REAL-TIME ENCODING OF LIVE RECONSTRUCTED MESH SEQUENCES FOR 3D TELE-IMMERSION

Rufael Mekuria[+], Dimitrios Alexiadis[*], Petros Daras[*] and Pablo Cesar[+]

[+]Centrum Wiskunde en Informatica
SciencePark 123 1098 XG Amsterdam
{r.n.mekuria, p.s.cesar}@cwi.nl

[*]Centre for Research & Technology - Hellas
6th km Xarilaou - Thermi, 57001, Thessaloniki
{dalexiad,daras}@iti.gr

ABSTRACT

3D tele-immersion systems based on live reconstructed geometry, 3D-based videoconferencing, require low-delay encoding and small bandwidth usage. Existing triangle mesh compression methods are not that useful, as they are generally optimized for "downloadable" 3D objects, with high encoding delays. In this paper we experiment with a real-time reconstruction system that outputs a sequence of triangular meshes. Explorative analysis on the mesh data shows that regularities happen in the connectivity and geometric data. Based on this, we develop a module that achieves a compact representation for each reconstructed mesh. Comparison with the SC3DMC standard on the live reconstructed mesh sequences reveals an encoding speedup of over 20 times at comparable rate/distortion levels. By exploiting the properties of the capture and reconstruction process, we provide an optimized encoding mechanism that enables real-time encoding. This paper discusses some potential directions to better handle the large byte size of live reconstructed 3D triangle mesh sequences with stringent constraints on bandwidth, delay and computational complexity.

Index Terms— 3D Mesh Reconstruction, Triangular Mesh Compression, Interactive Service, 3D Tele immersion

1. INTRODUCTION

Real-Time transmission of live triangular mesh reconstructions opens novel possibilities for 3D tele-immersion (3DTI) as it enables to mix real and virtual content. For example, instead of a person controlling an avatar or a model inside a virtual world, the person can be ported directly into the virtual world as a 3D triangular mesh sequence that is reconstructed on the fly, including all its geometrical details and movements. Figure 1 shows a reconstructed human "immersed" in a virtual world. The 3D mesh representation has advantages as it offers flexibility in rendering (shading, rendering with stereo or multiple views), integration with virtual worlds (collision detection, navigation) and is commonly supported in games and by modern graphics cards. Figure 2 illustrates a typical pipeline, where a 3D reconstruction process *R* generates a sequence of triangular meshes M^0 , M^1 … M^n (based on image and depth data from multiple cameras). These triangle meshes are compressed and transmitted over the IP network to a remote site. The remote site can be a participant in a multi-party conferencing session or a virtual world application server that renders objects together and forwards rendered images as videos to light weight viewer terminals. A current challenge is that the 3D mesh representation is less well supported by standardized compression and transmission methods. Current triangle mesh codecs introduce significant encoding delays that decrease the interactivity and frame rate in an interactive scenario.

Figure 1: A virtual world with a 3D human mesh, reconstructed in real-time from multiple depth cameras (right), and a virtual character (left).

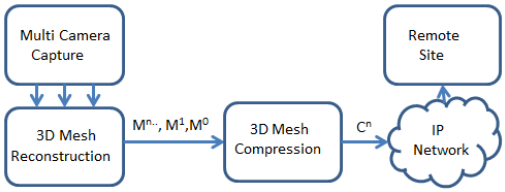

Figure 2 3D reconstruction generating a sequence of triangular mesh frames

The 3D mesh sequences reconstructed on-the-fly are generally temporally incoherent mesh sequence (TIMS) [3], i.e. the number of vertices and connectivity vary from frame to frame. If MF = $\{M^i = (V^i, E^i, F^i), i = 0...n\}$ is a

mesh sequence, $|V^i|$ is not constant over i. Therefore, compression techniques that have been developed for animated mesh sequences, i.e. temporally coherent mesh sequence (TCMS) [3], are not applicable. For TIMS, only static (intra) frame triangular mesh compression is available (compressing each frame separately). A framework that handles such TCMS for example is the MPEG-AFX (Animated frameworks extension) [9].

