Multimed Tools Appl (2011) 51:779–799
DOI 10.1007/s11042-010-0634-8

Receiver-based loss tolerance method for 3D progressive streaming

Ziying Tang · Xiaohu Guo · Balakrishnan Prabhakaran

Published online: 24 October 2010

Abstract While progressive compression techniques were proposed long time ago, fast and efficient streaming of detailed 3D models over lossy networks still remains a challenge. A primary reason is that packet loss occurring in unreliable networks is highly unpredictable, leading to connectivity inconsistency and distortions of decompressed meshes. Although prior researches have proposed various methods to handle errors caused by transmission loss, they are always accompanied by additional costs such as redundant transmission data, bandwidth overloads, and result distortions. In this paper, we address this problem from a receiver's point of view and propose a novel receiver-based loss tolerance scheme which is capable of recovering the lost data when streaming 3D progressive meshes over lossy networks. Specifically, we use some constraints during the model compression procedure on the server side, and suggest a prediction method to handle loss of structural and geometric data on the client/receiver side. Our algorithm works without any data retransmission or introducing any unnecessary protection bits. We stream mesh refinement data on reliable and unreliable networks separately so as to reduce the transmission delay as well as to obtain a satisfactory decompression result. The experimental results indicate that the decompression procedure can be accomplished quickly, suggesting that it is an efficient and practical solution. It is also shown that the proposed prediction technique achieves a very good approximation of the original mesh with low distortions, and in the mean time, error propagations are also well controlled.

Keywords Progressive compression · 3D streaming · Loss tolerance

Z. Tang (✉) · X. Guo · B. Prabhakaran
Computer Science Department, University of Texas at Dallas, Richardson, TX, USA
e-mail: zxt061000@utdallas.edu

X. Guo
e-mail: xguo@utdallas.edu

B. Prabhakaran
e-mail: praba@utdallas.edu

1 Introduction

With the development of the Internet, networks related applications have grown phenomenally in the last few decades. Among different media, 3D models have received increasing attentions due to their abilities of representing objects more realistic and comprehensive, and in the past several years, they have been extensively used in network multimedia applications such as online games, distributed simulations and Internet shopping. In general, the amount of data representing a high quality 3D mesh can be very huge. It may have thousands to millions of vertices to represent its complex geometry, and connectivity relationships among vertices to denote the complicated topology. Therefore, it has long been considered as a challenge to quickly transmit detailed 3D meshes to end users with acceptable quality.

Progressive compression techniques [1, 10, 19, 23] address this problem by sending to users a crude mesh followed by a sequence of refinements so that they can provide meshes in different level of details. By doing so, users are able to visualize a rough model quickly, and then gradually, the model increases its details. Most of those progressive compression algorithms are designed to transmit using lossless network protocols such as TCP. Using reliable channels is stable and safe, but it is not real-time streaming friendly and may result in an undesired delay due to the data retransmission mechanism. In order to raise the streaming speed, some algorithms [3, 4, 6] use unreliable but fast protocols, i.e., UDP, other techniques such as [15–18, 24] utilize hybrid channels to take advantages of both TCP and UDP. Although these methods are more streaming effective, but packet loss due to the usage of unreliable transmission protocol is a problem they cannot avoid. Packet loss leads to incomplete decoding data on the receiver side, which can cause crashing of the decoding procedure or large distortions on the decompressed results. Therefore, it is very necessary to have error handling scheme to deal with packet loss. Unfortunately, existing methods to handle packet loss either introduce extra protection information, which introduces extra transmission loads, wastes network bandwidth and substantially degrades the streaming performance, or simply retransmit the lost packet which causes the unavoidable time delay. Thus, there is no very efficient solution which maintains high transmission speed and low communication loads, and at the same time obtains good decompression results.

In this paper, we address the problem of packet loss during 3D progressive streaming from a different point of view, i.e., clients' side. We propose a receiver-based loss-tolerance streaming strategy, and our focus is in handling packet loss without retransmission or any recovery and protection bits. Specifically, we introduce a prediction method to obtain an approximated decompression result when a packet loss occurs. Since a vertex split transformation takes place in the split-vertex's one-ring neighborhood, which is generally a small region, we can project this region in 2D space and apply a hole filling technique to insert new vertices and then perform triangulation. In addition, we adjust new vertices' geometric coordinates using the idea of the partition of unity method [18]. The fact that vertex split transformation is a local operation and involves a small number of neighbor vertices on the mesh makes our prediction very efficient, i.e., we only need $O(n)$ time to predict n lost vertices. Furthermore, to control prediction error in a limited area, we define a constraint during model compression procedure in addition to the use of reliability bit [13, 16]. Our method enables fast and successful streaming of 3D models progressively over lossy networks. Besides, the prediction process can be completed very quickly and does not delay the decompression procedure.

