defined in Section 4.4
system-required
    Defined in Section 4.4
system-screen-size
    Defined in Section 4.4
system-screen-depth
    Defined in Section 4.4
title
    Defined in Section 3.3.1
    It is strongly recommended that all "par" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Note on Synchronization between Children

The accuracy of synchronization between the children in a parallel group is implementation-dependent. Take the example of synchronization in case of playback delays, i.e. the behavior when the "par" element contains two or more continuous media types such as audio or video, and one of them experiences a delay.
A player can show the following synchronization behaviors:

hard synchronization
    The player synchronizes the children in the "par" element to a common clock (see Figure 4.4 a)).
soft synchronization
    Each child of the "par" element has its own clock, which runs independently of the clocks of other children in the "par" element (see Figure 4.4 b)).

Attribute Values

clock value
    Clock values have the following syntax:

```
Clock-val               ::= Full-clock-val | Partial-clock-val | Timecount-val
Full-clock-val          ::= Hours ":" Minutes ":" Seconds ("." Fraction)?
Partial-clock-val       ::= Minutes ":" Seconds ("." Fraction)?
Timecount-val           ::= Timecount ("." Fraction)?
                            ("h" | "min" | "s" | "ms")? ; default is "s"
Hours                   ::= 2DIGIT; any positive number
Minutes                 ::= 2DIGIT; range from 00 to 59
Seconds                 ::= 2DIGIT; range from 00 to 59
Fraction        ::= DIGIT+
Timecount               ::= DIGIT+
2DIGIT         ::= DIGIT DIGIT
DIGIT         ::= [0-9]
```

    The following are examples of legal clock values:

- Full clock value: 02:30:03 = 2 hours, 30 minutes and 3 seconds
- Partial clock value: 02:33 = 2 minutes and 33 seconds
- Timecount values:
  3h = 3 hours
  45min = 45 minutes
  30s = 30 seconds
  5ms = 5 milliseconds

A fraction x with n digits represents the following value:

x * 1/10**n

Examples:

00.5s = 5 * 1/10 seconds = 500 milliseconds
00:00.005 = 5 * 1/1000 seconds = 5 milliseconds

element-event value
: An *element event* value specifies a particular event in a synchronization element.
An element event has the following syntax:

```
Element-event      ::= "id(" Event-source ")(" Event ")"
Event-source       ::= Id-value
Event              ::= "begin" | Clock-val | "end"
```

The following events are defined:

begin
: This event is generated at an element's effective begin.
Example use: `begin="id(x)(begin)"`

clock-val
: This event is generated when a clock associated with an element reaches a particular value. This clock starts at 0 at the element's effective begin. For "par" and "seq" elements, the clock gives the presentation time elapsed since the effective begin of the element. For media object elements, the semantics are implementation-dependent. The clock may either give presentation time elapsed since the effective begin, or it may give the media time of the object. The latter may differ from the presentation time that elapsed since the object's display was started e.g. due to rendering or network delays, and is the recommended approach.
It is an error to use a clock value that exceeds the value of the effective duration of the element generating the event.

Example use: `begin="id(x)(45s)"`

end
: This event is generated at the element's effective end.
Example use: `begin="id(x)(end)"`

Element Content

The par element can contain the following children:

a
    Defined in Section 4.5.1
animation
    Defined in Section 4.2.3
audio
    Defined in Section 4.2.3
img
    Defined in Section 4.2.3
par
    Defined in Section 4.2.1
ref
    Defined in Section 4.2.3
seq
    Defined in Section 4.2.2
switch
    Defined in Section 4.3
text
    Defined in Section 4.2.3
textstream
    Defined in Section 4.2.3
video
    Defined in Section 4.2.3

All of these elements may appear multiple times as direct children of a par element.

4.2.2 The `seq` Element

The children of a "seq" element form a temporal sequence.

Attributes

The seq element can have the following attributes:

abstract
    Defined in Section 4.2.1
author
    Defined in Section 4.2.1
begin
    Defined in Section 4.2.1
copyright
    Defined in Section 4.2.1
dur
    Defined in Section 4.2.1
end
    Defined in Section 4.2.1
id

Defined in Section 2

region

Defined in Section 4.2.1

The region attribute on "seq" elements cannot be used by the basic layout language for SMIL defined in this specification. It is added for completeness, since it may be required by other layout languages.

repeat

Defined in Section 4.2.1

system-bitrate

Defined in Section 4.4

system-captions

Defined in Section 4.4

system-language

Defined in Section 4.4

system-overdub-or-caption

Defined in Section 4.4

system-required

Defined in Section 4.4

system-screen-size

Defined in Section 4.4

system-screen-depth

Defined in Section 4.4

title

Defined in Section 3.3.1

It is strongly recommended that all "seq" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

The seq element can contain the following children:

a

Defined in Section 4.5.1

animation

Defined in Section 4.2.3

audio

Defined in Section 4.2.3

img

Defined in Section 4.2.3

par

Defined in Section 4.2.1

ref

Defined in Section 4.2.3

seq

Defined in Section 4.2.2

switch

Defined in Section 4.3

text

Defined in Section 4.2.3

textstream
: Defined in Section 4.2.3

video
: Defined in Section 4.2.3

4.2.3 Media Object Elements: The `ref`, `animation`, `audio`, `img`, `video`, `text` and `textstream` elements

The media object elements allow the inclusion of media objects into a SMIL presentation. Media objects are included by reference (using a URI).

There are two types of media objects: media objects with an intrinsic duration (e.g. video, audio) (also called "continuous media"), and media objects without intrinsic duration (e.g. text, image) (also called "discrete media").

Anchors and links can be attached to visual media objects, i.e. media objects rendered on a visual abstract rendering surface.

When playing back a media object, the player must not derive the exact type of the media object from the name of the media object element. Instead, it must rely solely on other sources about the type, such as type information contained in the "type" attribute, or the type information communicated by a server or the operating system.

Authors, however, should make sure that the group into which of the media object falls (animation, audio, img, video, text or textstream) is reflected in the element name. This is in order to increase the readability of the SMIL document. When in doubt about the group of a media object, authors should use the generic "ref" element.

Element Attributes

Media object elements can have the following attributes:

abstract
: Defined in Section 4.2.1

alt
: For user agents that cannot display a particular media-object, this attribute specifies alternate text. It is strongly recommended that all media object elements have an "alt" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

author
: Defined in Section 4.2.1

begin
: Defined in Section 4.2.1

clip-begin
: The clip-begin attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object.
: Values in the clip-begin attribute have the following syntax:

```
Clip-time-value   ::= Metric "=" ( Clock-val | Smpte-val )
```

```
Metric              ::= Smpte-type | "npt"
Smpte-type          ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val           ::= Hours ":" Minutes ":" Seconds
                        [ ":" Frames [ "." Subframes ]]
Hours               ::= 2DIGIT
Minutes             ::= 2DIGIT
Seconds             ::= 2DIGIT
Frames              ::= 2DIGIT
Subframes           ::= 2DIGIT
```

The value of this attribute consists of a metric specifier, followed by a time value whose syntax and semantics depend on the metric specifier. The following formats are allowed:

SMPTE Timestamp

SMPTE time codes [SMPTE] can be used for frame-level access accuracy. The metric specifier can have the following values:

smpte

smpte-30-drop

These values indicate the use of the "SMPTE 30 drop" format with 29.97 frames per second. The "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute.

smpte-25

The "frames" field in the time specification can assume the values 0 through 24.

The time value has the format hours:minutes:seconds:frames.subframes. If the frame value is zero, it may be omitted. Subframes are measured in one-hundredth of a frame.

Examples:

```
clip-begin="smpte=10:12:33:20"
```

Normal Play Time

Normal Play Time expresses time in terms of SMIL clock values. The metric specifier is "npt", and the syntax of the time value is identical to the syntax of SMIL clock values.

Examples:

```
clip-begin="npt=123.45s"
clip-begin="npt=12:05:35.3"
```

clip-end

The clip-end attribute specifies the end of a sub-clip of a continuous media object (such as audio, video or another presentation) that should be played. It uses the same attribute value syntax as the clip-begin attribute.

If the value of the "clip-end" attribute exceeds the duration of the media object, the value is ignored, and the clip end is set equal to the effective end of the media object.

copyright

Defined in Section 4.2.1

dur

Defined in Section 4.2.1

end

Defined in Section 4.2.1

fill
: For a definition of the semantics of this attribute, see Section 4.2.4. The attribute can have the values "remove" and "freeze".

id
: Defined in Section 2

longdesc
: This attribute specifies a link (URI) to a long description of a media object. This description should supplement the short description provided using the alt attribute. When the media-object has associated anchors, this attribute should provide information about the anchor's contents.

region
: Defined in Section 4.2.1

src
: The value of the src attribute is the URI of the media object.

system-bitrate
: Defined in Section 4.4

system-captions
: Defined in Section 4.4

system-language
: Defined in Section 4.4

system-overdub-or-caption
: Defined in Section 4.4

system-required
: Defined in Section 4.4

system-screen-size
: Defined in Section 4.4

system-screen-depth
: Defined in Section 4.4

title
: Defined in Section 3.3.1
: It is strongly recommended that all media object elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

type
: MIME type of the media object referenced by the "src" attribute.

Element Content

Media Object Elements can contain the following element:

anchor
: Defined in Section 4.5.2

4.2.4 SMIL Time Model

4.2.4.1 Time Model Values

In the following discussion, the term "element" refers to synchronization elements only.

For each element we define the implicit, explicit, desired, and effective begin, duration, and end.

The effective begin/duration/end specify what the reader of the document will perceive.

The implicit, explicit, and desired values are auxiliary values used to define the effective values.

The rules for calculating each of these values for the elements defined in SMIL 1.0 are described in the next section.

1. Each element in SMIL has an *implicit begin*.
2. Each element can be assigned an *explicit begin* by adding a "begin" attribute to the element:

   ```
   begin = "value of explicit-begin"
   ```

   It is an error if the explicit begin is earlier than the implicit begin of the element.

3. Each element in SMIL has an *implicit end*.
4. Each element can be assigned an *explicit end* by adding an "end" attribute to the element:

   ```
   end = "value of explicit-end"
   ```

5. The *implicit duration* of an element is the difference between the implicit end and the implicit begin.
6. Each element in SMIL can be assigned an *explicit duration* by adding a "dur" attribute to the element:

   ```
   dur = "value of explicit-duration"
   ```

7. The *desired begin* of an element is equal to the explicit begin if one is given, otherwise the desired begin is equal to the implicit begin.
8. Each element has a *desired end*.
9. The *desired duration* of an element is the difference between the desired end and the desired begin.
10. Each element has an *effective begin*.
11. Each element has an *effective end*. (Note: the effective end of a child element can never be later than the effective end of its parent.)
12. The *effective duration* of an element is the difference between the effective end and the effective begin.

4.2.4.2 Determining Time Model Values for SMIL 1.0 Elements

This section defines how time model values are calculated for the synchronization elements of SMIL 1.0 in cases that are not covered by the rules in Section 4.2.4.1.

Determining the *implicit begin* of an element

- The implicit begin of the first child of the "body" element is when the document starts playing. When this is falls outside the scope of this document.
- The implicit begin of a child of a "par" element is equal to the effective begin of the "par" element.
- The implicit begin of the first child of a "seq" element is equal to the effective begin of the "seq" element.
- The implicit begin of any other child of a "seq" element is equal to the desired end time of the previous child of the "seq" element.

