A 3D Tele-Immersion System Based on Live Captured Mesh Geometry

[1]Rufael Mekuria, [2]Michele Sanna, [2]Stefano Asioli, [2]Ebroul Izquierdo, [1]Dick C.A. Bulterman and [1]Pablo Cesar

[1]Centrum voor Wiskunde en Informatica
Science Park 123
1098 XG Amsterdam

[2]Queen Mary, University of London
Mile End Road, E1
4NS London UK

r.n.mekuria@cwi.nl, michele.sanna@eecs.qmul.ac.uk, stefano.asioli@eecs.qmul.ac.uk, ebroul.izquierdo@eecs.qmul.ac.uk, dick.bulterman@cwi.nl, p.s.cesar@cwi.nl

ABSTRACT

3D Tele-immersion enables participants in remote locations to share, in real-time, an activity. It offers users natural interactivity and immersive experiences, but it challenges current networking solutions. Work in the past has mainly focused on the efficient delivery of image-based 3D videos and on the realistic rendering and reconstruction of geometry-based 3D objects. The contribution of this paper is a complete media pipeline that allows for geometry-based 3D tele-immersion. Unlike previous approaches, that stream videos or video plus depth estimate, our streaming module can transmit the live-reconstructed 3D representations (triangle meshes). Based on a set of comparative experiments, this paper details the architecture and describes a novel component that can efficiently stream geometry in real-time. This component includes both a novel fast local compression algorithm and a rateless packet protection scheme geared towards the requirements imposed by real-time transmission of live-capture mesh geometry. Tests on a large dataset show an encoding and decoding speed-up of over 10 times at similar compression and quality rates, when compared to the high-end MPEG-4 SC3DMC mesh encoder. The implemented rateless code ensures complete packet loss protection of the triangle mesh object and avoids delay introduced by retransmissions. This approach is compared to a streaming mechanism over TCP and outperforms it at packet loss rates over 2% and/or latencies over 9 ms in terms of end-to-end transmission delay. As reported in this paper, the component has been successfully integrated into a larger tele-immersive environment that includes beyond state of the art 3D reconstruction and rendering modules. This resulted in a prototype that can capture, compress transmit and render triangle mesh geometry in real-time over the internet.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications – Computer conferencing, teleconferencing, and videoconferencing

General Terms

Algorithms, Measurement, Experimentation

Keywords

Graphics Streaming, LT Codes, 3D Meshes, 3D Tele-Immersion, Architecture, Compression Algorithms, Networking,

MMSys'13, February 26-March 1, 2013, Oslo, Norway

Figure 1 Our media pipeline enables the transmission, in real-time, of live-reconstructed meshes for 3D tele-immersion.

1. INTRODUCTION

3D Tele-immersion provides a common virtual space, where distributed participants can naturally interact. Advances on 3D reconstruction and rendering – and the success of the Microsoft's Kinect – enable, in real-time, the creation of highly realistic representations of the participants as triangle mesh models (see Figure 1). Efficient real-time transmission of these representations opens up new possibilities for 3D tele-immersion and mixing real and virtual environments. Unfortunately, existing video codecs and packetisation schemes do not support such representations (neither do geometry streaming mechanisms intended for downloading and interacting with remotely stored geometry-based objects). This paper takes the first step towards this direction, by reporting on our efforts in developing a complete media pipeline that is capable, in real-time, of efficiently transmitting live-captured mesh objects between remote locations. Results show that our solution outperforms existing mechanisms, when transmitting high quality 3D geometry representations.

3D tele-immersion has been studied in the past for a variety of application areas such as creative dancing, cyber-archeology, medicine, and gaming [1][2][3][4]. While high-resolution video conferencing systems may facilitate basic interaction between

distributed sites, they generally fail in providing users a natural way for eye contact (gaze), when more than two sites are involved, and they do not truly immerse users in the same virtual environment. Over the years, some attempts have been made to address these issues by using larger displays, multiple cameras, gaze correction mechanisms, and expensive fully furnished environments. [5]. Instead, a system like ours, that allows the transmission of live-captured 3D representations (such as a triangle mesh) of the participants and their body and facial expressions, will allow realistic immersion and interaction.

Capturing highly realistic representations of people in real-time was generally only possible at professional media studios using arrays of expensive stereo cameras and hardware for real-time depth estimation based on stereo correspondence. The recent commercial success of Microsoft's Kinect, which provides reasonable depth estimates, has brought inexpensive range cameras to the market (for less than $200). By deploying multiple Kinects from different angles, reasonable 360-degree geometric representations of people can be reconstructed (see Figure 1). Moreover, the wide availability of general-purpose GPU and parallel computing, allows a speed optimization of the 3D reconstruction process to real-time [6][7]. Finally, modern graphics cards and displays allow multi-view and auto-stereoscopic rendering of such 3D reconstructions.

Looking ahead in the future, when current challenges regarding calibration and synchronization of the different Kinects are solved, we believe that 3D tele-immersion will become integrated into social network experiences (similar to current Google's Hangout video conferencing). Still, in order to enable real-time transmission of such geometric representations over the Internet, efficient compression and streaming mechanisms that can operate in a realistic environment will be needed. For regular video, such mechanisms that can operate in real-time are already available: H.263+ and H.264 compression and network streaming profiles based on RTP/RTSP. However, if the captured human representation for 3D tele-immersion is geometry-based (e.g., triangle mesh), existing codecs and packetisation schemes cannot be used, as they do not support this format. There is a need for a compression and streaming scheme geared towards live-captured geometries, which is the main contribution of this paper.

Over the years, various schemes for geometry-based compression have been developed, mostly aimed at efficient rendering. Transmission schemes and middleware solutions for real-time streaming over lossy networks have been developed as well, generally dealing with remote downloading of stored objects [17 18 19]. None of these approaches handle the critical real-time requirement as imposed by live-captured triangle mesh for 3D Tele-immersion. This paper introduces the particular requirements of this new generation of 3D tele-immersion systems, it describes a working prototype of the transmission engine and its architecture, and it reports on a set of experiments, both with the integrated prototype and with offline collected data captured with five Kinects.

This paper is structured as follows. Section 3 overviews the related work regarding existing 3D tele-immersion systems, mesh geometry compression algorithms, and mesh geometry transmission solutions. Next, section 4 introduces the streaming pipeline of our 3D tele-immersion system. Section 5 presents a specific compression method we have designed for captured meshes, comparing it with some existing mesh compression schemes. Section 6 describes the proposed transmission scheme based on a rateless code, reporting results that highlight the benefits of this approach over TCP transmission. Section 7 presents our 3D Tele-immersion system, highlighting the performance of the overall system. We discuss the implications of our work in section 8. The paper concludes with section 9. The next section motivates our work and discusses our contributions.

2. MOTIVATION AND RESEARCH QUESTION

Real-time streaming of live-captured video is common in video conferencing systems, but streaming of captured triangle geometry objects has rarely been considered in the past. There are various reasons why we consider efficient real-time transmission of live-captured triangle geometry essential for the next-generation of 3D tele-immersion systems. First, modern graphics cards can take advantage of advanced rendering methods, such as multiple views for stereo and multi-stereoscopic or free-viewpoint rendering. Second, it allows for easy integration with virtual worlds, where triangle mesh representations are common. Finally, novel application areas can emerge such as natural interactions between people in real and virtual worlds. Geometry Streaming solutions can be also beneficial for applications like camera or terrain surveillance, where geometric data is live-captured and needs to be available in real-time to observers or automatic engines. In this paper we are concerned with the 3D tele-immersion application for interactive purpose.

In particular, this paper aims to answer the following research question:

What is an efficient way to transmit real-time live-captured triangle mesh geometry in real-time in the internet, as needed for 3D Tele-immersion?

Our main contribution is the design and development of a streaming module that takes into account the specific requirements for streaming captured meshes. The requirements are the following:

Support a full 3D triangle mesh representation: the engine should support streaming of the common 3D triangle mesh representation. That is, a list of points with properties (coordinate, normal, colour) and a list of faces indexing these points, resulting into a surface in the 3D space.

Low end-to-end Latency is generally considered the most important factor in 3D Tele-immersion, as the pipeline consists of bandwidth and computation savvy operations. For this paper we aim to provide an end-to-end transmission latency below 300 ms, based on video conferencing requirements.