In this paper we explore the properties of TIMS data resulting from a real-time reconstruction process R. We exploit such properties in order to achieve a bandwidth efficient representation. The main motivation of our work is to reduce the computational delay for encoding, introduced by the current state of the art triangle mesh encoders. For the reconstruction system under investigation we deploy an encoder/decoder based on differential coding with local quantization. Furthermore, we take advantage of the repetitions in the connectivity data, providing a connectivity mechanism similar to run-length encoding. The developed codec can handle mesh data at only slightly lower rate/distortion than the Triangle Fan Encoder from MPEG, but instead ensuring real-time encoding times (20 time speedup on the test data from R). We highlight some research directions that we are investigating in order to handle the large size of live reconstructed geometry data. This is useful for next-generation 3D tele-immersion / tele-presence services that may be based on live captured mesh geometry.

The structure of the paper is as follows. In section 2 we present some related work. In section 3 we revisit some of the implementation details of the state of art real-time 3D reconstruction component (R) that generates a temporal incoherent triangle mesh sequence (TIMS). In section 4 we investigate some of the properties of the triangle mesh data resulting from R and develop our compressed representation. In section 5 we present the experimental results that compare the developed method with regular triangle mesh compression methods from MPEG. We highlight alternative directions to handle live reconstructed TIMS in section 6. Conclusions based on this work are presented in section 7.

2. RELATED WORK

Techniques for reconstruction of triangle meshes from range (depth) images have been developed during previous decades in [1] and [2]. A recent implementation in [7] introduces various optimizations based on these techniques and uses consumer grade depth camera's to acquire the depth images. This system makes it possible to reconstruct, in real-time, triangle mesh geometry with equipment affordable to the general public. An interesting next step, is to investigate if these reconstructions are useful for communication between humans similar as in tele-presence. This is challenging because a medium quality reconstructed triangle mesh from [7] contains around 50,000 vertices with normal, color, position data, and about 100,00 triangles and therefore consumes about 3 Megabytes per frame (uncompressed). For TIMS at 8 fps this would consume 192 Mbit which is comparable to more than 13 MPEG-4 compressed HDTV streams (8-15 Mbit/s per stream). It is evident, that if TIMS will be used for tele-presence, efficient compression schemes need to be developed.

Many different compression methods for triangle meshes exist (a survey is presented in [4]), but these methods introduce large encoding complexity. They have been developed with offline encoding of static objects in mind, possibly downloaded from a server or distributed on a disc.

For example the recent mpeg standard TFAN aims at real-time decoding instead of encoding[5]. Complexity is often introduced in the encoding of the connectivity that can involve many searches and/or re-orderings of the edge list to obtain an efficient representation. Connectivity representations such as Triangle FAN's[5] or other extensions of the initial triangle strip concept allow subsequent triangles to share vertices and therefore enable efficient transport of the vertex data from the CPU towards the GPU. Apart from disc storage, this was one of the original aims of triangle mesh compression.

The methods developed in the literature have generally been tested on models available in online repositories, which allow fair comparisons between codecs.

However, for real-time reconstructed TIMS based on fixed reconstruction process R, this may not lead to the optimal result, both in terms of compression rate and encoding delay. Therefore, in this paper we investigate properties of TIMS resulting from a reconstruction process R and develop efficient real-time compression.

3. 3D RECONSTRUCTION PROCESS

In this section we shortly present the employed real-time frame-by-frame 3D reconstruction method, from multiple consumer depth cameras. The method shares similarities with [7] and is based on the notion of Step Discontinuity Constrained Triangulation (SDCT), i.e. terrain triangulation on the depth 2D image plane.

3.1. Capturing setup and calibration

A calibrated capturing setup with five RGB-Depth sensors (Kinects) was implemented. The sensors are connected on a single host PC with high processing power, as well as a CUDA enabled GPU. One sensor is placed horizontally at a height of 1.30m, to capture the front upper body, while the remaining four are placed vertically, at a height of approximately 1.80m to capture the whole human body. The sensors are positioned on a circle of diameter 3.60m, all pointing to the centre of the working volume, introducing a circular "active" region of diameter approx. 2.40m

In order to fully calibrate each single Kinect, we used the method of [8], which simultaneously estimates the depth and the RGB camera intrinsic parameters, the relative pose between them, as well as a depth (disparity) distortion model. With respect to the external calibration of the

multiple Kinects network, we use a custom-made calibration object with three intersecting large planar surfaces, which is moved inside the working volume and captured simultaneously by all sensors. For each captured frames, the planar surfaces are detected in the depth images. Then, a fast coarse pairwise calibration is realized, based on the normal vectors of the detected 3D planes, followed by minimization of the mean squared distance of the reconstructed points on the three planes with the corresponding planes in a reference camera;. Finally, a global (all-to-all cameras and for all frames) ICP optimization procedure is realized.