Determining the *implicit end* of an element

The first description that matches the element is the one that is to be used:

- An element with a "repeat" attribute with value "indefinite" has an implicit end immediately after its effective begin.
- An element with a "repeat" attribute with a value other than "indefinite" has an implicit end equal to the implicit end of a seq element with the stated number of copies of the element without "repeat" attribute as children.
- A media object element referring to a continuous media object has an implicit end equal to the sum of the effective begin of the element and the intrinsic duration of the media object.
- A media object element referring to a discrete media object such as text or image has an implicit end immediately after its effective begin.
- A "seq" element has an implicit end equal to the desired end of its last child.
- A "par" element has an implicit end that depends on the value of the "endsync" attribute. The implicit end is equal to the sum of the effective begin of the "par" element and the implicit duration which is derived as follows:
    - If the value of the "endsync" attribute is "last", or if the "endsync" attribute is missing, the implicit duration of the "par" element is the maximum of the desired durations of its children.
    - If the value of the "endsync" attribute is "first", the implicit duration of the "par" element is the minimum of the desired durations of its children.
    - If the value of the "endsync" attribute is an id-ref, the implicit duration of the "par" element is equal to the desired duration of the child referenced by the "id-ref".

Determining the *desired end* of an element

- If the element has both an explicit duration and an explicit end, the desired end is the minimum of:
    - the sum of the desired begin and the explicit duration; and
    - the explicit end.
- If the element has an explicit duration but no explicit end, the desired end is the sum of the desired begin and the explicit duration.
- If the element has an explicit end but no explicit duration, the desired end is equal to the explicit end
- Otherwise, the desired end is equal to the implicit end.

Determining the *desired begin* of an element

The desired begin of an element is determined by using rule 7 in Section 4.2.4.1.

Determining the *effective begin* of an element

The *effective begin* of an element is equal to the desired begin of the element, unless the effective end of the parent element is earlier than this time, in which case the element is not shown at all.

Determining the *effective end* of an element

- The effective end of the last child of the body element is player-dependent. The effective end is at least as late as the desired end, but whether it is any later is implementation-dependent.
- The effective end of the child of a "par" element can be derived as follows:
    - If the child has a "fill" attribute, and the value of the "fill" attribute is "freeze", the effective end of the child element is equal to the effective end of the parent.

    The last state of the element is retained on the screen until the effective end of the element.
    - If the child has a "fill" attribute, and the value of the "fill" attribute is "remove", the effective end of the child element is the minimum of the effective end of the parent and the desired end of the child element.
    - If the child element has no "fill" attribute, the effective end of the child depends on whether or not the child has an explicit duration or end.
        - If the child has an explicit duration or end, the effective end is determined as if the element had a "fill" attribute with value "remove".
        - If the child has neither an explicit duration nor an explicit end, the effective end is determined as if the element had a "fill" attribute with value "freeze".
- The effective end of the last child of a "seq" element is derived in the same way as the effective end of a child of a "par" element.
- The effective end of any other child of a "seq" element can be derived as follows:
    - If the child has a "fill" attribute, and the value of the "fill" attribute is "freeze", the effective end of the child element is equal to the effective begin of the next element
    - If the child has a "fill" attribute, and the value of the "fill" attribute is "remove", the effective end of the child element is the minimum of the effective begin of the next element and the desired end of the next child element.
    - If the child element has no "fill" attribute, the effective end of the child depends on whether or not the child has an explicit duration or end.
        - If the child has an explicit duration or end, the effective end is determined as if the element had a fill attribute with value "remove".
        - If the child has neither an explicit duration nor an explicit end, the effective end is determined as if the element had a fill attribute with value "freeze".

4.3 The `switch` Element

The switch element allows an author to specify a set of alternative elements from which only one acceptable element should be chosen. An element is acceptable if the element is a SMIL 1.0 element, the media-type can be decoded, and all of the test-attributes (see Section 4.4) of the element evaluate to "true".

An element is selected as follows: the player evaluates the elements in the order in which they occur in the switch element. The first acceptable element is selected at the exclusion of all other elements within the switch.

Thus, authors should order the alternatives from the most desirable to the least desirable. Furthermore, authors should place a relatively fail-safe alternative as the last item in the <switch> so that at least one item within the switch is chosen (unless this is explicitly not desired). Implementations should NOT arbitrarily pick an object within a <switch> when test-attributes for all fail.

Note that http URIs provide for content-negotiation, which may be an alternative to using the "switch" element in some cases.

Attributes

The switch element can have the following attributes:

id
    Defined in Section 2

title
: Defined in Section 3.3.1
: It is strongly recommended that all switch elements have a "title" attribute with a meaningful description Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

If the "switch" element is used as a direct or indirect child of a "body" element, it can contain the following children:

a
: Defined in Section 4.5.1

animation
: Defined in Section 4.2.3

audio
: Defined in Section 4.2.3

img
: Defined in Section 4.2.3

par
: Defined in Section 4.2.1

ref
: Defined in Section 4.2.3

seq
: Defined in Section 4.2.2

switch
: Defined in Section 4.3

text
: Defined in Section 4.2.3

textstream
: Defined in Section 4.2.3

video
: Defined in Section 4.2.3

All of these elements may appear multiple times as children of a "switch" element.

If the "switch" element is used within a "head" element, it can contain the following child:

layout
: Defined in Section 3.2
: Multiple layout elements may occur within the switch element.

4.4 Test Attributes

This specification defines a list of test attributes that can be added to any synchronization element, and that test system capabilities and settings. Conceptually, these attributes represent boolean tests. When one of the test attributes specified for an element evaluates to "false", the element carrying this attribute is ignored.

Within the list below, the concept of "user preference" may show up. User preferences are usually set by the playback engine using a preferences dialog box, but this specification does not place any restrictions on how such preferences are communicated from the user to the SMIL player.

The following test attributes are defined in SMIL 1.0:

system-bitrate

This attribute specifies the approximate bandwidth, in bits per second available to the system. The measurement of bandwidth is application specific, meaning that applications may use sophisticated measurement of end-to-end connectivity, or a simple static setting controlled by the user. In the latter case, this could for instance be used to make a choice based on the users connection to the network. Typical values for modem users would be 14400, 28800, 56000 bit/s etc. Evaluates to "true" if the available system bitrate is equal to or greater than the given value. Evaluates to "false" if the available system bitrate is less than the given value.
The attribute can assume any integer value greater than 0. If the value exceeds an implementation-defined maximum bandwidth value, the attribute always evaluates to "false".

system-captions

This attribute allows authors to distinguish between a redundant text equivalent of the audio portion of the presentation (intended for a audiences such as those with hearing disabilities or those learning to read who want or need this information) and text intended for a wide audience. The attribute can has the value "on" if the user has indicated a desire to see closed-captioning information, and it has the value "off" if the user has indicated that they don't wish to see such information. Evaluates to "true" if the value is "on", and evaluates to "false" if the value is "off".

system-language

The attribute value is a comma-separated list of language names as defined in [RFC1766].

Evaluates to "true" if one of the languages indicated by user preferences exactly equals one of the languages given in the value of this parameter, or if one of the languages indicated by

user preferences exactly equals a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-".

Evaluates to "false" otherwise.

Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags for which this tag is a prefix.

The prefix rule simply allows the use of prefix tags if this is the case.

Implementation note: When making the choice of linguistic preference available to the user, implementors should take into account the fact that users are not familiar with the details of language matching as described above, and should provide appropriate guidance. As an example, users may assume that on selecting "en-gb", they will be served any kind of English document if British English is not available. The user interface for setting user preferences should guide the user to add "en" to get the best matching behavior.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi", presented simultaneously in the original Maori and English versions, would call for:

```
            <audio src="foo.rm" system-language="mi, en"/>
```

However, just because multiple languages are present within the object on which the system-language test attribute is placed, this does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin," which is clearly intended to be used by an English-literate audience. In this case, the system-language test attribute should only include "en".

Authoring note: Authors should realize that if several alternative language objects are enclosed in a "switch", and none of them matches, this may lead to situations such as a video being shown without any audio track. It is thus recommended to include a "catch-all" choice at the end of such a switch which is acceptable in all cases.

system-overdub-or-caption
: This attribute is a setting which determines if users prefer overdubbing or captioning when the option is available. The attribute can have the values "caption" and "overdub". Evaluates to "true" if the user preference matches this attribute value. Evaluates to "false" if they do not match.

system-required
: This attribute specifies the name of an extension. Evaluates to "true" if the extension is supported by the implementation, otherwise, this evaluates to "false". In a future version of SMIL, this attribute value will be an XML namespace [NAMESPACES].

system-screen-size
: Attribute values have the following syntax:
`screen-size-val ::= screen-height"X"screen-width`
Each of these is a pixel value, and must be an integer value greater than 0. Evaluates to "true" if the SMIL playback engine is capable of displaying a presentation of the given size. Evaluates to "false" if the SMIL playback engine is only capable of displaying a smaller presentation.

system-screen-depth
: This attribute specifies the depth of the screen color palette in bits required for displaying the element. The value must be greater than 0. Typical values are 1, 8, 24 .... Evaluates to "true" if the SMIL playback engine is capable of displaying images or video with the given color depth. Evaluates to "false" if the SMIL playback engine is only capable of displaying images or video with a smaller color depth.

4.5 Hyperlinking Elements

The link elements allows the description of navigational links between objects.

SMIL provides only for in-line link elements. Links are limited to uni-directional single-headed links (i.e. all links have exactly one source and one destination resource). All links in SMIL are actuated by the user.

Handling of Links in Embedded Documents

Due to its integrating nature, the presentation of a SMIL document may involve other (non-SMIL) applications or plug-ins. For example, a SMIL browser may use an HTML plug-in to display an embedded HTML page. Vice versa, an HTML browser may use a SMIL plug-in to display a SMIL document embedded in an HTML page.

In such presentations, links may be defined by documents at different levels and conflicts may arise. In this case, the link defined by the containing document should take precedence over the link defined by the embedded object. Note that since this might require communication between the browser and the plug-in, SMIL implementations may choose not to comply with this recommendation.

If a link is defined in an embedded SMIL document, traversal of the link affects only the embedded SMIL document.

If a link is defined in a non-SMIL document which is embedded in a SMIL document, link traversal can only affect the presentation of the embedded document and not the presentation of the containing SMIL document. This restriction may be released in future versions of SMIL.

Addressing

SMIL supports name fragment identifiers and the '#' connector. This means that SMIL supports locators as currently used in HTML (e.g. it uses locators of the form "http://foo.com/some/path#anchor1").

Linking to SMIL Fragments

A locator that points to a SMIL document may contain a fragment part (e.g. http://www.w3.org/test.smi#par1). The fragment part is an id value that identifies one of the elements within the referenced SMIL document. If a link containing a fragment part is followed, the presentation should start as if the user had fast-forwarded the presentation represented by the destination document to the effective begin of the element designated by the fragment.

The following special cases can occur:

1. The element addressed by the link has a "repeat" attribute.
   1. If the value of the "repeat" attribute is N, all N repetitions of the element are played.
   2. If the value of the "repeat" attribute is "indefinite", playback ends according to the rules defined for repeat value "indefinite".
2. The element addressed by the link is contained within another element that contains a "repeat" attribute.
   1. If the value of the "repeat" attribute is N, playback starts at the beginning of the element addressed by the link, followed by N-1 repetitions of the element containing the "repeat" attribute.
   2. If the value of the "repeat" attribute is "indefinite", playback starts at the beginning of the element addressed by the link. Playback ends according to the rules defined for repeat value "indefinite".
3. The element addressed by the link is content of a "switch" element: It is forbidden to link to elements that are the content of "switch" elements.