Flexible I/O representation: the data should efficiently flow from the capturing and reconstruction blocks, via the streaming engine, to the renderers without blocking. To allow for minimum pipeline delay and possible synchronization between different streams a flexible I/O scheme is needed

Adaptability: the mechanism should be able to adapt to changing network conditions such as bandwidth, possibly reducing the quality of the captured stream. Some rate/complexity control is desired.

Robustness to packet loss as it occurs in congested networks is desired. Some quality degradation may happen, but it should be possible to reconstruct the triangle mesh at the receiver in case of packet loss.

Bandwidth: the triangle mesh stream should not consume too much bandwidth; some form of compression is desired.

No a priori information of the geometric properties of the objects can be assumed. The application should be able to stream any triangle mesh that is currently captured. Similar to video conferencing, any object that is captured should be streamed. This means that no pre-stored avatars or models can be transmitted as placeholders.

Real time Live-captured triangle meshes should be supported. Contrary to live captured video, where frames generally consist of a fixed number of points (320x240, 640x480), captured triangle mesh frames can have different numbers of points i.e. no such point (pixel) correspondence between frames can be made. This means that the triangle meshes have to be transmitted as a sequence of static meshes. To our knowledge, a mechanism to estimate this correspondence between captured points in real time does not exist.

Contribution: this paper presents a component that can efficiently stream in real-time triangle mesh geometry, that is live captured and reconstructed. The component has two main subcomponents: encoding and transmission. First, we introduce an encoding mechanism that reduces the size of the reconstructed mesh to values similar to those obtained with a state of the art TFAN MPEG encoder, but over 10 times faster. This meets our requirements on latency and bandwidth. Second, we provide a transmission scheme that uses a rateless code, meeting the requirements of robustness, latency and adaptability to changing network conditions.

3. RELATED WORK

3.1 3D Tele-immersion

A 3D tele-immersion application presents great challenges to capturing, streaming/networking and rendering technologies. The first 3D tele-immersive research dates back to the beginning of this millennium. The National Tele-Immersive initiative NTII, a consortium of American Universities and industries, demonstrated a first 3D immersive system that captured 3D point cloud representations and stereo video, and rendered using stereoscopic displays [9]. This system allowed immersive interaction between sites but it was not optimized for efficient transmission, as the focus was on rendering and reconstruction. The system used the Internet2 and TCP/IP to send the point cloud representations and the stereo video uncompressed. In this system, video streams were sent from different machines to a cluster. As these different TCP/IP streams often utilize a common link, the TCP bandwidth congestion mechanism made such streams to compete, degrading the overall quality. Ott and Patel [10] developed a coordination protocol that allowed, in the gateway, coordination between the streams to alleviate this problem. At the University of Illinois, Yang et al. looked at the case of streaming multiple live-captured 3D videos in an overlay with 4-9 sites on the Internet2, leading to a large amount video traffic [11]. By adapting the forwarding mechanisms, in the overlay network, to the view of the specific recipient user, they achieved bandwidth gains by timely dropping irrelevant streams. Huang et al. looked at a similar problem, also taking into account the synchronization/skew level between the streams. They developed a scheme, called sync-cast [12], that allowed video streams, in an overlay with multiple immersive sites, to be forwarded based on bandwidth, synchronization and latency requirements. Vasuvedan et al. proposed a 3D tele-immersion system for capturing and rendering. This system can in real-time reconstruct humans in the scene with high level of detail [7]. They used 12 clusters of 4 DragonFly cameras that reproduce 3D colour plus depth images of a human. On this depth image, triangular meshing is applied. The main advantages of this technique are that by interpolating points a more efficient representation of the depth image is possible, and that by changing the size of the triangles it is possible to adapt the level of detail. Wu et al. developed a streaming engine that exploits the aspect of this representation, [8], where the representation is referred to as colour plus depth and level of detail (CZLoD). In this study they first investigated human perception, by testing a set of samples generated with a stimulus engine. The authors found the just noticeable degradation and just acceptable degradation levels, and subsequently used these for the dynamic adaptation of the CZLoD, depending on network and user conditions. With this engine, the real-time CZLoD can be adapted to match user perception for the given available network bandwidth and user conditions. This system represents the current state of the art in 3D tele-immersive systems. Note that the CZLoD representation is different from the full 3D polygonal mesh that we deal with in this paper. The CZLoD representation is a triangulation on a depth image, while a polygonal mesh is a triangulation in a full 3D space.

3.2 Compression of Triangle Meshes

Peng [13] provides a survey of a number of compression methods up to 2005. Some common terms in mesh compression are single rate encoding, which refers to coding at one quality level; progressive encoding, which allows lower quality reconstructions if part of the data is received. Mesh coding can be connectivity driven (encodes indices first) or geometry driven (encodes vertex data first). Generally, geometry encoding can be lossy, but connectivity coding should be lossless to maintain the original topology. The single rate encoder that achieves the highest compression rate is the Touma and Gotsman encoder [22]. Compressed progressive mesh [24] is a progressive compression scheme that reconstructs a mesh by successive vertex splits and allows both more coarse and more detailed views. Streaming compression, as proposed by Isenburg et al. [14], can work on small parts of the mesh, making it more memory efficient and faster (large meshes may be too large to fit the main memory). Currently, a standardization activity of 3D graphics as geometry meshes is ongoing in the Motion Picture Experts Group (MPEG): Scalable complexity 3D Mesh Coding (SC3DMC). The standard provides 3 types of encoders with various settings in complexity to allow a tradeoff between decoding speed and compression ratio [15 16]. The most sophisticated coder, the triangle fan encoder, is reported to achieve a compression rate close to the Touma and Gotsman encoder.

Many other mesh compression methods that exploit various topological properties have been developed. They offer the compression performance needed to reduce the large data streaming rate required in 3D tele-immersion. Unfortunately these encoding mechanisms have not been explicitly designed for real-time encoding, transmission and robustness to packet losses needed for 3D Tele-immersion. A loss of connectivity data, for example, could destroy the entire topology [17].

3.3 Transmission of Triangle Mesh Geometry

Robust real-time transmission of compressed mesh geometry over lossy networks is challenging. Regib and Altunbasak [17] propose 3TP, a streaming protocol that sends parts of the compressed data over TCP and other parts over UDP. Given an allowed acceptable degradation of the mesh, 3TP selects a combination of packets to be sent over TCP and UDP. This selection is then optimized for reducing the transmission delay given the current level of packet loss. 3TP gives the user a lower

download time, but extensive offline preprocessing of the object is required to do the selection. This makes it unsuitable for the interactive streaming case, where geometry is captured, encoded and transmitted live. Another somewhat similar but more generic approach, by Li et al, is a middleware that allows streaming of various different compressed mesh representations [18]. The essence of this approach is that subsets of the representation can be sent reliably or unreliably based on the type of data (i.e. geometry/connectivity), the loss rate and the type of user environment (renderer, terminal type). This approach works for any type of mesh representations, as values are calculated offline using the general distortion measure for mesh geometry Hausdorff distance, described in [27]. This makes it more generic, but less applicable for live captured data due to the complex pre-processing step. Cheng et al. studied dependencies between data, when streaming compressed progressive mesh (CPM) based representations [19]. They found that packet loss in the initial phase, when reconstructing from a coarse mesh with vertex split operations, blocks the decoding process. When a vertex split packet is lost, many other packets cannot be decoded until the packet is retransmitted and successfully received. Cheng et al. do not solve this problem, but model the effect mathematically. This mathematical model of the dependencies in the data (using a graph representation) is used to find an optimal packet scheduling strategy (order of sending packets). Experimental evaluation shows that they are able to reduce the delays (by minimizing long term dependencies between packets).

In our work, we develop a 3D tele-immersion system that streams live captured mesh geometry. Existing compression and streaming methods are not geared towards the requirements of such application.