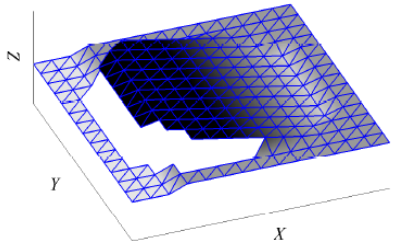

Figure 3. The idea of SDCT. Each 2 × 2 square neighborhood of a depth map is bisected into two triangles, unless the absolute difference of the neighbor depth values is greater than a predefined threshold (= 2cm for the presented results).

3.2. Reconstruction process

The overall reconstruction approach can be summarized as follows:

- **Pre-processing:** 1) A weak 2D bilateral filter is applied to the depth maps to reduce Kinect measurements noise. The spatial Gaussian kernel of the bilateral filter was selected with $s = 4$ pixels and the depth-difference Gaussian kernel with $s = 50$ mm. 2) Additionally, a binary "human silhouette" map is generated for each depth maps, by segmenting out the foreground object of interest (human). Background subtraction is realized by checking the absolute difference of the current frame from a "background" depth image, accumulated in multiple frames.
- **Triangulation:** From each depth map, we construct a mesh via SDCT, considering the depth values only inside the "human silhouette" map. The idea in SDCT is that depth measurements that are adjacent in the 2D depth image plane are assumed to be connected, unless their Euclidean 3D distance is higher than a predefined threshold (= 2cm for the presented results). The idea is illustrated in Figure 3. Since the spatial sampling step in X,Y is very small compared to this threshold, it is adequate to consider only the depth (Z) difference of the adjacent measurements, in order to save execution time. SDCT is formally summarized as follows: i) For each pixel $Z = D(u,v)$ of the depth map, consider its adjacent pixels at the right, right-bottom and bottom, $Z_r = D(u+1,v)$, $Z_b = D(u,v+1)$, $Z_{rb} = D(u+1,v+1)$, respectively; ii) If all depth distances $|Z - Z_r|$, $|Z - Z_b|$ and $|Z_r - Z_b|$ are below the threshold, generate a triangle by connecting the corresponding 3D points; iii) Similarly, if all $|Z_r - Z_b|$, $|Z_r - Z_{rb}|$ and $|Z_{rb} - Z_b|$ are below the threshold, generate a second triangle.

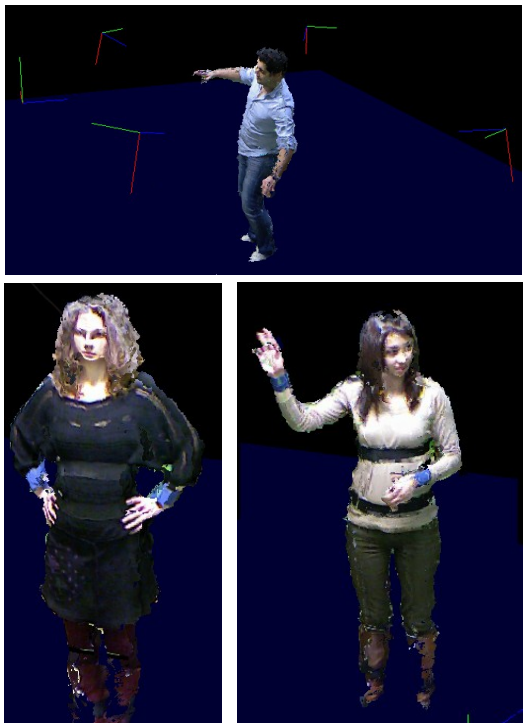

Figure 4 Examples of real-time reconstructions. In the upper row, the colored lines indicate the positions and orientations of the cameras.