4.5.1 The a Element

The functionality of the "a" element is very similar to the functionality of the "a" element in HTML 4.0 [HTML40] . SMIL adds an attribute "show" that controls the temporal behavior of the source when the link is followed. For synchronization purposes, the "a" element is transparent, i.e. it does not influence the synchronization of its child elements. "a" elements may not be nested. An "a" element must have an href attribute.

Attributes

The "a" element can have the following attributes:

id
: Defined in Section 2

href
: This attribute contains the URI of the link's destination.
The "href" attribute is required for "a" elements.

show
: This attribute controls the behavior of the source document containing the link when the link is followed. It can have one of the following values:

  - "replace": The current presentation is paused at its current state and is replaced by the destination resource. If the player offers a history mechanism, the source presentation resumes from the state in which it was paused when the user returns to it.
  - "new": The presentation of the destination resource starts in a new context, not affecting the source resource.
  - "pause": The source presentation is paused at its current state, and the destination resource starts in a new context. When the display of the destination resource ends, the source presentation resumes from the state in which it was paused.

  The default value of "show" is "replace".

title
: Defined in Section 3.3.1
It is strongly recommended that all "a" elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

The "a" element can contain the following children:

animation
: Defined in Section 4.2.3

audio
: Defined in Section 4.2.3

img
: Defined in Section 4.2.3

par
: Defined in Section 4.2.1

ref
: Defined in Section 4.2.3

seq
: Defined in Section 4.2.2

switch
: Defined in Section 4.3

text
: Defined in Section 4.2.3

textstream

Defined in Section 4.2.3

video
: Defined in Section 4.2.3

4.5.2 The `anchor` Element

The functionality of the "a" element is restricted in that it only allows associating a link with a complete media object. HTML image maps have demonstrated that it is useful to associate links with spatial subparts of an object. The anchor element realizes similar functionality for SMIL:

1. The anchor element allows associating a link destination to spatial and temporal subparts of a media object, using the "href" attribute (in contrast, the "a" element only allows associating a link with a complete media object).
2. The anchor element allows making a subpart of the media object the destination of a link, using the "id" attribute.
3. The anchor element allows breaking up an object into spatial subparts, using the "coords" attribute.
4. The anchor element allows breaking up an object into temporal subparts, using the "begin" and "end" attributes. The values of the begin and end attributes are relative to the beginning of the media object.

Attributes

The anchor element can have the following attributes:

begin
: Defined in Section 4.2.1

coords
: The value of this attribute specifies a rectangle within the display area of a visual media object. Syntax and semantics of this attribute are similar to the coords attribute in HTML image maps, when the link is associated with a rectangular shape. The rectangle is specified by four length values: The first two values specify the coordinates of the upper left corner of the rectangle.The second two values specify the coordinates of the lower right corner of the rectangle. Coordinates are relative to the top left corner of the visual media object (see Figure 4.5). If a coordinate is specified as a percentage value, it is relative to the total width or height of the media object display area.
An attribute with an erroneous coords value is ignored (right-x smaller or equal to left-x, bottom-y smaller or equal to top-y). If the rectangle defined by the coords attribute exceeds the area covered by the media object, exceeding height and width are clipped at the borders of the media object.
Values of the coords attribute have the following syntax:

```
coords-value ::= left-x "," top-y "," right-x "," bottom-y
```

end
: Defined in Section 4.2.1

id
: Defined in Section 2

show
: Defined in Section 4.5.1

skip-content

Defined in Section 3.3.1

title

Defined in Section 3.3.1
It is strongly recommended that all anchor elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

CARDIFF CARDIFF UNIVERSITY
EXAMINATION PAPER

Academic Year:	2000-2001
Examination Period:	Autumn 2000
Examination Paper Number:	CM0340
Examination Paper Title:	Multimedia
Duration:	2 hours

Do not turn this page over until instructed to do so by the Senior Invigilator.

Structure of Examination Paper:
There are three pages.
There are four questions in total.

There are no appendices.

The maximum mark for the examination paper is 100% and the mark obtainable for a question or part of a question is shown in brackets alongside the question.

Students to be provided with:
The following items of stationery are to be provided:
One answer book.

Instructions to Students:
Answer THREE questions.

1. (a) What is meant by the terms *Multimedia* and *Hypermedia*? Distinguish between these two concepts. [2]

   (b) What is meant by the terms *static* media and *dynamic* media? Give two examples of each type of media. [4]

   (c) What are the main facilities that must be provides in a system designed to support the integration of multimedia into a multimedia presentation? [8]

   (d) Describe giving suitable code fragments how you would effectively combine a video clip and an audio clip in an MHEG application and start a subtitle text display 19000 milliseconds into the video clip. You may assume that both clips are of the same duration and must start at the same instant. [13]

2. (a) Why is data compression necessary for Multimedia activities?

   [3]

   (b) Briefly explain how the LZW Transform Operates. What common compression methods utilise this transform? [10]

   (c) Show how the LZW transform would be used to encode the following 2D array of image data. You should use 2x2 window elements for the tokens used in the encoding.

0	1	0	2	2	0
2	0	1	1	1	0
0	2	0	1	0	2
1	0	2	0	1	1
2	0	0	2	0	2
1	0	1	0	1	1

[14]

3. (a) What key features of Quicktime have led to its adoption and acceptance as an international multimedia format? [4]

   (b) Briefly outline the Quicktime Architecture and its key components. [10]

   (c) Quicktime provides many basic built-in *visual effect* procedures. By using fragments of Java code show how a cross-fade effect between two images can be created. You solution should concentrate only on the Java code specific to producing the Quicktime effect. You may assume that the images are already imported into the application and are referred to as sourceImage and destImage. You should not consider any Graphical Interfacing aspects of the coding.

4. (a) What is MIDI? How is a basic MIDI message structured? [6]

   (b) In what ways can MIDI be used effectively in Multimedia Applications, as opposed to strictly musical applications? [8]

   (c) How can MIDI be used with modern data compression techniques? Briefly describe how such compression techniques may be implemented. [13]

Multimedia
BSc Exam Questions Jan 2001 SOLUTIONS

Exam paper format:

(b) Time Allowed: 2 Hours
(c) Answer 3 Questions out of 4
(d) Each Question Carries 27 Marks

1. (a) What is meant by the terms Multimedia and Hypermedia? Distinguish between these two concepts.

Multimedia ---- An Application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.

Hypermedia --- An application which uses associative relationships among information contained within multiple media data for the purpose of facilitating access to, and manipulation of, the information encapsulated by the data.

2 MARKS ---- BOOKWORK

(b) What is meant by the terms static media and dynamic media? Give two examples of each type of media.

Static Media – does not change over time, e.g. text, graphics

Dynamic Media --- Time dependent (Temporal), e.g. Video, sound, animation.

4 MARKS --- BOOKWORK

(c) What are the main facilities that must be provides in a system designed to support the integration of multimedia into a multimedia presentation?

The following functionality should be provided:

- Digital Representation of Media --- Many formats for many media
- Capture: Digitisation of Media --- special Hardware/Software

- Creation and editing --- assemble media and alter it
- Storage Requirements --- significant for multimedia
- Compression --- related to above and below, ie can save on storage but can hinder retrieval
- Structuring and retrieval methods of media --- simple to advanced DataBase Storage
- Display or Playback methods --- effect of retrieval must view data
- Media Synchronisation --- display multimedia as it is intended

8 MARKS --- BOOKWORK

(*d) Describe giving suitable code fragments how you would effectively combine and start at the same time a video clip and an audio clip in an MHEG application and start a subtiltle text display 19000 milliseconds into the video clip. You may assume that both clips are of the same duration and must start at the same instant.*

TheMHEG code listing below is illustrates the solution:

```
{:Application ("SYNC_APP.mh5" 0)
:OnStartUp ( // sequence of initialization
actions
}
{:Scene ("main_scene.mh5" 0)
:OnStartUp ( // sequence of initialization
actions
preload (2) // the connection to the
source of the video clip is set up
preload (3) // the connection to the
source of the audio clip is set up

...
setCounterTrigger (2 3 190000) // book a
time code event at 190000 msec for example
...
)
:Items ( // both presentable ingredients and
links
{:Bitmap 1 // background bitmap  NOT IMPORTANT FOR SOLN
:InitiallyActive true
:CHook 3 // JPEG
:OrigContent
:ContentRef ("background.jpg")
:OrigBoxSize 800 600
:OrigPosition 0 0
}
{:Stream 2 // video clip
:InitiallyActive false
:CHook 101 // MPEG-1
:OrigContent
:ContentRef ("video.mpg")
:Multiplex (
{:Audio 3 // audio component of the
video clip
:ComponentTag 1 // refers to audio
elementary stream
:InitiallyActive true
}
... // possibly  more presentable ingredients

}
{:Link 49 // the video clip crosses a pre
defined time code position
:EventSource (2) // video clip
:EventType CounterTrigger
:EventData 3 // booked at startup by
setCounterTrigger (2 3 190000)
:LinkEffect (
:SetData (5 // text subtitle is set
to a new string, that is
:NewRefContent ("subtitle.txt")) //
"Subtitle text"
:SetHighlightStatus (20 true) //
hotspot 20 is highlighted
)
}
... //  more links
)
:SceneCS 800 600 // size of the scene's
presentation space
}
```

Key Points:

- Preloading both clips is essential to start streaming:
- Need to “book” 190000 msec event for subtitles
- Content loaded and Video and audio is multiplexed
- Set a link transition for subtiltle

13 MARKS - UNSEEN

2. (a) Why is file or data compression necessary for Multimedia activities?

Multimedia files are very large therefore for storage, file transfer etc. file sizes need to be reduced often as part of the file format. Text and other files may also be encoded/compressed for email and other applications.

3 Marks -- Bookwork

(b) Briefly explain how the LZW Transform Operates. What common compression methods utilise this transform.

Suppose we want to encode the Oxford Concise English dictionary which contains about 159,000 entries. Why not just transmit each word as an 18 bit number?

Problems:

* Too many bits,
* everyone needs a dictionary,
* only works for English text.
* **Solution**: Find a way to build the dictionary adaptively.

- Original methods due to Ziv and Lempel in 1977 and 1978. Terry Welch improved the scheme in 1984 (called LZW compression).
- It is used in UNIX *compress* and GIF compression

The LZW Compression Algorithm can summarised as follows:

```
w = NIL;
while ( read a character k )
    {
      if wk exists in the dictionary
       w = wk;
      else
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
```

* Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.

10 MARKS – BOOKWORK

(c) Show how the LZW transform would be used to encode the following 2D array of image data, you should use 2x2 window elements for the characters:

0	1	0	2	2	0
2	0	1	1	1	0
0	2	0	1	0	2
1	0	2	0	1	1
2	0	0	2	0	2
1	0	1	0	1	1

SEE HANDWRITTEN SOLN attached

14 Marks UNSEEN

3 (a) What key features of Quicktime have lead to its adoption and an international multimedia format?

QuickTime is the most widely used cross-platform multimedia technology available today. QuickTime developed out of a multimedia extension for Apple's Macintosh(proprietry) System 7 operating system. It is now an international standard for multimedia interchange and is avalailbe for many platforms and as Web browser plug ins.