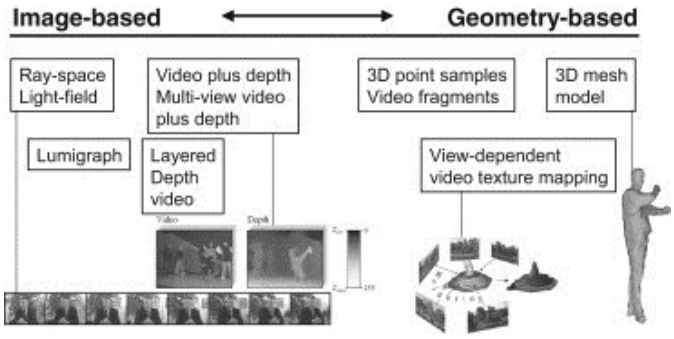

Figure 2 3D representations, from image based to geometry based, from [16]

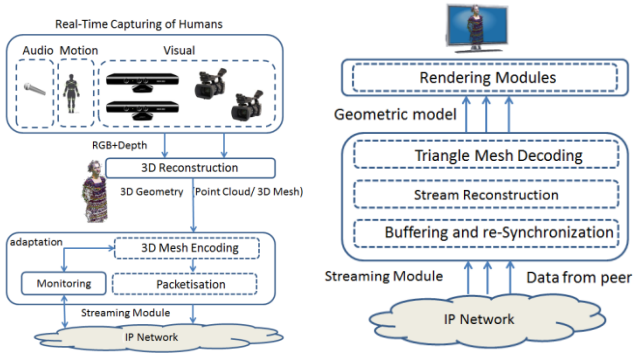

Figure 3 Our 3D tele-immersion media pipeline realized(left sender) receiver (left)

4. 3D IMMERSIVE MEDIA PIPELINE

4.1 3D Representation

3D video can be interpreted in different ways, such as the 3D stereo video in the cinema with an artificial depth perception by rendering a left and right image, or the free-view video that allows viewpoint navigation. The work in [16], which we find particularly useful, categorizes different representations on a spectrum between image-based methods and geometry-based methods (Figure 2). Examples of geometry-based representations include triangle meshes and point clouds. Image-Based methods, on the other hand, are similar to traditional video, since they use multiple separate (possibly) interpolated views. Traditionally, geometry-based methods have been restricted to games and virtual worlds.

4.2 Media Pipeline

Figure 3 shows our media pipeline at the sender site. Further details about the individual components of the system, and reports of comparison to existing mechanisms, are provided in the next two sections. First, humans are captured in real-time. Different media are captured, audio, motion and visual. In this paper we focus on the visual pipeline for a mesh geometry representation of a human. As humans interact in real-time, low delay is required. After capturing the representation has to be encoded using an efficient compression method. Subsequently, streams and packets are sent over the network. The right side of Figure 3 shows the streaming module at the receiver site. The received packets are first buffered and synchronized. Subsequently, the re-construction of the stream takes place and the object is decoded and rendered in real-time.

5. FAST COMPRESSION

Compression is one of the key methods to relieve the high bandwidth demands. As mentioned in Section 2, subsequently captured meshes can have a different number of independent points and faces. Due to the lack of such a direct relationship between the frames, we investigate static mesh codecs to encode each mesh independently, without inter-frame coding.

First, we qualitatively compare the different static mesh compression mechanism identified in the literature for their properties and applicability. Then, we present our own solution to the problem, specifically addressing the requirements for enabling a 3D tele-immersive system. We independently assess the performance of our method and compare it with MPEG SC3DMC Codecs.

5.1 Qualitative Comparison Existing Mesh Compression Methods for 3D Tele-Immersion

In this section we qualitatively compare compression methods based on the following criteria: **C** *Compression Rate,* **E** *Encoding speed,* **D** *decoding speed,* ***SC*** *scalable complexity,* ***L*** *tolerance to loss* and **P** *progressive* transmission, **T** *shape topology independence (*if the compression method is geared to a specific triangle mesh topology*)*. We rate them from 1 (bad) to 5 (very good). We use these criteria to make a selection of mesh encoders that are suitable for real-time streaming.

Touma and Gotsman propose an encoder that encodes at high rates, intended for semi-regular meshes. In the literature, this codec is considered the best in terms of compression rate. This encoder is not optimized for losses, progressive transmission, or real-time encoding. Gumhold and Strasser propose a fast (real-time) compression and decompression technique that codes at

high compression rates. We investigated this approach further, and found a specific disadvantage. The data structure that is fed to the algorithm (that makes it run in linear time) is not a typical list of points and faces, but a data structure that allows random access to a vertex indexed based on an edge. Creating such a data structure from the captured mesh representation would minimize the reported speedup advantages. We made an attempt to obtain such data structure by using algorithms available in the C++ standard library. Specifically, we created a map with pairs of points (edges) to index the other vertices in a typical captured mesh. This already resulted in delays of over 200 ms on an Intel i7 machine compiled with a 64-bit Visual Studio Compiler in release mode. The three SC3DMC encoders standardized in MPEG-4 SC3DMC (2009) have been developed for fast decoding and have many coding options that provide scalable complexity. The QBCR and SVA [16] are faster, but simpler codecs. The TFAN [15] is a more complex single rate encoder, which allows fast decompression for rendering and achieves compression rates near to the Touma and Gotsman encoder. Up to now, MPEG or IETF have not published any scheme for transmission using these codecs over the Internet. However, their speed may make them useful for 3D tele-immersion application. Khodakovsky et al. [26] developed a progressive wavelet-based coder. The advantage of this approach is that reconstructions are possible, even with partially received data. A disadvantage is that it works only for semi-regular meshes. The current implementation is available, but currently works with ASCII-based files as an input, introducing extra delays in the encoding. Compressed progressive Mesh (CPM) presented in [24] was used in many studies on transmission of geometries. It offers progressive transmission, but losses of packets can delay the decoding process as studied in [19]. Apart from that, this codec is not specifically optimized for encoding speed. The approach of Topological surgery was standardized in MPEG 3D Mesh coding (MPEG3DMC) [25], but in practice it has not been used much mainly due to its slow encoding speed and on its dependency on MPEG BIFS. Also, we compressed some meshes with an open source compression mechanism, OpenCTM[1]. This software employs entropy coding and quantization to compress meshes, instead of state of the art compression techniques geared to 3D meshes. This codec introduces a delay of over 1s when applied to the modules captured in our system and did not achieve rates comparable to other methods that are described in the scientific literature.

Table 1 Qualitative Comparison of Triangle Mesh Compression methods

Encoder Name	C	E	D	SC	L	P	T
Touma and Gotsman [23]	5	1	3	3	1	1	2
Gumhold and Strasser [24]	5	3	5	2	2	1	3
TFAN (SC3DMC) [16]	5	2	4	3	2	1	2
SVA (3DMC) [17]	3	4	4	4	3	2	2
QBCR (3DMC) [17]	2	5	5	3	3	2	4
Khodakovsky et al. [30]	5	2	4	5	4	5	1
Pajarola and Rossignac [25]	4	3	5	4	1	5	2
Taubin and Rossignac (3DMC) [26]	4	2	2	1	1	1	2

[1] www.openctm.org

The 3 MPEG-4 SC3DMC encoders with different complexities and options provide good compression rates and fast decoding. We chose to integrate them with our system for further evaluation. The software, available on mymultimediaworld.com also works with ASCII (text based) formats. The classes provided by this codecs can also be used to directly encode the binary incoming indexed face set, the common representation of polygon a Mesh. To achieve this, we wrote an extra C++ class to interface this codec, integrating it with our 3D tele-immersive system.

5.2 Fast Compression Heuristic for Captured Meshes

None of the current solutions for 3D triangle mesh compression are specifically geared to our envisioned 3D tele-immersion use case. In order to address this issue, we developed our own fast local compression method that is capable of real-time encoding and decoding and that meets the requirements introduced in Section 2. Unlike generic mesh compression methods, this algorithm takes advantage of the properties, present in meshes reconstructed from multiple depth images. The two most important observations exploited are:

1. Subsequent coordinates in the list of vertices are co-located, we exploit this with differential coding and local quantization.

2. The face indices resulting from triangulation on multiple depth images are highly structured and show repetitions that can be exploited.

Our algorithm for compressing the geometry (points) is presented as Algorithm 1, the connectivity compression method in Algorithm 2. The code is provided as pseudo-code in Tables 2, 3 and 4. Algorithm 1 takes the mesh geometry (a list of points) from the capturing component and processes it piece by piece. The algorithm allocates storage for the expected number of compressed blocks stored in coded_data. In the *while* iteration the geometric data points are processed into compressed blocks of an approximately constant size of about 1436 bytes. Algorithm 1b illustrates how the individual blocks are processed. For each local block of data, first the maximum difference in its range is computed with the method *compute max diffs per vertex value()*. Based on the local maximum difference per subvalue quantization vectors are computed. These vectors are used to quantize the rest of the differences between the specific subvalues in the block. A non-linear base quantizer was adopted with a higher resolution in the lower values, as experimentation showed that this kept more specific details in the mesh. In line 3 in algorithm 1a, based on the computed maxima, the number of vertices for each block is computed. By default each value is assigned 4 bits, but 0 bits will be assigned if certain subvalues do not change. As the blocks maintain about equal size, the number of vertices in the packet differs, based on this allocation (this is calculated by computing n_vertices for required datablock_size(P)). Subsequently, in the *for* loop in Algorithm 1a, the differences are encoded by quantizing them with the local quantizer (that was computed from the maxima). The values are stored in the block coded_data. The starting coordinates, local maxima and the number of vertices are also stored in the compressed block (data structure c_block), as they are needed in the decoding process. The index value is currently also added, in the future this could allow more flexible processing such as parallel or on the fly execution using GPU's. Currently the blocks are encoded and decoded in linear fashion.