- **Post geometry smoothing:** A fast mesh-smoothing operation is applied to the reconstructed mesh, which calculates the average position of each vertex with its connected neighbor vertices. Two iterations are used.
- **Weighted color mapping:** The RGB color of each vertex $\mathbf{v}$ is obtained as the weighted average of the pixel colors in all visible cameras that the vertex projects to. Visibility is inferred by comparing the actual Z distance of a vertex to a camera and the corresponding depth value observed by the camera. Let $\mathbf{n}_i$ denote the normal and $\mathbf{v}_i$ the vertex coordinates, with respect to the i-th camera's coordinate system. The inner product (angle) $w_i = \mathbf{n}_i \cdot \vec{v}_i$ is used for weighting the color observation in the i-th camera, where $\vec{v}_i = -\mathbf{v}_i / \|\mathbf{v}_i\|$. Practice showed the use of such a weighted color mapping scheme can significantly improve the visual quality.

4. REAL-TIME COMPRESSED REPRESENTATION

4.1. Data Exploration of real-time reconstructed TIMS

We explore the data output by the component in 5 different data settings. The first setting is with one Kinect reconstructing a person sitting in front a Kinect sensor at approximately 1m (130 cm max). This dataset (N=20 sample frames/models) contains around 70,000 vertices per reconstructed model with each vertex featuring 3D position coordinates, normals and colors. The second dataset is taken with persons a maximum distance of 300 cm away, with a set of 5 Kinects. The dataset represents the case of a small room in which a participant is reconstructed (5 Kinect low, high) while performing some activity. The high-resolution reconstruction data contains around 250,000 vertices per model (N=24 reconstructed models) and the low-resolution data approximately 28,000 points (with N=26 reconstructed models). The last two datasets tested feature reconstructions with one Kinect at maximum 300 cm distance (1 Kinect low, high). The high-resolution datasets contains about 65,000 vertices (N=26 reconstructions) and the low-resolution approximately 18,000 per mode (N=22 reconstructed models). Figure 5 shows the number of vertices in each Mesh M^i generated by the reconstruction process. It shows that the number of vertices $|V^i|$ changes and that therefore it is verified that a temporally incoherent mesh sequence (TIMS) is produced.

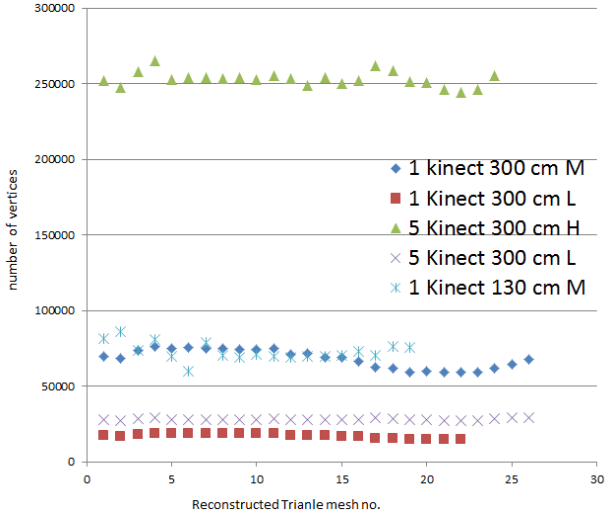

Figure 5 Number of Vertices in each reconstructed mesh in sequence

Figure 6 shows the relationship when vertices occur in the list V^i and when they are indexed in F^i. It can be observed that no outliers occur and that the triangles indexing *kth* vertex are generally clustered around the 2*kth* triangle in the list. This implies that relative indexing is possible and can be efficient.

Table 1 shows an example of some of the triangles from F^i. It can be seen that in column 1,2 and 3 that the difference between the second values is constant. This occurs in many parts of the reconstructed mesh, due to the triangulation process on the original depth images. We can exploit this by explicitly coding the regions where this occurs. Similar regularities were found in each of the live reconstructed datasets.