The following main features are:

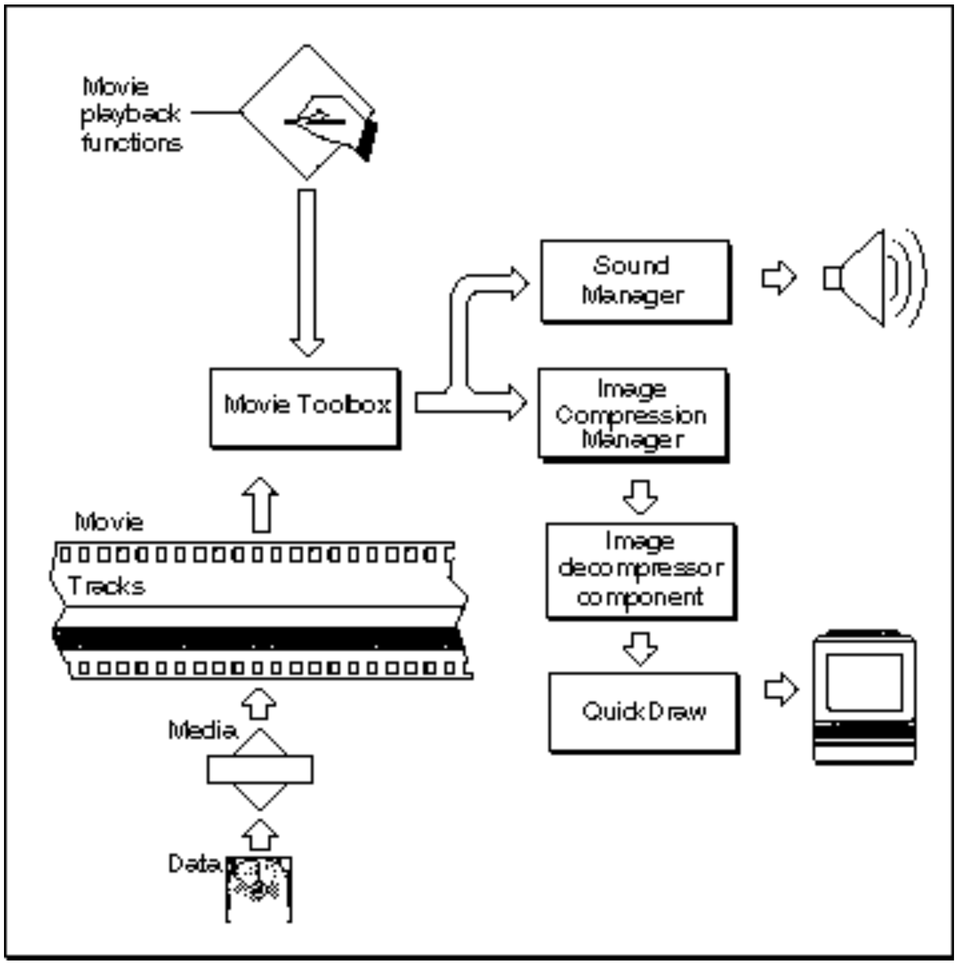
- Versatile support for web-based media
- Sophisticated playback capabilities
- Easy content authoring and editing
- QuickTime is an ***open standard*** -- it embraces other standards and incorporates them into its environment. It supports almost every major Multimedia file format

4 Marks – BOOKWORK

(b) Briefly outline the Quicktime Architecture and its key components.

The QuickTime Architecture:

QuickTime comprises two managers: the Movie Toolbox and the Image Compression Manager. QuickTime also relies on the Component Manager, as well as a set of predefined components. Figure below shows the relationships of these managers and an application that is playing a movie.



The Movie Toolbox
-- Your application gains access to the capabilities of QuickTime by calling functions in the Movie Toolbox. The Movie Toolbox allows you to store, retrieve, and manipulate time-based data that is stored in QuickTime movies. A single movie may contain several types of data. For example, a movie that contains video information might include both video data and the sound data that accompanies the video.

The Movie Toolbox also provides functions for editing movies. For example, there are editing functions for shortening a movie by removing portions of the video and sound tracks, and there are functions for extending it with the addition of new data from other QuickTime movies.

The Movie Toolbox is described in the chapter "Movie Toolbox" later in this book. That chapter includes code samples that show how to play movies.

The Image Compression Manager
--

The Image Compression Manager comprises a set of functions that compress and decompress images or sequences of graphic images.

The Image Compression Manager provides a device-independent and driver-independent means of compressing and decompressing images and sequences of images. It also contains a simple interface for implementing software and hardware image-compression algorithms. It provides system integration functions for storing compressed images as part of PICT files, and it offers the ability to automatically decompress compressed PICT files on any QuickTime-capable Macintosh computer.

In most cases, applications use the Image Compression Manager indirectly, by calling Movie Toolbox functions or by displaying a compressed picture. However, if your application compresses images or makes movies with compressed images, you will call Image Compression Manager functions.

The Image Compression Manager is described in the chapter "Image Compression Manager" later in this book. This chapter also includes code samples that show how to compress images or make movies with compressed images.

The Component Manager
--

Applications gain access to components by calling the Component Manager. The Component Manager allows you to define and register types of components and communicate with components using a standard interface. A component is a code resource that is registered by the Component Manager. The component's code can be stored in a systemwide resource or in a resource that is local to a particular application.

> Once an application has connected to a component, it calls that component directly. If you create your own component class, you define the function-level interface for the component type that you have defined, and all components of that type must support the interface and adhere to those definitions. In this manner, an application can freely choose among components of a given type with absolute confidence that each will work.

QuickTime Components :

- movie controller components, which allow applications to play movies using a standard user interface standard image compression dialog components, which allow the user to specify the parameters for a compression operation by supplying a dialog box or a similar mechanism
- image compressor components, which compress and decompress image data sequence grabber components, which allow applications to preview and record

video and sound data as QuickTime movies video digitizer components, which allow applications to control video digitization by an external device
- media data-exchange components, which allow applications to move various types of data in and out of a QuickTime movie derived media handler components, which allow QuickTime to support new types of data in QuickTime movies
- clock components, which provide timing services defined for QuickTime applications preview components, which are used by the Movie Toolbox's standard file preview functions to display and create visual previews for files sequence grabber components, which allow applications to obtain digitized data from sources that are external to a Macintosh computer
- sequence grabber channel components, which manipulate captured data for a sequence grabber component
- sequence grabber panel components, which allow sequence grabber components to obtain configuration information from the user for a particular sequence grabber channel component

10 Marks BookWork

(c) Quicktime provides many basic built-in visual effect procedures. By using fragments of Java code show how a cross-fade effect between two images can be created. You solution should concentrate only on the Java code specific to producing the Quicktime effect. You may assume that the images are already imported into the application and are referred to as sourceImage and destImage. You should not consider any Graphical Interfacing aspects of the coding.

```
This code shows how a Cross Fade Transition effect could be built  NOT
ALL THE INTERFACING STUFF INCLUDED BELOW IS REQUIRED SEE COMMENTS AFTER
FOR IMPORTANT PARTS THAT NEED ADDRESSING.

/*
 * QuickTime for Java Transition Sample Code

 */

import java.awt.*;
import java.awt.event.*;
import java.io.*;

import quicktime.std.StdQTConstants;
import quicktime.*;
import quicktime.qd.*;
import quicktime.io.*;
import quicktime.std.image.*;
import quicktime.std.movies.*;
import quicktime.util.*;
import quicktime.app.QTFactory;
import quicktime.app.time.*;
import quicktime.app.image.*;
import quicktime.app.display.*;
import quicktime.app.anim.*;
import quicktime.app.players.*;
import quicktime.app.spaces.*;
import quicktime.app.actions.*;


public class TransitionEffect extends Frame implements
StdQTConstants, QDConstants {

     public static void main(String args[]) {
          try {
               QTSession.open();

               TransitionEffect te = new
TransitionEffect("Transition Effect");
               te.pack();
               te.show();
               te.toFront();
          } catch (Exception e) {
               e.printStackTrace();
               QTSession.close();
          }

     }



     TransitionEffect(String title) throws Exception {
          super (title);

          QTCanvas myQTCanvas = new
QTCanvas(QTCanvas.kInitialSize, 0.5f, 0.5f);
          add("Center", myQTCanvas);

          Dimension d = new Dimension (300, 300);
          addWindowListener(new WindowAdapter() {
               public void windowClosing (WindowEvent e) {
                    QTSession.close();
                    dispose();
               }

               public void windowClosed (WindowEvent e) {
                    System.exit(0);
               }
          });

          QDGraphics gw = new QDGraphics (new QDRect(d));
          Compositor comp = new Compositor (gw,
QDColor.black, 20, 1);

          ImagePresenter idb = makeImagePresenter (new
QTFile (QTFactory.findAbsolutePath ("pics/stars.jpg")),
                                             new
QDRect(300, 220));
          idb.setLocation (0, 0);
          comp.addMember (idb, 2);

          ImagePresenter id = makeImagePresenter (new
QTFile (QTFactory.findAbsolutePath ("pics/water.pct")),
                                             new
QDRect(300, 80));
          id.setLocation (0, 220);
          comp.addMember (id, 4);

          CompositableEffect ce = new CompositableEffect
();
          ce.setTime(800); // TIME OF EFFECT
           ce.setSourceImage(sourceImage);
           ce. setDestinationImage(destImage);
          ce.setEffect (createSMPTEEffect,
kEffectCrossFade, KRandomCrossFadeTransitionType);

          ce.setDisplayBounds (new QDRect(0, 220, 300,
80));
          comp.addMember (ce, 3);

          Fader fader = new Fader();
          QTEffectPresenter efp = fader.makePresenter();
          efp.setGraphicsMode (new GraphicsMode (blend,
QDColor.gray));
          efp.setLocation(80, 80);
          comp.addMember (efp, 1);

          comp.addController(new TransitionControl (20, 1,
fader.getTransition()));

          myQTCanvas.setClient (comp, true);
          comp.getTimer().setRate(1);
     }

     private ImagePresenter makeImagePresenter (QTFile
file, QDRect size) throws Exception {
```

```
          GraphicsImporterDrawer if1 = new
GraphicsImporterDrawer (file);
          if1.setDisplayBounds (size);
          return ImagePresenter.fromGraphicsImporterDrawer
(if1);
     }
}
```

……..

FULL CODE NOT REQUIRED as above

Important bits

- Set up an atom container to use an SMPTE effect using CreateSMPTEeffect()

- Set up a Transition with the IMPORTANT parameters:

    - ce.setTime(800); // TIME OF EFFECT
    - ce.setSourceImage(sourceImage);
    - ce. setDestinationImage(destImage);
    - ce.setEffect (createSMPTEEffect, kEffectCrossFade, KRandomCrossFadeTransitionType);

- A doTransition() or doAction() method performs transition

13 Marks --- UNSEEN

4. (a) *What is MIDI? How is a basic MIDI message structured?*

MIDI: a protocol that enables computer, synthesizers, keyboards, and other musical or (even) multimedia devices to communicate with each other.

MIDI MESSAGE:

- MIDI message includes a status byte and up to two data bytes.
- Status byte

  The most significant bit of status byte is set to 1.
- The 4 low-order bits identify which channel it belongs to (four bits produce 16 possible channels).
- The 3 remaining bits identify the message.
- The most significant bit of data byte is set to 0.

6 Marks --- Bookwork

(b) *In what ways can MIDI be used effectively in Multimedia Applications, as opposed to strictly musical applications*?

Many Application:

Low Bandwidth/(Low Quality?) Music on Web, Quicktime etc supports Midi musical instrument set
Sound Effectts --- Low Bandwidth alternative to audio samples, Sound Set part of GM soundset
Control of external devices --- e.g Synchronistaion of Video and Audio (SMPTE), Midi System Exclusive, AUDIO RECORDERS, SAMPLERS
Control of synthesis --- envelope control etc
MPEG 4 Compression control --- see Part (c)
Digital Audio

8 Marks --- Applied Bookwork: Discussion of Information mentioned in Notes/Lectures.

(c) How can MIDI be used with modern data compression techniques? Briefly describe how such compression techniques may be implemented?