ALGORITHM 1a	Geometry Compression
INPUT: P	Block of floating point geometric data: nV vertices with w floats of data per vertex
OUTPUT: c_data, N	N blocks of piecewise compressed data (c_data)

```
COMPRESS_GEOMETRY P, nV, w:
c_blocks  = c_block[max_number_of_blocks]
nr_vertices_processed = 0
nr_blocks = 0
pos = 0
While  (nr_points_processed < nV)
        compress_geometry_block(P.next()  c_block.next())
        nr_points_processed += c_block->nr_vertices_in_block
        nr_blocks++
End
return   c_data, nr_blocks
```

ALGORITHM 1b COMPRESS_GEOMETRY_BLOCK P, &block :	
INPUT: P, block	P: block of floating point values representing local points of the mesh , Block: an empty struct c_block representing a compressed block of data (to be filled)
OUTPUT: n_vertices, block	The number of vertices encoded in the block, the filled block of compressed data

```
Vector max_diffs = compute_max_diffs_per_vertex_value()
q_vec = compute_quantization_steps(max_diffs,  w_vec)
nr_vertices=compute_n_vertices_for_required_datablock_size(P)
coded_data[w][n_vertices]
For(j=1….w)
   prev[j] =P[j][0]
      For(i=1…..nV){
         diff =  P[j][i]- prev[j]
         cindex == qvec.find_index_closest_to( diff)
         coded_data[j][i]  = cindex
         prev[j] =    qvec[cindex] +  prev[j];
      End
End
Block->start_coords[0…w] = P[0…w][0]
Block-> max_v[0…w] = max_diffs[0…w]
Block-> nr_vertices = nr_vertices
Return n_vertices
```

```
DATASTRUCTURE C_BLOCK
Int start_index,  int nr_vertices;
 vector max_v    ,vector start_coords
byte[] coded_data
```

Table 2 Algorithm 1 Geometry Compression

Algorithm 2 handles the connectivity compression. The mesh reconstruction process introduces repetition patterns in the connectivity data. The patterns were repeating differences [+a,+a ….], alternating differences [+a,+b,+a,+b] or [+a,+a,+b,+b.] As such, patterns can occur many times, actively searching for them and encoding them as a coded sequence which we call a run obtains a large reduction in connectivity data size. Algorithm 2 starts by initializing three vectors to store the differences between points of subsequent faces. For example if face 1 <a,b,c> and face 2 <d,e,f>, then these differences will be d-e, e-b and f-c. In the beginning of Algorithm 2, we run over the entire list of faces to find these patterns (*find_run_pattern*) and store them in pattern_runs. Subsequently, in the *for* loop in Algorithm 2, either a difference between values in consecutive faces is stored in T_Coded, or, if a pattern_run was previously found, this pattern is added to T_Coded. Specifically, this is done by adding an escape value to the T_coded vector and the sequence of values representing the pattern_run (see datastructure pattern_run). The loop index *i* is then incremented with the length represented in the pattern run. In this way, the indices are either stored as 16 bit differences, or encoded in a pattern run. Specific entropy coding such as Huffman and Entropy encoding are avoided, so no extra latency is introduced. In practice, over 90% of the connectivity information is stored in runs.

Algorithm 2b represents the pattern search algorithm. In our case, we store repeating differences, and when a pattern is broken that has been repeated more than 32 times, we store the pattern as a pattern run. These patterns are then assessed in Algorithm 2.

ALGORITHM 2	Connectivity Compression
INPUT: T , nT	Array of 3 by nT representing the Triangles
OUTPUT: T_coded	3 vectors with coded geometry data

```
COMPRESS_CONNECTIVITY T, nT
T_coded[3][]
pattern_runs[]  = find_run_patterns(T,nT)
For( j=0….3)
    For(int i=0…nT)
          d_[j] =  T[j][i] – T[j][i-1];
          T_coded[j] .append(d_[j] );
          if( pattern_run.colum == j AND pattern_run.start ==i)
                T_coded [j].insert_pattern_run(pattern_run)
                pattern_run = pattern_run.next()
                i.increment(pattern_run.length)
          End
     End
End
return   c_data, nr_blocks
```

ALGORITHM 2a FIND_RUN_PATTERNS	
INPUT: T[w][nT] , nT	Array of 3 by nT representing the Triangles
OUTPUT: run_d[3]	3 vectors with pattern run data structures

```
run_d[3][]
diffs[4]
run_counter[nr_modes]
first_triangle[3] = T[0…2][0]
For(i=2; ……nT )
        diffs = compute_local_diffs
         if(pattern = find_pattern())
            run_counter.increment()
         if(pattern broken and run_counter > thresh)
           patterns.add(pattern);
End
```

```
DATASTRUCTURE PATTERN_RUN
Int mode, length, diff1,diff2,start,value;
```

Table 3 Algorithm 2 Connectivity Compression

The decompression algorithm is provided as pseudo-code in Table 4. The blocks of compressed geometry data, represented as *c_block* datastructures are all subsequently processed by algorithm 3a. Algorithm 3a re-computes the local quantization vector of the differences based on the *max_v field*. Then, based on start_coords the differential decoding of the geometric data is performed. The second decompression step involves the decoding

of the connectivity data. The first values of T_Coded represent the first face, then based on this face differential reconstruction of the different face columns is performed. When an escape value is found in T_coded for a run, the run is decoded into respective column of the faces. By processing all the values in T_Coded, the complete connectivity is reconstructed.

ALGORITHM 3	**De-Compression**
INPUT: c_data[N] T_coded	N coded data c_blocks from algorithm 1 and the vectors T_coded obtained from algorithm 2
OUTPUT: P[w][nV], T[3][], nT	The geometry data: nV vertices of w floats each, the number of faces nT

```
DECOMPRESS_MESH c_data, T
P[][]
for(block in c_data)
    P.append(decode_c_data_block(block))
For(j=0…3)
   T[j][0] = T_coded[j][0]
   for(i=1…… nT, k=1…..nT){
      if(T_coded[j][k] == run_start )
            T[j][i…i+runlength] = decode_run(T_coded[j][i])
            i.increment(runlength);
            j.incremenent( 6);
       Else{
           T[j][i] =  T[j][i-1] + T_coded[j][i];
      End
End
```

ALGORITHM 3a Decode c_data_block	
INPUT: c_block	A coded block of geometric data (a struct of c_data)
OUTPUT: P[w][]	Block of points from the decoded mesh of w floats per vertex

```
Prev[0…w] = c_block-> start_coords[0….w]
Compute_local_quantization_bounds( c_block->max_v)
for(i=0…..w)
  prev = c_block->start_coords[i]
  for(j=0……c_block->n_vert)
     P[i][j] = prev + q_diff( c_block->coded_data[i][j])
     Prev = P[i][j]
  End
End
```

Table 4 Algorithm 3 Decompression Algorithm

5.3 Experimental Results

In this section we evaluate the performance of our method with live captured data. Figures 4 to 11 show the results in terms of compression size, coding latency and distortion.

Our scheme was implemented in C++ using Visual studio and compiled using a 64 bit compiler. The tests ran on an Asus laptop (PRO64J) with a first generation (1.6 Ghz) mobile Intel i7 processor with 4GB of ram and Windows 7 home edition. The MPEG SC3DMC Codecs were compiled from source code with the same compiler and ran in the same environment. The tests were run offline with previously captured data stored in files. All files are first completely loaded into memory before the compression routine is started. The running times are recorded with os wall clock times in boost C++ that provide a wall clock time in Windows 7 with a resolution around 366 ns.