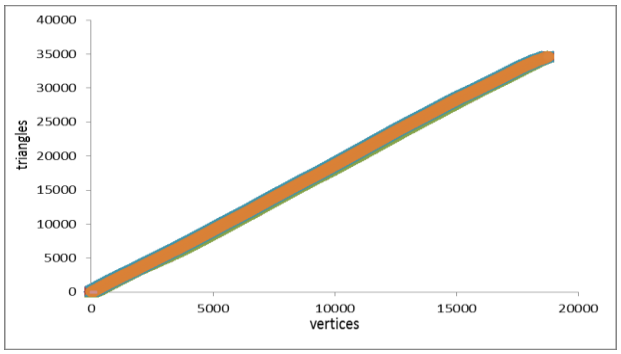

Figure 6 Vertex Indexing of different triangles

Table 1 Example regularity in reconstructed triangle mesh

1632 1520 1633
1520 1521 1633
1633 1521 1634
1521 1522 1634
1634 1522 1635
1522 1523 1635
1635 1523 1636
1523 1524 1636

4.2. TIMS Encoder

To exploit the properties of R we develop a coding module, the schematic is shown in Figure 7.

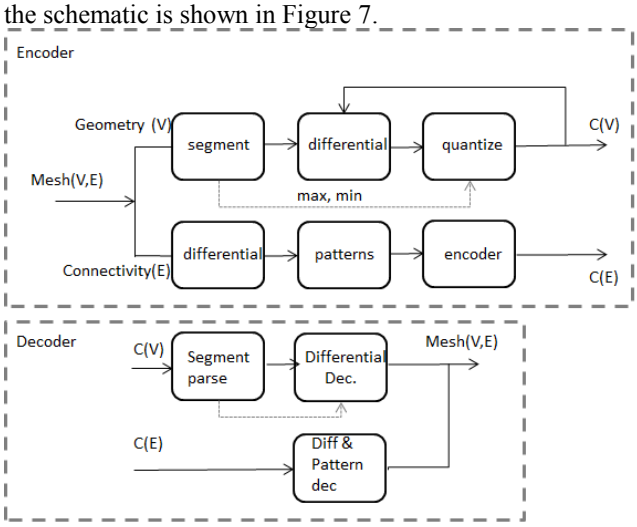

Figure 7 Encoder and Decoder schematics

The upper part of Figure 7 illustrates the schematics of the encoder. The encoder consists of two branches, one handling the connectivity data *Connectivity(E)* and one handling the geometric data *Geometry(V)*.

The geometry data (coordinates, colors and normals) are processed in blocks of 300-600 points and differentially encoded per column (i.e. color, normal and position sub-value). We *quantize* the differences locally, based on the range of values in each sub-value. Normally, 4 bits are assigned to quantize the differences, in case the values are constant, 0 bits are assigned.

Additionally, a separate threshold value *T* can be assigned that can limit the effects of coarse quantization in case *max* is large. So, basically, if the differential encoding results in error above *T* the value and the current index is stored in a separate block of data that is also added to C(V). The decoder can then decode these exactly stored values first, and use them to improve the differential decoding quality of these points at the cost of some extra storage space.

The compressed representation can be quickly decoded. Based on the starting values of the block and *max* that are stored in each block, the local quantization vector can be computed and differential decoding can be performed. The advantage of the proposed approach is that in different regions of the mesh different resolutions/level of detail are obtained based on the denseness of subsequent points in the list that are generally co-located. While other techniques can also achieve this, the extremely low computational load is desirable in this real-time application.

The lower part of the encoder block in Figure 7 handles the connectivity compression of the edges between vertices.

The module *differential* in Figure 7 computes the difference between indexes in subsequent triangles in the face list. If triangle [A B C] is followed by [B C D] in the list the difference would be computed as [A-B B-C C-D].

The module *patterns* in Figure 7 aims to find the regularities in the model connectivity. As such, patterns occur many times, they can be coded as a run (number of repetitions). Examples of sequence of difference are for examples 0 0 1 1 00 11 and so on. Table 2 shows the data structure that can store run. The run is stored as 5 integers: the type of pattern (mode), difference value 1, difference value 2, the start position (in the indexed face set list) and value which are the amount of repetitions (faces) coded in the run.

Table 2 Data Structure pattern_run

type	Field				
int	Mode	Diff1	Diff2	Start	value

In the encoder block, either the differential value is added to the compressed representation C(E), or the coded run data-structure is added. In the connectivity data generated by the reconstruction process, over 90% of the data can be coded in runs over length 32. More information and details on the encoding algorithm can be found in [10].