- We have seen the need for compression already in Digital Audio -- Large Data Files

- Basic Ideas of compression (see next Chapter) used as integral part of audio format -- MP3, real audio etc.

- Mpeg-4 audio -- actually combines compression synthesis and midi to have a massive impact on compression.

- Midi, Synthesis encode what note to play and how to play it with a small number of parameters -- Much greater reduction than simply having some encoded bits of audio.

- Responsibility to create audio delegated to generation side.

MPEG-4 comprises of 6 Structured Audio tools are:

- SAOL the Structured Audio Orchestra Language

- SASL the Structured Audio Score Language
- SASBF the Structured Audio Sample Bank Format
- a set of MIDI semantics which describes how to control SAOL with MIDI
- a scheduler which describes how to take the above parts and create sound
- the AudioBIFS part of BIFS, which lets you make audio soundtracks in MPEG-4 using a variety of tools and effects-processing techniques

MIDI IS the control language for the synthesis part:

- As well as controlling synthesis with SASL scripts, it can be controlled with MIDI files and scores in MPEG-4. MIDI is today's most commonly used representation for music score data, and many sophisticated authoring tools (such as sequencers) work with MIDI.

- The MIDI syntax is external to the MPEG-4 Structured Audio standard; only references to the MIDI Manufacturers Association's definition in the standard. But in order   to make the MIDI controls work right in the MPEG context, some semantics (what the instructions "mean") have been redefined in MPEG-4. The new semantics are carefully defined as part of the MPEG-4 specification.

  13 Marks --- UNSEEN but Basic Ideas mentioned in lectures and earmarked for further reading . Detailed application of Lecture notes material

Multimedia IGDS MSc Exam 1999

Setter: ADM
Checker: OFR

Time Allowed: 2 Hours
Answer 3 Questions out of 4
Each Question Carries 24 Marks

1. (a) What is meant by the terms *Multimedia* and *Hypermedia*? Distinguish between these two concepts. [2]

(b) What is meant by the terms *static* media and *dynamic* media? Give examples of each type of media. [4]

(c) What types of functionality need to be provided in order to effectively use a wide variety of media in Multimedia applications. Your answer should briefly address how such functionality can be facilitated in general Multimedia applications. [8]

(d) Different types of media will require different types of supporting operations to provide the adequate levels of functionality. For the examples of static and dynamic media given in part 1(b) briefly discuss what operations are needed to support a wide range of multimedia applications. [10]

2. (a) Why is file or data compression necessary for Multimedia activities? [2]

(b) Briefly explain how the Discrete Cosine Transform Operates, and why is it so important in data compression in Multimedia applications [10]

(c) A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. Take top left pixel as the base value for the block, pixel *A*.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel *A, i.e. B-A, C-A, D-A.*
3. Store the base pixel and the differences as the values of the transform.

Given the above transform:

(i) What is the inverse transform? [2]
(ii) How may such a transform scheme be used to compress data?

[3]

(iii) Show how you would encode and compress the following image block:

10	20	20	25
15	25	15	20
20	25	10	20
15	20	15	25

[4]

(iv) Why is this scheme not very suitable for general image compression?

[3]

3 (a) What are the major factors when considering storage requirements for Multimedia Systems?

[4]

(b) What is RAID technology and what advantages does it offer as a medium for the storage and delivery of large data?

[4]

(c) Briefly explain the *eight* levels of RAID functionality .

[8]

(d) A digital video file is 40 Mb in size. The disk subsystem has four drives and the controller is designed to support read and write onto each drive, concurrently. The digital video is stored using the *disk striping* concept. A block size of 8 Kb is used for each I/O operation.

(i) What is the performance improvement in *sequentially* reading the complete file when compared to a single drive subsystem?

(ii) What is the percentage performance improvement for this system compared to a single drive system?

[8]

4 (a) Give a definition of Virtual Reality. [2]

(b) What specialised input and output devices have been developed for Virtual Reality? Describe each device briefly. [8]

(c) Virtual Reality is not only for entertainment. How can Virtual Reality help in professional environments? Write a brief essay discussing this topic. Your answer may include current applications in this field or address future avenues for this application of the technology. [14]

Multimedia IGDS MSc Exam 1999 SOLUTIONS

Setter: ADM
Checker: OFR

Answer 3 Questions out of 4

1. (a) *What is meant by the terms Multimedia and Hypermedia? Distinguish between these two concepts.*

Multimedia ---- An Application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.

Hypermedia --- An application which uses associative relationships among information contained within multiple media data for the purpose of facilitating access to, and manipulation of, the information encapsulated by the data.

2 MARKS ---- BOOKWORK

(b) What is meant by the terms static media and dynamic media? Give examples of each type of media.

Static Media – does not change over time, e.g. text, graphics

Dynamic Media --- Time dependent (Temporal), e.g. Video, sound, animation.

4 MARKS --- BOOKWORK

(c) What issues of functionality need to be provided in order to effectively use a wide variety of media in Multimedia applications. Your answer should briefly address how such functionality can facilitated in general Multimedia applications.

The following functionality should be provided:

- Digital Representation of Media --- Many formats for many media
- Capture: Digitisation of Media --- special Hardware/Software
- Creation and editing --- assemble media and alter it
- Storage Requirements --- significant for multimedia
- Compression --- related to above and below, ie can save on storage but can hinder retrieval
- Structuring and retrieval methods of media --- simple to advanced DataBase Storage
- Display or Playback methods --- effect of retrieval must view data

- Media Synchronisation --- display multimedia as it is intended

8 MARKS --- BOOKWORK

(d) Different types of media will require different types of supporting operations to provide the adequate levels of functionality. For the examples of static and dynamic media given in part 1(b) briefly discuss what operations are need to support a wide range of multimedia applications.

A selection of the items below is reuired for good marks NOT ALL. Other Solns Possible?

Typical Range of operations required for common media

Text: Editing
Formatting
Sorting
Indexing
Searching
Encrypting
ABOVE REQUIRE: :
Character Manipulation
String Manipulation

Audio: Audio Editing
Synchronisation
Conversion/Translation
Filtering/ Sound Enhancing Operators
Compression
Searching
Indexing
ABOVE REQUIRE: :
Sample Manipulation
Waveform Manipulation

Graphics: Graphic primitive Editing
Shading
Mapping
Lighting
Viewing
Rendering
Searching
Indexing
ABOVE REQUIRE: :
Primitive Manipulation
Structural/Group Manipulation

Image: Pixel operations
Geometric Operations
Filtering
Conversion
Indexing
Compression
Searching

Animation: Primitive/Group Editing
Structural Editing
Rendering
Synchronistaion
Searching
Indexing

Video: Pixel Operations
Frame Operations
Editing
Synchronisation
Conversion
Mixing
Indexing
Searching
Video Effects/Filtering

10 MARKS --- UNSEEN

2. (a) Why is file or data compression necessary for Multimedia activities?

Multimedia files are very large therefore for storage, file transfer etc. file sizes need to be reduced. Text and other files may also be encoded/compressed for email and other applications.

2 MARKS --- BOOKWORK

(b) Briefly explain how the Discrete Cosine Transform Operates and why is it so important in data compression in Multimedia applications

The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's
visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain.

With an input image, A, the coefficients for the output "image," B, are:

$$B(k_1,k_2) = \sum_{i=0}^{N_1-1}\sum_{j=0}^{N_2-1} 4\cdot A(i,j)\cdot\cos\left[\frac{\pi\cdot k_1}{2\cdot N_1}\cdot(2\cdot i+1)\right]\cdot\cos\left[\frac{\pi\cdot k_2}{2\cdot N_2}\cdot(2\cdot j+1)\right]$$

The basic operation of the DCT is as follows:

- The input image is N2 pixels wide by N1 pixels high;
- A(i,j) is the intensity of the pixel in row i and column j;
- B(k1,k2) is the DCT coefficient in row k1 and column k2 of the DCT matrix.
- The DCT input is an 8 by 8 array of integers. This array contains each pixel's gray scale level;
- 8 bit pixels have levels from 0 to 255.
- he output array of DCT coefficients contains integers; these can range from -1024 to 1023.
- For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT.
- The lower right values represent higher frequencies, and are often small - small enough to be neglected with little visible distortion.

10 MARKS --- BOOKWORK

(c) A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. *Take top left pixel as the base value for the block, pixel A.*
2. *Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. B-A, C-A, D-A.*
3. *Store the base pixel and the differences as the values of the transform.*

Given the above transform

- *What is the inverse transform?*

Let transform be:

X0 = A
X1 = B – A
X2 = C – A
X3 = D- A

Then the inverse transform is:

An = X0
Bn = X1 + X0
Cn = X2 + X0
Dn = X3 + X0

- *How may such a transform scheme be used to compress data?*

  Any Redundancy in the data has been transformed to values, *Xi.*
  So We can compress the data by using fewer bits to represent the differences.
  I.e if we use 8 bits per pixel then the 2x2 block uses 32 bits/
  If we keep 8 bits for the base pixel, *X0*, and assign 4 bits for each difference then we only use 20 bits. Which is better than an average 5 bits/pixel

- *Show how you would encode with compression the following image block*

10	*20*	*20*	*25*
15	*25*	*15*	*20*
20	*25*	*10*	*20*
15	*20*	*15*	*25*

Basically there are 4 2x2 blocks here so apply transform to each in turn

So we get For top left 2x2
X0 = 10
X1 = 10
X2 = 15
X3 = 15

NOTE: Easy to compress these values to 4 bits/pixel woth NO LOSS. Not general though

For Top Right 2x2:
X0 = 20
X1 = 5
X2 = -5
X3 = 0

Note: Here need 1 bit for negative value so 4 bits may not seem so BIG NOW.

Bottom left 2x2 identical to top right
Bottom Right 2x2 identical to top left

- *Why is this scheme not very suitable general image compression?*

TOO SIMPLE:

Needs to operate on larger blocks (typically 8x8 min)

Calculation is also too simple and from above we see that simple encoding of differences for large values will result in loss of information --- v poor losses possible here 4 bits per pixel = values 0-15 unsigned, -7 – 7 signed so either quantise in multiples of 255/max vale or massive overflow!!

12 MARKS --- UNSEEN

3 (a) What are the major factors that need to be considered when considering what storage requirements are necessary for Multimedia Systems?

Major factors:

Large volume of date
Real time delivery
Data format
Storage Medium
Retrieval mechanisms

8 MARKS --- Unseen/applied bookwork

(b) What is RAID technology and what advantages does it offer as a medium for the storage and delivery of large data?

RAID --- Redundant Array of Inexpensive Disks

Offers:

Affordable alternative to mass storage
High throughput and reliability

RAID System:

Set of disk drives viewed by user as one or more logical drives
Data may be distributed across drives
Redundancy added in order to allow for disk failure

4 MARKS --- BOOKWORK

(c) Briefly explain the eight levels of RAID functionality .

Level 0 – Disk Striping --- distributing data across multiple drives
Level 1 – Disk Mirroring --- Fault tolerancing
Level 2 – Bit Interleaving and HEC Parity

Level 3 - Bit Interleaving with XOR Parity
Level 4 – Block Interleaving with XOR Parity
Level 5 - Block Interleaving with Parity Distribution
Level 6 – Fault Tolerant System --- Error recovery
Level 7 – Heterogeneuos System --- Fast access across whole system

8 MARKS --- BOOKWORK

(d) A digital video file is 40 Mb in size. The disk subsystem has four drives and the controller is designed to support read and write onto each drive, concurrently. The digital video stored using the disk striping concept. A block size of 8 Kb is used for each I/O operation.

(i) What is the performance improvement in sequentially   reading the complete file when compared to a single drive subsystem?