The first set of experiments (Figures 4-7) show the performance when the capturing device is a single Kinect and the capturing mechanism is tuned to capture objects within a 130 cm range. This represents a situation where a user is behind a pc with a Kinect on it. We captured high-quality representations for this dataset with on average 72,855 vertices per frame and 143,302 faces. Each vertex points contains 9 floating point values, 3 for coordinates, normal and colors each. The raw frames are therefore about 4.3 MB each. As shown in Figure 4, the size is reduced by more than a factor 10, close to the performance of triangle mesh Encoder TFAN (tuned with 8 bit quantization and differential encoding). Figure 5 shows a qualitative comparison between the reconstructed frames from the different coding mechanisms. This qualitative comparison is based on the Haussdorf distance (rms) and measured with a tool developed in [31]. The values measured represent the root mean square distance between the original and the reconstructed surface. The models decoded with our scheme have slightly less distortion and in theory are slightly better reconstructions. Note that the distortion differences (Haussdorf distance rms) between the models of 0.0004 are not significant (the reconstructions are of comparable quality). We chose the quantization values such that they allow a fair comparison between different algorithms (operating at similar quality). As we have lower distortion, it is fair to compare speed and size (assuming the quality is at least as good from human perception). The main gain of our heuristic is in the speedup. We can encode the high quality representation in about 70 ms, and decode it consistently below 10 ms.

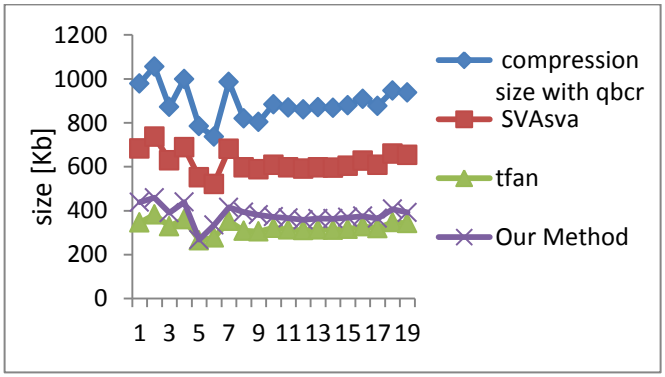

Figure 4 Compression size of live captured data frames with different methods (130 cm 1 Kinect) (Kb)

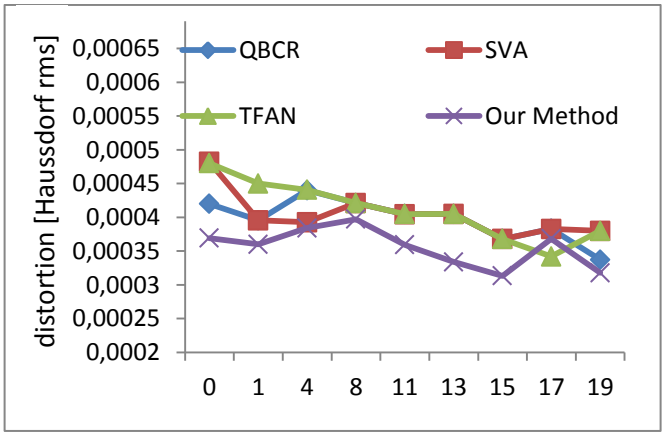

Figure 5 Distortion quality of decompressed representation - Hausdorff Distance (rms) - to original frame (130 cm 1 Kinect data)

This result implies a speedup of over 100% compared to the state of the art MPEG 3DGraphics encoder at only a slightly lower compression gain. Most gain is achieved in compressing the connectivity data, of which in most cases over 90% is encoded in runs. Figures 9-12 compare the different possible setups and

encoding solutions. In this case we compare setups with 5 Kinects and 1 Kinect at both high and low quality at a bit longer distance (300 cm), representing a more console like or living room like experience. The high quality 5 Kinect representation is the most challenging, as the frames consist of about 253,000 vertices and 487,500 faces on average. Our heuristic is able to process such a frame into a 1 MB block in on average 160ms. Further parallelization would allow transportation of such highly realistic captured realistic representations in real-time. Our heuristic heavily outperforms other methods on the speed requirement that is critical for our application.

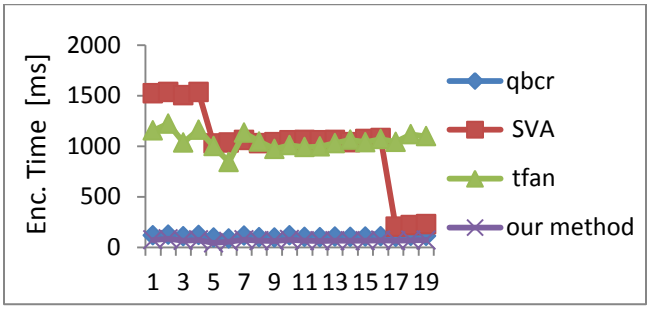

Figure 6 Encoding time with different methods [ms] (130 cm one kinect)

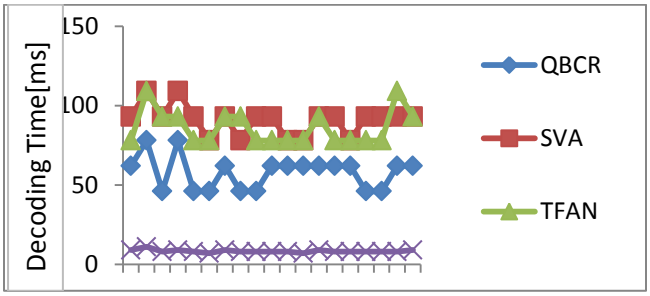

Figure 7 Decoding time with different methods [ms] (130 cm one Kinect)

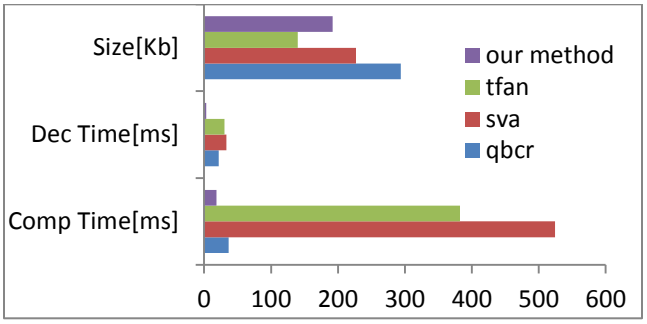

Figure 8 Low Res (~253K vertices) Data with 5 Kinects (300 cm)

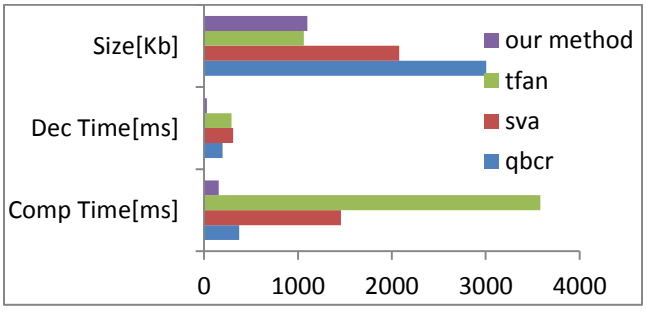

Figure 9 High Res (~72K vertices) Data with 5 Kinects (300 cm) (size in kb time in ms)

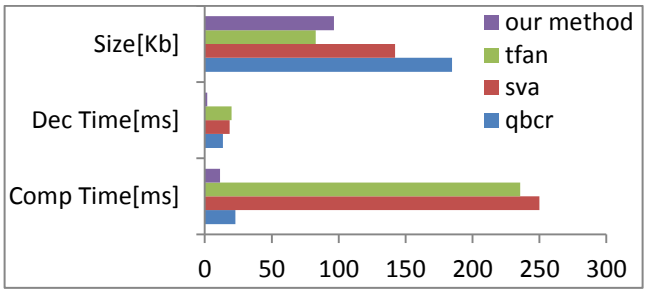

Figure 10 Low Res Data (17K vertices) with One Kinect (300cm)

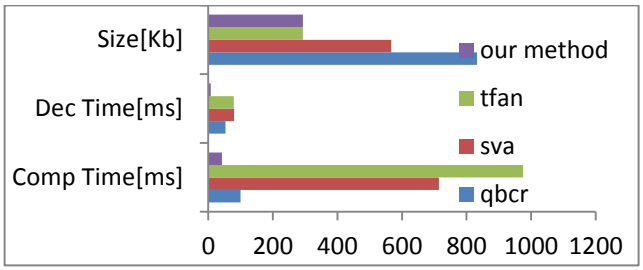

Figure 11 High Res (~70K vertices)Data With 1 Kinect (300 cm)

5.4 Discussion

The component developed in this section meets the requirements for 3D triangle mesh based tele-immersion. First, it handles in real-time full geometric input frames of up to 100,000 vertices. It avoids (except for once in the connectivity encoding) global searches/re-orderings. Moreover, the way the data is handled in small consecutive independent blocks allows parallelization and is computation/memory efficient. These small blocks also enable more flexible I/O. For example, in future scenarios meshes might be only partially sent/compressed or reconstructed. Also, in terms of bandwidth it performs similarly to the TFAN codec on the live captured meshes reconstructed in our capturing system.