5. EXPERIMENTAL RESULTS

Figures 8-11 show the comparison results, where imtp denotes the proposed compression method, while tfan is the high end coder defined in the MPEG Standard SC3DMC[5] while SVA and QBCR are two low complexity codecs also defined within MPEG SC3DMC. Figure 8 shows the encoding (compression time), which is important if meshes are captured in real-time in an interactive session. Figure 9 shows the decompression time achieved with the different codecs. Figure 10 shows the compression gain achieved in the different settings, the performance is only slightly worse compared to the TFAN codec. Figure 11 shows the quality distortion caused by the different codecs measured on the dataset 1 Kinect, High 130 cm. We measured the root mean square distance from the original to the decoded model with the tool available in [6]. The distortion of a mesh D(M) is given as

$$D(M) = \sqrt{1/|V| \sum_{\forall v^i \in V_{original}} (dist\ (v_{i\_originab}, S_{decoded}))^2}$$

where |V| is the number of vertices in the mesh and dist(v,S) is the Euclidian distance between the original point and the reconstructed surface.

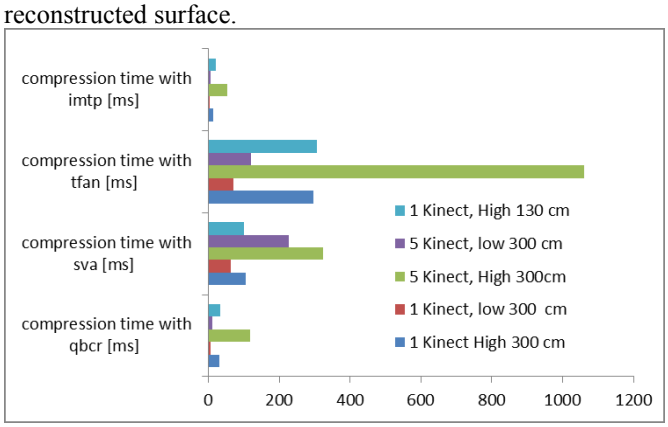

Figure 8 compression time achieved with different codecs

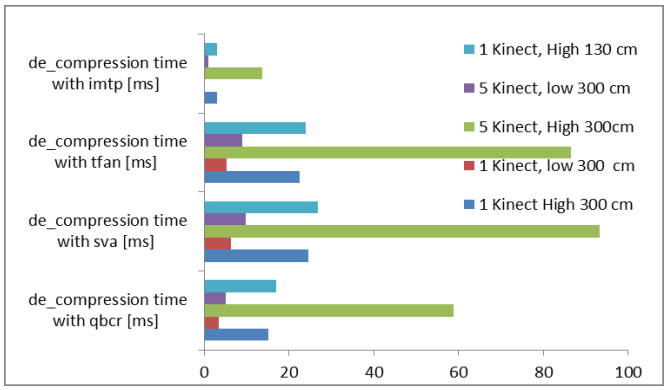

Figure 9 decompression time achieved with different codecs

Overall, with a simple coding scheme we show encoding speed increase of around 20x compared to the TFAN codec and a 10x speedup in decoding at only slightly larger size. This result holds for all the different settings/datasets that

were tested. The Machine used to obtain the results was an intel i7 desktop Machine (3.2 GhZ), with 8 GB of RAM. Generally, decoding and encoding times are below 20 ms enabling real-time pipelines suitable for 3D tele-presence.