We have 5120 segments to write to RAID disks. Given 4 disks we have 1280 actual I/Os to perform

On 1 drive we clearly have 5120 operations to perform.

(ii) What is the percentage performance improvement expressed as the number of physical I/O operations to be executed in on the RAID and single drive systems?

The improvement is
(5120 – 1280)/1280*100 = 300%. Obvious given 4 concurrent drives and RAID!!

8 MARKS --- UNSEEN

4 (a) Give a definition of Virtual Reality.

Virtual Reality provides a human interface such that computers and its devices create a sensory environment which is dynamically controlled by the actions of the individual, so that the environment appears real to the user.

2 MARKS --- BOOKWORK

(b) What specialised input and output devices have been developed for Virtual Reality? Describe each device briefly.

Head Mounted Displays
Glove input devices
Body Suits

Three dimensional auditory displays
Stereoscopic computer display
Touch feedback gloves/suits

8 MARKS --- BOOKWORK

(c) Virtual Reality is not only for entertainment. How can Virtual Reality help in professional environments? Write a brief essay discussing this topic. Your answer may include current applications in this or address future avenues for this application of the technology.

Open ended anwer could embrace any of the following plus many more

Advertising
Architecture/Science – exploration of buildings or other objects
Simulations
Cooperative Work environments
Meeting Room Telepresence

14 MARKS --- Unseen, discussion of points made in lectures/bookwork etc.

Multimedia IGDS MSc Exam 2000

Setter: ADM
Checker: ACJ

Additional Material:

SMIL Language Sheet

Answer 3 Questions out of 4

Time Allowed 2 Hours

1. (a) Why is data compression, including file compression, highly desirable for Multimedia activities? [2]

   (b) Briefly explain how entropy coding and transform coding techniques work for data compression, clearly identifying the differences between them. Illustrate your answer with a simple example of each type of encoding. [7]

   (c) (i) Show how you would use *Huffman coding* to encode the following set of tokens:

   BABACACADADABBCBABEBEDDABEEEBB

   How is this message transmitted when encoded? [7]

   (ii) How many bits are needed to transfer this coded message? What is its entropy? [3]

   (iii) What amendments are required to this coding technique if data is generated live or is otherwise not wholly available? Show how you could use this modified scheme by appending the tokens ADADA to the end of the above message. [5]

2 (a) Give a definition of a Multimedia Authoring System. What key features should such a system provide? [2]

   (b) What Multimedia Authoring paradigms exist? Describe each paradigm briefly. [8]

   (c) You have been asked to provide a Multimedia presentation that can support media in both English and French. You have been given a sequence of 10 images and a single 50 second digitised audio soundtrack in both languages.

Each image should be mapped over consecutive 5 second fragments of the audio. All images are of the same 500x500 pixel dimension.

Describe, giving suitable code fragments, how you would assemble such a presentation using SMIL. Your solution should cover all aspects of the SMIL presentation. [14]

3. (a) What is meant by the Quality of Service of a multimedia application? [2]

(b) What major factors affect the Quality of Service of a multimedia application? [8]

(c) Perform a CORR analysis on the following Traffic scheduling problem:

A scheduler has an allocation cycle time of length 5 seconds and is serving 3 connections. The connections have rates of 2.5, 1.5 and 1 cells per cycle respectively. You should assume that all three connections are initially *backlogged*. What is the ideal delivery rate where fractional slots can be allocated? How should the connections be scheduled using CORR? [14]

4. (a) What is MIDI? How is a basic MIDI message structured? [6]

(b) In what ways can MIDI be used effectively in Multimedia Applications, as opposed to strictly musical applications? [8]

(c) How can MIDI be used with modern data compression techniques? Briefly describe how such compression techniques may be implemented? [10]

Multimedia IGDS MSc Exam 2000 SOLUTIONS

Setter: ADM
Checker: ACJ

Answer 3 Questions out of 4

Time Allowed 2 Hours

1. *(a)* Why is data compression, including file compression, highly desirable for Multimedia activities?

Multimedia files are very large therefore for storage, file transfer etc. file sizes need to be reduced. Text and other files may also be encoded/compressed for email and other applications.

2 MARKS --- BOOKWORK

(b) *Briefly explain, clearly identifying the differences between them, how entropy coding and transform coding techniques work for data compression. Illustrate your answer with a simple example of each type.*

Compression can be categorised in two broad ways:

Lossless Compression
-- where data is compressed and can be reconstituted (uncompressed) without loss of detail or information. These are referred to as bit-preserving or reversible compression systems also.
Lossy Compression
-- where the aim is to obtain the best possible fidelity for a given bit-rate or minimizing the bit-rate to achieve a given fidelity measure. Video and audio compression techniques are most suited to this form of compression.

Lossless compression frequently involves some form of entropy encoding and are based in information theoretic techniques

Lossy compression use source encoding techniques that may involve transform encoding, differential encoding or vector quantisation.

ENTROPY METHODS:

The entropy of an information source S is defined as:

$H(S) = SUM_I (P_I \, Log_2 (1/P_I)$

where P_I is the probability that symbol S_I in S will occur.

$Log_2 (1/P_I)$ indicates the amount of information contained in S_I, i.e., the number of bits needed to code S_I.

Encoding for the Shannon-Fano Algorithm:

A top-down approach

1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.

3 Recursively divide into two parts, each with approx. same number of counts.

(Huffman algorithm also valid indicated below)

A simple transform coding example

A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. Take top left pixel as the base value for the block, pixel A.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. B-A, C-A, D-A.
3. Store the base pixel and the differences as the values of the transform.

Given the above we can easily for the forward transform:

and the inverse transform is trivial

The above transform scheme may be used to compress data by exploiting redundancy in the data:

Any Redundancy in the data has been transformed to values, Xi. So We can compress the data by using fewer bits to represent the differences. I.e if we
use 8 bits per pixel then the 2x2 block uses 32 bits/ If we keep 8 bits for the base pixel, X0, and assign 4 bits for each difference then we only use 20 bits.
Which is better than an average 5 bits/pixel

7 MARKS --- BOOKWORK

(c) (*i) Show how you would use Huffman coding to encode the following set of tokens:*

BABACACADADABBCBABEBEDDABEEEBB

How is this message transmitted when encoded?

The Huffman algorithm is now briefly summarised:

1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).

2. Repeat until the OPEN list has only one node left:

(a) From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.

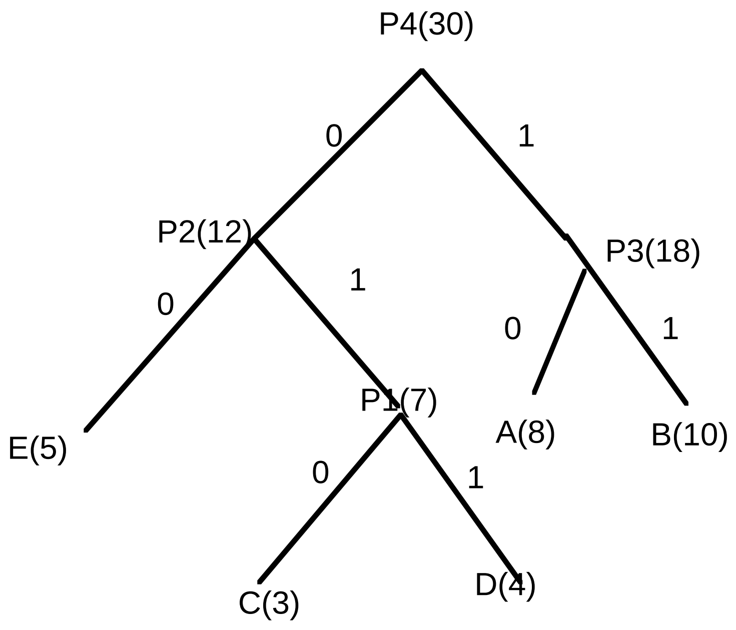
(b) Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.

(c) Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.

Symbol	Count	OPEN (1)	OPEN (2)	OPEN (3)
A	8			18
B	10			-
C	3	7	12	
D	4	-		
E	5		-	
Total	30			

8 indicate merge node with other node with number in column

Finished Huffman Tree:



Symbol	Code
A	10
B	11
C	010
D	011
E	00

How is this message transmitted when encoded?

Send code book and then bit code for each symbol.

7 Marks --- UNSEEN

(ii) How many bits are needed transfer this coded message and what is its Entropy?

Symbol	Count	Subtotal # of bits
A	8	16
B	10	20
C	3	9
D	4	12
E	5	10

Total Number bits (excluding code book) = 62

Entropy = 62/30 = 2.06667

8 MARKS --- UNSEEN

(iii) What amendments are required to this coding technique if data is generated live or is otherwise not wholly available? Show how you could use this modified scheme by appending the tokens ADADA *to the end of the above message.*

Adaptive method needed:

Basic idea (encoding)

```
Initialize_model();
 while ((c = getc (input)) != eof)
   {
     encode (c, output);
     update_model (c);
   }
```

So encode message as before:

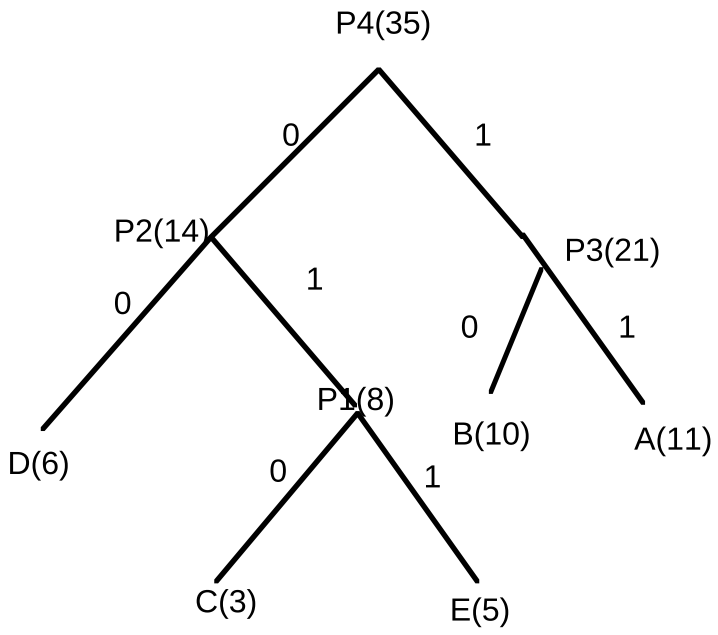
A= 01 D = 0000

So addd stream:

01000001000001

Modify Tree:

Symbol	Count	OPEN (1)	OPEN (2)	OPEN (3)
A	11			21
B	10			-
C	3	8	14	
D	6		-	
E	5	-		



5 Marks --- UNSEEN

2 (a) Give a definition of a Multimedia Authoring System. What key features should such a system provide?

An Authoring System is a program which has pre-programmed elements for the development of interactive multimedia software titles.
Authoring systems vary widely in orientation, capabilities, and learning curve.

There is no such thing (at this time) as a completely point-and-click automated authoring system; some knowledge of heuristic thinking and algorithm design is necessary.

Authoring is basically just a speeded-up form of programming --- VISUAL PROGRAMMING; you don't need to know the intricacies of a programming language, or worse, an API, but you do need to understand how programs work.