The development of this 3D mesh compression mechanism, which works well with capturing systems, is a key step towards enabling the 3D tele-immersion system based on geometry. Local operation, real-time encoding and decoding are desired properties of such mechanisms.

6. REAL-TIME TRANSMISSION

Generally, interactive communication over lossy networks has been tackled with the possibility of omitting information at the receiver. Modern video codecs implement the possibility of decoding at reduced resolution or frame rate, should not all the information arrive at destination within a target end-to-end delay. Units that can be dropped are generally small and have poor impact on the continuity of the service. Triangle mesh compression has not been designed to be resilient to information losses. Thus the loss of a packet can waste a lot of resources since:

1. The packet needs to be retransmitted to make sure the mesh can be decoded, yielding to uncontrollable delays (e.g., TCP)
2. If the frame is skipped, bandwidth is wasted for information that is not decoded.

In our tele-immersive system, we implemented a rateless code to achieve minimal end-to-end delay and protection against packet

losses. In this section we introduce the concept of rateless coding and we compare it to resilient transmission via TCP, based on a number of real experiments, and show its favorable properties for geometry transmission and 3D tele-immersion.

6.1 Rateless Coding

Random Linear Coding aims to achieve packet loss protection with near optimal rate and quick adaptation to the network conditions. The idea of rateless and fountain codes is that any amount of packets can be generated at the sender. The first practical Random linear codes were first proposed in [20]. Currently, codes like Raptor and RaptorQ have been proposed as standards by IETF [29][30]. The benefits include linear encoding and decoding time of the data (compared to quadratic time in Reed Solomon codes). The codes are called rateless, as the amount of data generated is not fixed, in case of increased packet loss in the network, the data generated can be increased for extra protection. This constitutes one of the main advantages compared to traditional fixed rate FEC codes such as Reed Solomon codes. Additionally rateless codes also reduce the end-to-end delay, because they do not need retransmission of information. The receiver then only has to receive a set of packets to make sure the reconstruction of the frame is possible. A symbol based version of rateless codes, more similar to our proposed technique, has been also adopted in the field of network coding [31, 32] to allow receivers to decode from packets recursively encoded by different nodes.

6.2 Implementation

The data stream is divided in units that are encoded together (similar to NAL units in the H.264/AVC standard), e.g., frames containing the triangular mesh at a certain instant. We further divide these segments in generations and datablocks (See Figure 12). We adopt packet-based random linear coding on a Galois Field (GF). Blocks belonging to a generation are always meant to be coded with blocks from the same generation.

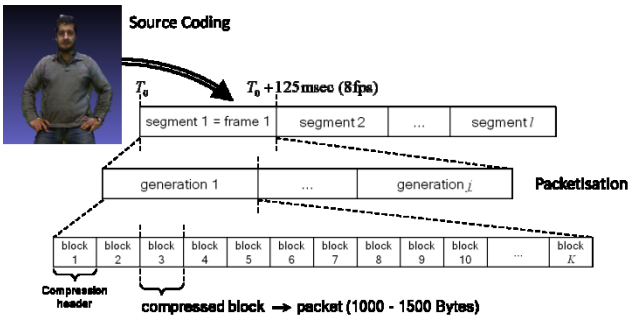

Figure 12: Arrangement of blocks for rateless coding

Each block is a sequence of codewords, each codeword made of m bits each (typically 8 bits, or a multiple of 8 bits), so that encoding and decoding operations are performed in an algebra over a Galois Field (GF) of size 2^m. A new packet is generated by linearly combining the K source blocks of the current generation with random coefficients $c_1, c_2, \dots, c_K$. A codeword $b_{new,j}$ from a new coded block $\boldsymbol{b}_{new}^{(g)} = b_{new,1}, \dots, b_{new,\ Nw}$, of generation g can be expressed as:

$$b_{new,j} = \sum_{i=1}^{K} c_i \, b_{i,j}^{(g)}, \qquad j = 1, 2, \dots, N_w \tag{1}$$

where $b_{i,j}^{(g)}$ is the j-th codeword of the i-th block in generation g.

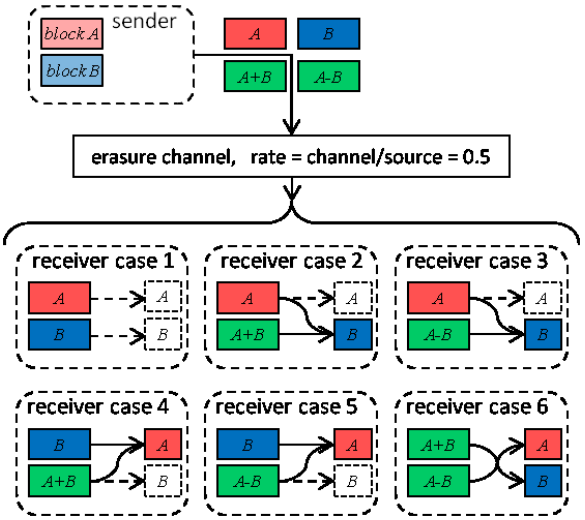

Figure 13: Example of rateless coding and decoding from linearly independent subsets of packets

The coefficients of the linear combination are embedded in the packet header, to make sure the receiver knows which specific linear combination has been received. Decoding operations are also performed between blocks labelled with the same generation. As soon as enough packets are received for a generation, the coefficients are used to build a $K \times K$ linear system that allows decoding and recovering the data. Figure 13 shows a simple example of how packets randomly-coded from 2 two source blocks are decoded from any linearly independent subset of blocks. In order to reduce the decoding computational load, we construct a composite matrix of data, and coefficients of the incoming packets and perform Gaussian elimination each time a new packet is received. This spreads the computational cost over time and drastically reduces the decoding complexity. Such complexity can be still reduced by reducing the dimension of the coding space. These factors need to be properly considered:

1. Loss protection (Larger coding diversity).
2. Decoding complexity (Smaller linear system).

Properly balancing the coding space between big linear systems (more coding diversity, more decoding complexity) and small systems (less coding diversity, less complexity) allows achieving optimal delay performance and the required resilience against packet losses.

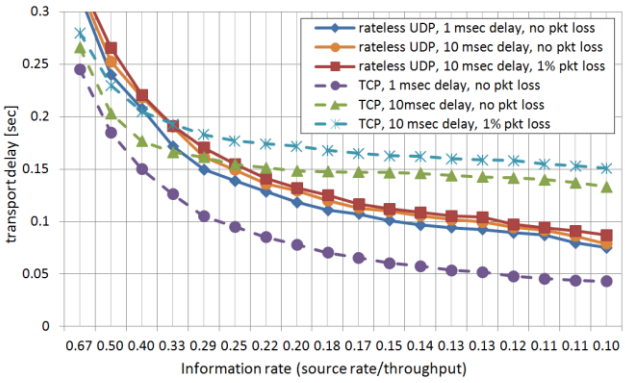

Figure 14: Transmission delay of TCP and rateless coding varying depending on the available channel rate.

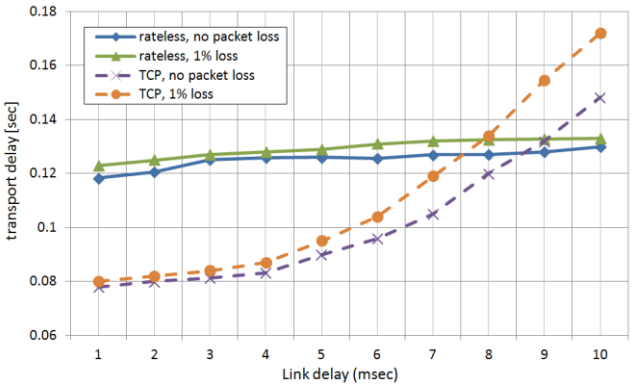

Figure 15: Transmission delay of TCP and rateless coding in seconds, due to network delays.