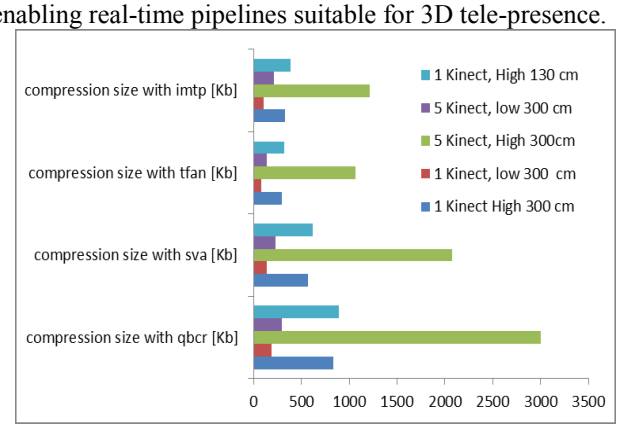

Figure 10 compression gain achieved with different coding options

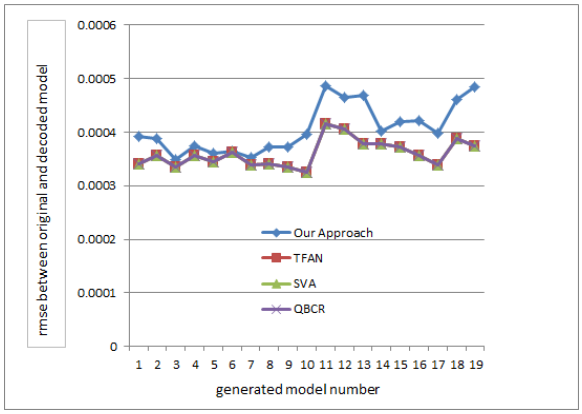

Figure 11 Distortion level obtained of the different codecs, measured with available tool [6]

6. DISCUSSION

This article explored a possibility for compressing live reconstructed mesh geometry sequences for use in multi-party conferencing and tele-immersion applications by taking advantage of specific properties of the reconstruction process *R*. Using this approach a low complexity encoder was developed. Alternative approaches include using regular triangle mesh codecs, which introduce delay and result in low frame rates, due to their computational complexity. Alternatively, sending all the depth images and running full 3D reconstruction at the receiver sites, is likely to increase the computational load on the receivers too much when the number of participants increases. Moreover, the encoding and transmission of the depth images will introduce additional delays. We believe that studying the data from live reconstruction modules, and modeling it can be used to realize low complexity encoding of the bandwidth heavy *TIMS* introduced by reconstruction process *R*. As 3D tele-immersive application are demanding in networking and computational resources, efficient low complexity encoding is critical. For the reconstruction module under investigation we can achieve a significant (real-time) speedup compared to state of the art mesh encoders. In further research we want to investigate more different reconstruction modules and use more advanced models to encode the TIMS data.

7. CONCLUSIONS & ACKNOWLEDGEMENT

This paper demonstrated a real-time 3D mesh reconstruction method combined with a lightweight compression method. It shows that lower encoding/decoding delays can be achieved compared to available MPEG-4 Mesh encoders. This is of importance for delay critical applications such as 3D conferencing and 3D tele-immersion using reconstructed mesh geometry. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. ICT-2011-7-287723 (REVERIE project).

REFERENCES

[1] G. Turk and M. Levoy. 1994. Zippered polygon meshes from range images. SIGGRAPH '94. pp. 311-318

[2] B. Curless and M. Levoy. 1996. A volumetric method for building complex models from range images. SIGGRAPH '96., pp. 303-312.

[3] R. Arcila, C. Cagniart, F. HéTroy, E. Boyer, and F. Dupont. 2013. Segmentation of temporal mesh sequences into rigidly moving components. *Graph. Models* 75, 1, Jan. 2013, pp. 10-22.

[4] Peng J., Kim C.S., C-C jay Kuo. Technologies for 3D Mesh Compression: A survey. Elsevier journal of visual comm. and image repr.(2005) pp. 688-733

[5] Mamou, K., Zaharia, T. and Prêteux, F. TFAN: A low complexity 3D mesh compression algorithm (2009),. Comp. Anim. Virtual Worlds, 20: 343–354.

[6] Aspert, N. Santa-Cruz D., Ebrahimi T., MESH: Measuring Error between Surfaces using the Hausdorff distance (2002), in Proceedings of the IEEE International Conference on Multimedia and Expo 2002

[7] D. Alexiadis, D. Zarpalas, and P. Daras, "Real-time, full 3-D reconstruction of moving foreground objects from multiple consumer depth cameras," IEEE Trans on Multimedia, vol. 15, pp. 339–358, Feb. 2013.

[8] D. Herrera, J. Kannala, and J. Heikkila, "Joint depth and color camera calibration with distortion correction," IEEE Trans Pattern Anal Mach Intell., vol. 34, 2012.

[9] M. Bourges-Sevenier and E.S. Jang, " An introduction to the MPEG-4 animation framework extension", IEEE Tr. on Circ. and Sys. for Video Technology, vol. 14

[10] R. Mekuria, M. Sanna, S. Asioli, E. Izquierdo, D.C.A Bulterman,. and Cesar, P. A 3D Tele-Immersion System Based on Live Captured Mesh Geometry. Proceedings of the *4th ACM Conference on Multimedia Systems* (MMSys 2013)