2 MARKS ---- BOOKWORK

(b) *What Multimedia Authoring paradigms exist? Describe each paradigm briefly.*

There are various paradigms, including:

<u>Scripting Language</u>

The Scripting paradigm is the authoring method closest in form to traditional programming. The paradigm is that of a programming language, which specifies (by filename) multimedia elements, sequencing, hotspots, synchronization, etc. A powerful, object-oriented scripting language is usually the centerpiece of such a system; in-program editing of elements (still graphics, video, audio, etc.) tends to be minimal or non-existent. Scripting languages do vary; check out how much the language is object-based or object-oriented. The scripting paradigm tends to be longer in development time (it takes longer to code an individual interaction), but generally more powerful interactivity is possible. Since most Scripting languages are interpreted, instead of compiled, the runtime speed gains over other authoring methods are minimal.

The media handling can vary widely; check out your system with your contributing package formats carefully. The Apple’s HyperTalk for HyperCard, Assymetrix’s

OpenScript for ToolBook and Lingo scripting language of Macromedia Director are examples of a Multimedia scripting language.

Here is an example lingo script to jump to a frame

```
glob
al
gNav
```

```
                                                        Spri
                                                        te
on exitFrame
go the frame
play sprite gNavSprite
end
```

Iconic/Flow Control

This tends to be the speediest (in development time) authoring style; it is best suited for rapid prototyping and short-development time projects. Many of these tools are also optimized for developing Computer-Based Training (CBT). The core of the paradigm is the Icon Palette, containing the possible functions/interactions of a program, and the Flow Line, which shows the actual links between the icons. These programs tend to be the slowest runtimes, because each interaction carries with it all of its possible permutations; the higher end packages, such as Authorware or IconAuthor, are extremely powerful and suffer least from runtime speed problems.

Frame

The Frame paradigm is similar to the Iconic/Flow Control paradigm in that it usually incorporates an icon palette; however, the links drawn between icons are conceptual and do not always represent the actual flow of the program. This is a very fast development system, but requires a good auto-debugging function, as it is visually un-debuggable. The best of these have bundled compiled-language scripting, such as Quest (whose scripting language is C) or Apple Media Kit.

Card/Scripting

The Card/Scripting paradigm provides a great deal of power (via the incorporated scripting language) but suffers from the index-card structure. It is excellently suitedfor Hypertext applications, and supremely suited for navigation intensive (a la Cyan's "MYST" game) applications. Such programs are easily extensible via XCMDs andDLLs; they are widely used for shareware applications. The best applications allow all objects (including individual graphic elements) to be scripted; many entertainment applications are prototyped in a card/scripting system prior to compiled-language coding.

Cast/Score/Scripting

The Cast/Score/Scripting paradigm uses a music score as its primary authoring metaphor; the synchronous elements are shown in various horizontal tracks withsimultaneity shown via the vertical columns. The true power of this metaphor lies in the ability to script the behavior of each of the cast members. The most popularmember of this paradigm is Director, which is used in the creation of many commercial applications. These programs are best suited for animation-intensive orsynchronized media applications; they are easily extensible to handle other functions (such as hypertext) via XOBJs, XCMDs, and DLLs.

Macromedia Director uses this .

Hierarchical Object

The Hierarchical Object paradigm uses a object metaphor (like OOP) which is visually represented by embedded objects and iconic properties. Although the learning curve is non-trivial, the visual representation of objects can make very complicated constructions possible.

Hypermedia Linkage

The Hypermedia Linkage paradigm is similar to the Frame paradigm in that it shows conceptual links between elements; however, it lacks the Frame paradigm's visual linkage metaphor.

Tagging

The Tagging paradigm uses tags in text files (for instance, SGML/HTML, SMIL (Synchronised Media Integration Language), VRML, 3DML and WinHelp) to link pages, provide interactivity and integrate multimedia elements.

8 Marks --- BOOKWORK

(c) You have been asked to provide a Multimedia presentation that can support media in both English and French. You have been given a sequence of 10 images and a single 50 second digitised audio soundtrack in both languages. Each Image should be mapped over consecutive 5 second fragments of the audio. All Images are of the same 500x500 pixel dimension.

Describe, giving suitable code fragments, how you would assemble such a presentation using SMIL. Your solution should cover all aspects of the SMIL presentation

```
<smil>
            <head>
             <layout>
              <root-layout height="500" width="500" background-
color="#000000" title="MultiLingual"/>
              <region id="image1" width="500" height="500" top="0"
left="0" background-color="#000000" z-index="1" />
              <region id="image2" width="500" height="500" top="0"
left="0" background-color="#000000" z-index="1" />
……….
            </layout>
             </head>

<body>
      <par>
             <switch>
            <!-- English only -->
             < audio system-language="en" src ="english.au" />

               <!-- French only -->
               <audio system-language="fr" src ="francais.au" />
            </switch>

<seq>
              <img src="image1.jpg" region="image1" begin="0.00s"
dur="5.00s" />
              <img src="image2.jpg" region="image2" begin="5.00s"
dur="5.00s" />

…….

</seq>
</par>

</body>
</smil>
```

14 Marks ---- UNSEEN

3. (a) *What is meant by the Quality of Service of a multimedia application?*

Quality of Service (Qos) DEFN:

- this is basically a collection of parameters that relate to a sequence as seen at the source and as seen at the destination of a multimedia presentation.
-

2 Marks --- BOOKWORK

(b) *What major factors affect the Quality of Service of a multimedia application?*

The QoS measure probably is the ultimate measure of a multimedia system. Four essential parameters are:

- Bandwidth -- capacity of the transfer mechanism between source and destination.
- Delay -- the time a multimedia unit spends in tranmission from sourse to destination.
- Delay Jitter -- Variation in delay delivery of data
- Loss Probability -- the ratio of units of information that an application can afford to lose.

- Hardware problems:
  - Network Connection
  - Traffic Analysis --- scheduling, buffer design, congestion control and synchronisation

8 Marks -- BOOKWORK

(c) *Perform a CORR analysis on the following Traffic scheduling problem:*

A scheduler has an allocation cycle time of length 5 seconds and is serving 3 connections. The connections have rates of 2.5, 1.5 and 1 cells per cycle respectively. You should assume that all three connections are initially backlogged. What is the ideal delivery rate where fractional slots can be allocated? How should the connections be scheduled using CORR?

BASIC CORR ALGORITHM:

time-line divide and allocate policy moderated by a queuing schedule. Distributes excess bandwidth over active connections.

- The Maximum length of allocation cycle is T
- Max. No. of cells transmitted in on cycle is T -- assuming cell transmission time is unit of time
- At time of admission, each connetion Ci is allocated a Rate Ri ---
- Ri s real numbers.
- *Scheduling Goal*: To allocate each connection Ciclose to Ri
- slots in each cycle.

- Excatly Ri slots of a long time frame

- *3 Event CORR Scheduler*: Asynchronous ~~~
    - Initilaize --- invoked for each new admitted connection
        - add connection to list Ci .
        - Sort Ci in decreasing Ri .
    - Enqueue--- Activate on arrival of a packet
        - Put packet in appropriate connection queue
        - update cell count of connection

    - Dispatch--- invoked at begining of busy periods
        - Ensures that connections cannot swamp othersa

CORR Algorithm:

For each connetion Ci scheduler:

- maintains a separate queue, ni

- keeps count of the waiting cells
- holds the number of current slots credited, ri .

Slot allocation divided into 2 cycles:

- Major Cycle --- integral requrement of each connection satisfied first
    - Slots left over assigned to minor cycle.
- Minor Cycle --- Elligible connections assigned allocated a full slot each in minor cycle (can't have fraction slot allocations!!)
-

At the begining of a busy period ({\em Dispatch})

- All ri 's set to zero
- New cycle started for each new cycle,

1. current number of unallocated slots T is initialised to T.
2. For each major cycle, cycle through each Ci update ri to ri + Ri

   - If number of cells queued, ni < ri set ri to n
   - Dispatch minumum of T and ri cells from C_{i}
3. A minor cycle starts with slots left over from preceding major cycle
   - `cylce through connection list, if slots are left dispatch connection

   - IF AND ONLY IF (a) packets queued or (b) ri > 0
   - if t =0 or no connection end cycle

SOLUTION:

CYCLE LENGTH T = 5

RATES = CELLS PER CYCLE
R1 = 2.5, R2 = 1.5, R3 = 1

<u>FIRST MAJOR CYCLE:</u>

[R1] = 2, [R2] = 1, [R3] = 1

1ST MINOR Cycle:
r1 = 0.5, r2 = 0.5, r3 = 0

SLOT left over for C1
Update ri
r1 = -0.5, r2 = 0.5, r3 = 0

<u>Second MAJOR CYCLE:</u>

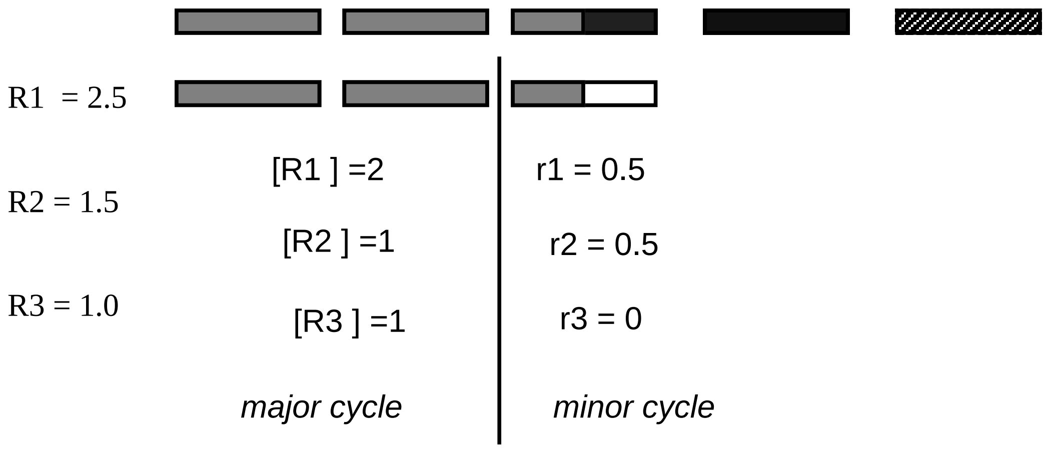
Update Ri = ri + Ri

So R1 = 2.0, R2 = 2.0, R3 = 1.0

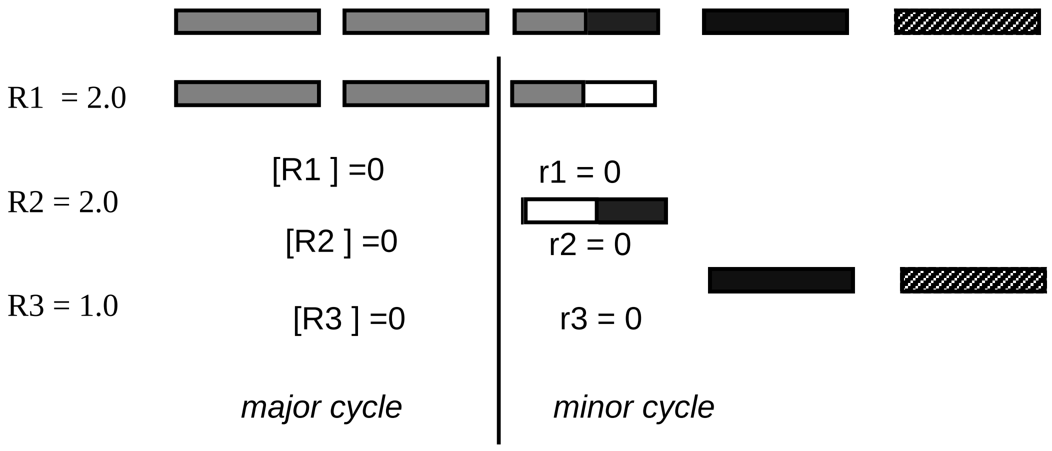
All integer so satisfy MAJOR CYCLE

Schedule is:

CYCLE 1



CYCLE 2



3 slots r1, 2 slots r2 1 slots R3 (as expected?)