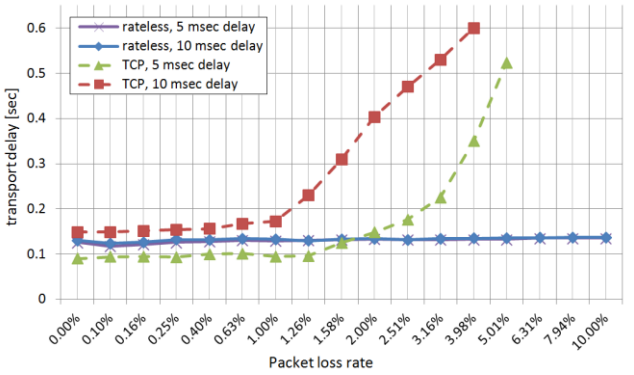

Figure 16: Transmission delay of TCP and rateless coding in seconds, due to packet losses.

6.3 Experimental Results

In order to assess the delay performance of the rateless coding system we run some experiment on an experimental setup composed by two Intel Core i5 machines (3.10 GHz): one in charge of capturing, encoding and transmission and one receiving from the network and decoding the source data; a network emulator that reproduces a large variety of network conditions in terms of delay, bandwidth and packet loss rate is run in the receiving machine. Our rateless transmission system makes use of UDP packets and is compared with a standard self-managed and reliable TCP connection. The factor influencing the efficiency of our rateless transmission is the ratio between throughput and source rate, given a certain packet loss rate. This should always be able to sustain the source information rate. We show the delay performance relative to a set of experiments with limited bandwidth, variable delay, and packet loss rate. Delays and packet losses in the network affect only linearly the delay performance of the rateless decoding, as opposed to TCP that needs to engage mechanisms of recovery every time a packet is lost. The mechanisms of recovery are further affected by the link delay. We analyse now the delay introduced by the transmission and channel coding and decoding and the way this is affected by the network conditions. Figure 14 shows the introduced delay and the influence of the available bandwidth, with different packet loss and delay conditions. Rateless coding works best when extra bandwidth is available, in order to cope with some additional overhead. Figure 15 shows the delay performance depending on the latency of the network, whereas Figure 16 shows again the transmission delay and its sensitivity to packet losses. Although in some ideal conditions TCP reaches optimal transmission performance, it suffers large transmission delay in the case of network delays and packet losses, making it unsuitable in realistic networking conditions. In the rateless system, the influence of network impairments is linear and controllable, often hardly mutable even in realistic networking conditions. The rateless code achieves good delay performance.

7. 3D TELE-IMMERSIVE SYSTEM

7.1 3D Triangle Capturing and Rendering

The capturing component was provided by the Center for Research and Technology Hellas. It creates reconstructions in real-time (8-10fps) of humans using range images (RGB plus depth) captured with multiple Kinects. An early version of the system is described in detail in [7]. The system is based on merging tessellated depth images using either zippering or volumetric method, which are the most common methods to reconstruct a triangle mesh from multiple depth images. The render component renders meshes in real time with shading based on global illumination effect using the normal data in the vertices. This component was implemented with OpenGL and QT and provided by Institut Telecom, Paris. These components have been linked together to construct the immersive prototype.

7.2 Media Pipeline Performance

We tested the computational performance of the media pipeline in two different ways. Table 5 shows the results when running the sender and receiver on one machine (a 1.6 Ghz intel i7 laptop, 4GB Ram). We captured meshes with one depth camera, reconstruct them and send them over the local interface back and render them. In the process, we recorded the time taken by capturing, encoding, decoding and the rate at which frames were sent and received. The update frequency (refresh rate) of the renderer to the screen was also tested. This illustrates achievable frame rate in the pipeline.

Sub-part	Average [ms]	Std [ms]
Capturing	94 ms	5,3 ms
Encoding	52 ms	3,5 ms
Decoding	11 ms	1,1 ms
Rendering	46 ms	4 ms
Send-Rate	5-8 fps	
Recv-Rate	5-8 fps	

Table 5 Performance of 3D Tele-Immersive Pipeline on local interface (Captured Meshes with around 50,000 vertices)

The graphs in Figure 17 and Figure 18 show the global end-to-end delay of the system from capturing to rendering, using both the traditional transmission over TCP and our rateless coding system. In this case two Intel Core i5 machines (3.10 GHz) PC's are connected (sender and receiver) and the network impairment simulator is used to generate the networking conditions between the machines (connected in the LAN). The delay over the link was 10ms in all experiments, while data could be medium or high resolution. The medium resolution frames contained around 16-18k vertices (90-100 kbytes per compressed frame). The high resolution was around 50k vertices, (280-300 kbytes compressed per frame). We performed different tests, with 0% and 1% packet loss. Figure 17 and 18 show that the system still enables real-time communication in the case of packet loss/delay. The system also allows reconstruction and reliable transmission without retransmission. By employing rateless coding over UDP and fast mesh compression we avoid exceeding the target end-to-end delay

for interactive applications of about 300 ms including live capturing and 3D rendering.

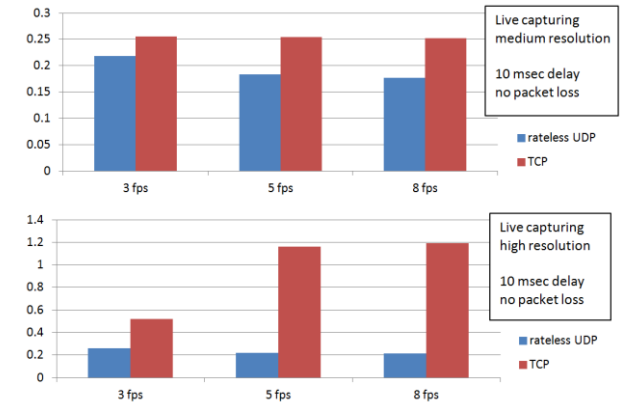

Figure 17 End-to-end delay with no packet loss

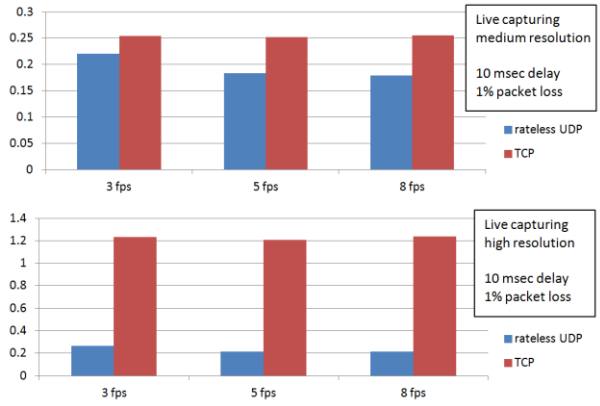

Figure 18 End-to-end delay with 1% packet loss

8 Discussion

This paper presents a prototype implementation that enables conferencing between remote participants, focusing on the compression and the transmission components. The novelty is that our 3D tele-immersion system is based on triangle mesh representations, unlike previous solutions. The triangle mesh representation has traditionally been supported by computer graphics in virtual worlds and games. Real-Time streaming of captured mesh data, therefore, will enable novel applications that can integrate real and virtual worlds. The prototype addressed some of the significant challenges that triangle mesh representation poses to the media streaming pipeline in terms of latency, data-volume and robustness to losses.

The contributions can be summarized as follows:

Encoding/Decoding: triangle mesh codecs have not been designed with the interactive scenario in mind. The encoding time is simply often too long. We developed a local method that takes advantage of specific properties resulting in a speed increase of ten times when compared to the TFAN encoder from MPEG, at comparable quality/rate. The method works well with the meshes produced by our capturing system. It takes advantage of the coherence property of the captured mesh. This property was also found to be present in reconstructed and captured meshes in [14]. Also, the relatively simple operations of this method and block separation allow for fast parallel or hardware implementations.

Streaming: systems that efficiently transmit geometry in real-time, generally dealt with stored objects instead of captured reconstructions. Such middleware solutions, and application-layer protocols [17][18][19], facilitate efficient real-time downloading of a mesh. This is achieved by an offline optimization step, which cannot be performed in geometry-based 3D tele-immersion systems. We do not develop a specific middleware or protocol but introduce an implementation of a rateless code (inter-packet fec) that protects each mesh frame against packet losses. Rateless code allows any given number of extra packets to be generated, depending on the amount of protection that is needed. Rateless codes, like Raptor and its successor RaptorQ, have both been proposed as standard by IETF [33][34] in 2007 and 2012, but have seldomly been tried for real-time streaming of geometry-based data. Our experiments show the favorable properties of the rateless code, such as low end-to-end delay compared to TCP in case of packet loss of over 2%. Moreover, the distributed implementation of the packet decoder introduced a modest delay of 50 ms. In addition, the streaming method is generic, video audio and other data can also be efficiently transported in real-time based on this code.

3D Tele-immersion: this paper presents a prototype of a triangle mesh based 3D tele-immersion system. This prototype is integrated with state of the art capturing and rendering components. None of the previously presented 3DTI systems can capture and stream full 3D reconstructions in real-time. On top of that, state of the art rendering techniques can be applied such as global illumination. The focus in this paper was on the visual media pipeline, but further integration with spatial audio techniques such as binaural hearing are planned in the near future. A next integration step will include merging the received 3D mesh live into a virtual world adding spatial audio.

CONCLUSION

This paper presented a 3D tele-immersion pipeline based on captured mesh geometry. The demanding requirements of the application have been addressed in different parts of the media pipeline. By doing this, we extended the state of the art in handling mesh geometry to meet the live captured case. First, we developed a local fast compression method that, contrary to previous codecs, takes advantage of the specific properties of real-time captured/reconstructed meshes. Second, we implemented a streaming mechanism based on a rateless code that allows robust transmission as is needed for protecting the loss-sensitive mesh. This system was integrated with state of the art rendering and capturing components. In the future, further integration with spatial audio and virtual words is planned.

8. ACKNOWLEDGMENTS

Thanks to Demetrios Alexiadis and Petros Daras at the CERTH for providing datasets and the reconstruction software,. Tamy Boubekeur from Telecom Paris Tech for providing the lightweight MESH Rendering Engine. 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). We thank the people at Institut Telecom and Multimedia world for making the codecs available.

9. REFERENCES

[1] Yang Z., Yu B., Diankov R., Wu W., and Bajcsy W. Collaborative Dancing in Tele-Immersive Environment(2006). ACM International Conference on Multimedia (MM) pp. 723-726

[2] Forte M., Kurillo G., Cyberarchaeology - Experimenting with Teleimmersive Archaeology (2010) 16th International Conference on Virtual Systems and Multimedia (VSMM 2010), pp 155-162.

[3] Kurillo G. , Bajcsy R., Kreylos O., Rodriguez R., Tele-immersive environment for remote medical collaboration (2009). In Proceedings of Medicine Meets Virtual Reality (MMVR17), pp. 148-150.

[4] Nahrstedt K. , Bajcsy R., Wymore L., Kurillo G., Mezur K., Sheppard R., Yang Z. , and Wu. W. (2007), Symbiosis of Tele-Immersive Environments with Creative Choreography. ACM Workshop on Supporting Creative Acts Beyond Dissemination 2007 (in conjunction with CCC'07).

[5] Kauff P., Schreer O. An immersive 3D video-conferencing system using shared virtual team user environments (2002). In Proceedings of the 4th international conference on Collaborative virtual environments *(CVE '02)*. ACM, pp 105-112

[6] Alexiadis D., Kordelas D.S., G. Apostolakis, K.C. ; Agapito, J.D. ; Vegas, J.M. ; Izquierdo, E. ; Daras, Reconstruction for 3D immersive virtual environments (2012). 3th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS) pp.1-4

[7] Vasudevan R., Kurillo K., Lobaton E., Bernardin T.,Kreylos O., Bajcsy R., Nahrstedt, K. High Quality Visualization for Geographically Distributed 3-D Tele-immersive Applications(2011). *IEEE Transactions on Multimedia*, Vol. 13, NO 3, June 2011 pp. 573-584

[8] Wu W., Arefin A., Kurillo G., Argawal P., Nahrstedt K., Bajcsy R., Color-plus-Depth Level-of-Detail in 3D Tele-Immersive Video: A psychophysical Approach (2011). In Proceedings of the 19th ACM international conference on Multimedia *(*MM '11). ACM, New York, NY, USA, pp. 13-22.

[9] Towles H. , Chen W. , Yang R., Kum S. , Fuchs H. Kelshikar., Mulligan J. , Holden L., Zeleznik B., Sadagic A. , Forman J.L., 3D Tele-collaboration over internet2. (2002). *International workshop on tele presence 2002, Juan Les Pins*

[10] Ott, D.E. and Mayer-Patel, K., 2004. Coordinated multi-streaming for 3D tele-immersion (2004). 12th Annual ACM International Conference on Multimedia, pp. 596—603.

[11] Yang Z., Wu W., Nahrstedt K., Kurillo G., and Bajcsy R.. 2010. Enabling multi-party 3D tele-immersive environments with *ViewCast*. *ACM Trans. Multimedia Comput. Commun. Appl.* 6, 2, Article 7 (March 2010), 30 pages.

[12] Huang, Z., Wu, W., Nahrstedt, K., Rivas, R. and Arefin, A., SyncCast: Synchronized dissemination in multi-site interactive 3D tele-immersion (2011). 2nd Annual ACM Conference on Multimedia Systems (MMSys '11)., pp. 69—80.

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

[14] Isenburg M., Lindstrom P., 2005. Streaming meshes (2005). IEEE Visualization (VIS 05) pp 231-238

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

[16] Jang E.s., Lee S., Koo B., Kim D., Son K.Fast 3D Mesh Compression using Shared Vertex Analysis (2010) ETRI Journal, Volume 32, Number 1, February 2010

[17] AlRegib G. and Altunbasak Y., 3TP: An application-layer protocol for streaming 3-D graphics,(2005) *IEEE Trans. on Multimedia*, Vol. 7, No. 6, pp. 1149-1156

[18] Li H., Li M., and Prabhakaran B.. Middleware for streaming 3D progressive meshes over lossy networks (2006.). *ACM Trans. Multimedia Comput. Commun. Appl.* 2, 4 (November 2006), 282-317.

[19] Wei Cheng, Wei Tsang Ooi, Sebastien Mondet, Romulus Grigoras, and Géraldine Morin. 2011. Modeling progressive mesh streaming: Does data dependency matter?.(2011) *ACM Trans. Multimedia Comput. Commun. Appl.* 7, 2, Article 10 (March 2011), 24 pages.

[20] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data(1998). *SIGCOMM Comput. Commun. Rev.* 28, 4 pp. 56-67.

[21] Smolic, A., 3D Video and Free-viewpoint Video-From Capture to Display (2011). *Elsevier Pattern Recognition* Volume 4 Issue 9 pp. 1958-1968

[22]] Touma C., Gotsman T., Triangle mesh compression, (1998) Proceedings of Graphics Interface, 1998,pp. 26–34.

[23] Gumhold S., Straßer W., Real time compression of triangle mesh connectivity(1998), in: ACM SIGGRAPH'98 pp. 133–140.

[24] Pajarola R., Rossignac J., Compressed progressive meshes, *IEEE Trans. Vis. Comput. Graph.* 6 (1) (2000) pp. 79–93.

[25] Taubin G., Rossignac J., Geometric compression through topological surgery, *ACM Trans. Graph.*17 (2) (1998) pp.84–115.

[26] Khodakovsky A., Schroeder P., Sweldens W. Progressive geometry compression(2000). In Proceedings of the 27th annual conference on Computer graphics and interactive techniques SIGGRAPH '00. pp. 271-278.

[27] 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 ICME pp. 705-708

[28] M. Luby (2002). "LT Codes". *Proceedings of the IEEE Symposium on the Foundations of Computer Science*: pp. 271–280.

[29] M. Luby, A. Shokrollahi M. Watson, T. Stockhammer, RFC 5053 Raptor Forward Error Correction Scheme for Object Delivery (2007)

[30] M.Luby A. Shokrallahi, M. Watson, T. Stockhammer, L.Minder RFC 6330 RaptorQ Raptor Forward Error Correction Scheme for Object Delivery**(**2011)

[31] Li, S.-Y.R.; Yeung, R.W.; Ning Cai; , "Linear network coding,"(2003) *Information Theory, IEEE Transactions on* , vol.49, no.2, pp.371-381, Feb. 2003

[32] Chou P.A., Wu Y., and Jain K.. Practical network coding. *In Allerton Conference in Communication, Control and Computing, Monticello*, IL, Oct. 2003.