14 Marks -- UNSEEN

4. (a) *What is MIDI? How is a basic MIDI message structured?*

MIDI: a protocol that enables computer, synthesizers, keyboards, and other musical or (even) multimedia devices to communicate with each other.

MIDI MESSAGE:

- MIDI message includes a status byte and up to two data bytes.
    - Status byte

      The most significant bit of status byte is set to 1.
    - The 4 low-order bits identify which channel it belongs to (four bits produce 16 possible channels).
    - The 3 remaining bits identify the message.
- The most significant bit of data byte is set to 0.

4 Marks --- Bookwork

(b) *In what ways can MIDI be used effectively in Multimedia Applications, as opposed to strictly musical applications*?

Many Application:

Low Bandwidth/(Low Quality?) Music on Web, Quicktime etc supports Midi musical instrument set
Sound Effectts --- Low Bandwidth alternative to audio samples, Sound Set part of GM soundset
Control of external devices --- e.g Synchronistaion of Video and Audio (SMPTE), Midi System Exclusive, AUDIO RECORDERS, SAMPLERS
Control of synthesis --- envelope control etc
MPEG 4 Compression control --- see Part (c)
Digital Audio

8 Marks --- Applied Bookwork: Discussion of Information mentioned in Notes/Lectures.

(c) How can MIDI be used with modern data compression techniques? Briefly describe how such compression techniques may be implemented?

- We have seen the need for compression already in Digital Audio -- Large Data Files

- Basic Ideas of compression (see next Chapter) used as integral part of audio format -- MP3, real audio etc.

- Mpeg-4 audio -- actually combines compression synthesis and midi to have a massive impact on compression.

- Midi, Synthesis encode what note to play and how to play it with a small number of parameters -- Much greater reduction than simply having some encoded bits of audio.

- Responsibility to create audio delegated to generation side.

MPEG-4 comprises of 6 Structured Audio tools are:

- SAOL the Structured Audio Orchestra Language

- SASL the Structured Audio Score Language
- SASBF the Structured Audio Sample Bank Format
- a set of MIDI semantics which describes how to control SAOL with MIDI
- a scheduler which describes how to take the above parts and create sound
- the AudioBIFS part of BIFS, which lets you make audio soundtracks in MPEG-4 using a variety of tools and effects-processing techniques

MIDI IS the control language for the synthesis part:

- As well as controlling synthesis with SASL scripts, it can be controlled with MIDI files and scores in MPEG-4. MIDI is today's most commonly used representation for music score data, and many sophisticated authoring tools (such as sequencers) work with MIDI.

- The MIDI syntax is external to the MPEG-4 Structured Audio standard; only references to the MIDI Manufacturers Association's definition in the standard. But in order to make the MIDI controls work right in the MPEG context, some semantics (what the instructions "mean") have been redefined in MPEG-4. The new semantics are carefully defined as part of the MPEG-4 specification.

  10 Marks --- Basic Ideas mentioned in lectures. Details application of Lecture notes material

BSc Multimedia (CM0340) Assessed Exercises

- Exercise 1 (PDF)
- Exercise 2 (PDF)

BSc Multimedia (CM0340) Lab Worksheets

- Lab Worksheet 1 (PDF)

Multimedia
Module No: CM0340
Assessed Exercise 1:
Macromedia Director/Lingo Scripting

Dr. D. Marshall

October 3, 2001

Submission of Coursework

Please hand in the exercise below as part of your *assessed coursework* for this module. Only the first question below is assessed.

- **Hand in Date**: Hand in a hard copy of the exercise in the Lecture (11.10 AM) on Monday Nov 12rd
- In order to verify that you interactive programs work you must also get you exercise submission sheet signed by the tutor to *verify* that the programs work according to specification.
- The tutor is only guaranteed to be available to sign at Multimedia Laboratory Sessions.
- The coursework is worth 50% of the coursework component,*i.e.* 10 Marks for the whole module.

Submission Details

The exercises involve developing a Macromedia Director Movie. You hard copy solution must convey all the appropriate information to demonstrate and explain your programming philosophy.

You solution must include:

- An overview of you program design and implementation.
- A complete list of all Cast members and their use, and
- Any lingo scripts and their use.
- All lingo scripts source must be given.

In the exercise high marks will be awarded for solutions that provide good solutions to the basic problems and also some novel extensions/additional features.

Assessed Exercises

1. Implement an interface to play the game checkers in a Macromedia Director Movie:
    - You need only concern yourself with providing an adequate interface — There are no real marks for fancy graphics. A basic looking Interface is all that is required.
    - Your Movie need only check for valid moves — Your Movie **is not required** to play the game.
    - Your movie should however be able to replay the most recent game

Not Assessed Coursework

The following exercise does not form part of your assessed coursework.

1. Write a Macromedia Director (or Flash) Movie that given a valid URL of an HTML file:
    - Scans the html for the first occurence of a director movie (you could look for `.dir` extension for example)
    - Having found a file stream the file to your movie.
    - You should inform the user if a file is not found.
    - You should be able to play a file from any valid URL:

  - A file from a local disk

- Example director files are freely available on the Internet, for example:
  - *www.shockrave.com*
  - *http://www.macromedia.com/support/director/* Director developer's centre

Multimedia
Module No: CM0340
Assessed Exercise 2: SMIL/Quicktime

Dr. D. Marshall

Submission of Coursework

Please hand in the exercises below as part of your assessed coursework for this module.

- **Hand out Date**: Monday 12th Nov, 2001.
- **Hand in Date**: Friday Dec 14th, 2001.
- The Hard copy should be handed in between to Dr. A.D. Marshall in his office (Room S 2.20) between 3.00-4.00 PM Friday 14th December.
- In order to verify that you interactive programs work you must also get you exercise submission sheet signed by the tutor to *verify* that the programs work according to specification.
- The tutor is only guaranteed to be available to sign at Multimedia Laboratory Sessions.
- The coursework is worth 50% of the coursework component,*i.e.* 10 Marks for the whole module.

Submission Details

The exercises involve developing a Multimedia Presentation using SMIL and Quicktime.

You should submit a hard copy (as below) and mount your presentation on the local project web space.

Your submission must include:

- Complete SMIL code listings
- Demonstrable printouts of program output that clearly illustrates that the program works to specification.
- The hard copy **must** clearly indicate a URL to your *project web space* where the presentation should be mounted and it should remain there until the coursework is marked.
- A link from your *project web space* home page to the presentation is also required.

Aims and Objectives

The Aims and objectives of this exercise is to:

- Create a Multimedia Presentation using SMIL and Quicktime.
- To gain experience of developing and delivering a multimedia presentation
- To gain practical experience of SMIL
- To gain a practical experience of using Quicktime
- To investigate what Media formats may be supported by Quicktime and utilise these effectively in a Multimedia presentation.

Assessed Exercise

1. This exercise involves creating and delivering a multimedia presentation using SMIL and Quicktime.

   The topic of the presentation is *JPEG Compression*

   You may use any material to base your presentation on. You may for example use the lecture notes and online Lecture Web Pages:

   http://www.cs.cf.ac.uk/Dave/Multimedia

   for sample material.

   Your presentation however will be marked according the following criteria:

   - Cohesive and logical presentation of material
   - Effectiveness of the presentation in terms of teaching about the topic
   - Effective and varied use of Multimedia (simple text pages presentation is **not adequate**)
   - Presentation requirements (see below) satisfied.
   - SMIL requirements (see below) satisfied.
   - Quicktime requirements (see below) satisfied.

   The following requirements should be included in your presentation:

   Presentation :

   - The presentation must include a variety of multimedia. You will be marked accordingly on use of a good variety of multimedia. However, the presentation must be coherent as a whole. *I.e.* You there must be good justification of the use of the media.
   - The presentation **must** include *at least* one piece of animated or video multimedia. You should use Macromedia Flash to produce some animation and include it in Quicktime.

- The presentation **must** include some audio facility in the presentation also.
  Microphones are connected to the Macintosh G3/G4 PowerMacs in the Multimedia Lab (S 2.21a). The Microphones are not ordinary microphones: *They do not work on other platforms.*
  A variety of Macintosh applications can import sound. Flash can do it, Macromedia Soundedit is the main sound editing application available on the Macintoshes in the Multimedia Lab.
  You may of course use any computer facility available to you to create component parts of your multimedia presentation.

SMIL :

- Logical layout of multimedia within a SMIL presentation
- Smooth transitions between SMIL pages or different media.
- Good use of SMIL commands.
- Your SMIL presentation **must** include *at least* one alternative path or configuration within the presentation. *I.e.* You must use the SMIL `switch` statement and appropriate `test-attributes`. The exact nature of the use of this is left to your individual creativity.

Quicktime :

- You should investigates and aim to support as wide a range of multimedia that Quicktime supports that is appropriate for the presentation.
- You must be able to display *at least* one animation or video sequence. *Hint*: Quicktime supports Flash.

Multimedia
Module No: CM0340
Laboratory Worksheet 1:
Macromedia Director/Lingo Scripting, Flash

Dr. D. Marshall

October 6, 2000

Aims and Objectives

After working through this worksheet you should be familiar with:

- The basic operation of Director and Flash
- How to creat movies in Director.
- How to use Lingo Scripts
- Know a basic set of the Lingo Language
- Produce Web-based Director Movies

Director

1. Follow the guided tour of Director in the Director Manual/Online Director Help, Familiarise yourself with all the basic functions of Director:

    - Setting up Director Movies
    - Using Vector shapes
    - Creating and using Cast members

- Creating and Using Sprites
- Attaching behaviours
- Lingo: How does this enhance Director? How does one use Lingo? Examine some simple Lingo examples in Director.
- Familiarise yourself with all the basics of Director: Work Area, Score, Stage, Casts and Sprites Channels, Markers etc.

2. Run the guided tour Director movies in the `Director:Show me` folder. The Lingo movie is particularly relevant
3. Try Creating your own simple Banner as per the example in the guided tour of Director.
4. Prepare the Web banner for Web delivery with Aftershock.
5. Develop a Director presentation using Lingo *e.g* on that can Play a sound using Lingo, create basic interactive presentations opening Lingo scripts, or play a video clip in Lingo.
6. Write a Lingo script (attached to a Director Movie) that can detect when a single cast member has been moved on the stage. You should inform the user of a motion with an alert window prompt.
7. Write a lingo script that move a sprite randomly around the stage.
8. Write a lingo script attached to an appropriate means of data entry that can dynamically resize a sprite on the stage.
9. Write a lingo script attached to an appropriate means of data entry that can dynamically set simple paths of motion on a stage (e.g circular, square, simple curved paths
10. Write a Lingo script that can accept text from a given field and download and display data pointed to by a URL if valid URL is given.
11. Write a Lingo script that will scan through a given text of field cast member for an appropriate piece of text and print out the number lines in another cast member.
12. Write a Lingo script that will search through a given text of field cast member for an appropriate piece of text and print out the lines that match the search in another text cast member.

Flash

1. Follow the Flash Basics Chapter 1 in the Flash Manual/Online Flash Help, Familiarise yourself with all the basic functions of Flash:

   - How to create Flash Movies for the Web.
   - What the stage, timeline, frame, inspector concepts mean in Flash.
   - How to use layers, scenes and libraries.
   - How to import media.
   - How to deliver Flash movies in a Web presentation.

2. *Application:* Produce an animation where the words multimedia stream out of simple 2d (or 3d) box character by character.

3. *Application:* Produce a simple Web Banner.

4. *Application:* Include the Flash applications on a web page