Similar to [3, 13], we use hybrid channels to deliver different parts of encoded information separately. In order to satisfy various quality requirements and network capacities, refinements are further constructed into smaller packets through a subdivision procedure. On the user side, we utilize a reconstructor to synchronize hybrid channels, reassemble packets and detect packet loss. During decompression, depending on whether complete refinement data for decoding is available, a regular decompression or the proposed loss tolerance scheme is applied to gradually obtain a detailed 3D mesh.

Our approach has the following novelties. This receiver-based loss tolerance scheme works independently of channel behaviors, and we are able to predict both the structural data and the geometry information. We can successfully decode meshes without retransmission or any protection bits, thus there is no additional transmission load and no extra transmission delay. The prediction achieves a good decompression quality without incurring high computational cost.

The rest of this paper is organized as follows. After reviewing the related works in Section 2, Section 3 gives a short overview of progressive compression technique and related concepts. Details of the loss tolerance method are presented in Section 4, and Section 5 explains the performance evaluations. The last part, Section 6, concludes this paper with some future works.

2 Related works

Our work is closely related to two research directions: the progressive compression techniques and related works in transmission of progressive meshes over networks.

2.1 Progressive compression techniques

Progressive Mesh (PM) [10] is the first progressive compression technique. It defines the elementary mesh transformations: vertex split and edge collapse, as shown in Fig. 1. In this method, at every compression level, only one vertex is removed through an edge collapse transformation. Similarly, during decompression, one new vertex is inserted into the current mesh, using a reverse transformation, vertex split. Though PM is able to provide meshes in different level of details (LODs), it fails to gain a good compression ratio in one level.

Both Progressive Forest Split (PFS) [23] and Compressed Progressive Meshes (CPM) [19] follow a similar compression strategy and group edge collapse transformations into batches to achieve a higher compression ratio. In CPM, the connectivity information of a removed vertex is stored in two cut edges while its geometric information is represented as an error correction vector storing the difference between the original position and the predicted displacement. This idea has been extended in Modified Compressed Progressive Meshes (MCPM) [24]. In [24] the geometric data of each refinement batch is divided into some sub layers, so that optimal transmission decision could be made for each sub layer. Alliez and Desbrun introduce a valence-driven decimating approach [1] for lossless transmission of triangle meshes. By checking the degree, a vertex can be removed with a valence of more than three. In [2], the encoded geometric data contains the difference between the predicted and the real vertex positions, which is similar to CPM, but the connectivity information is the only criteria to select removed vertices, so [2] is less accurate than CPM. Devillers and Gandoin propose a geometry-driven compression algorithm [8], where they use the *k–d* tree to represent the geometric information of vertices and use neighboring vertices to derive the structural data.

From a different point of view, some works such as [9, 10, 21] support selective refinements. They consider the fact that users cannot see the whole model’s all parts at one

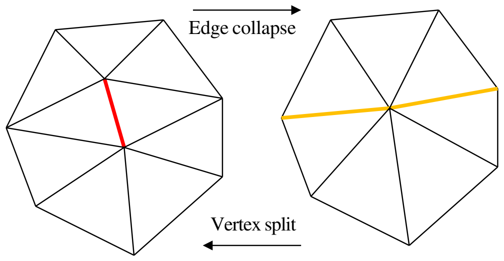


Fig. 1 Vertex split and edge collapse operations

time, so taking the client's viewpoint information, they only improve the LODs at particular parts which are visible for clients. By doing so, they can save network loads and reduce the data which needs to be transferred so as to improve streaming efficiencies. Note that here the mesh needs to be partitioned offline in advance, and when we get a request and viewpoint information from client, particular partitions are selected and transmitted.

2.2 Transmission of progressive meshes

In order to improve the transmission speed, some progressive compression algorithms use fast but lossy network protocols such as UDP. This kind of streaming protocol does not have re-transmission mechanism, so it is unreliable and may cause packet loss during streaming, which leads to crashing or distortions for decoding. Therefore packet loss handling becomes quite necessary. Yan et al. [24] and Al-Regib et al. [2, 4] separately propose different approaches using redundant bits. They measure the importance of the data in advance, and insert redundant bits accordingly to tolerate packet loss during transmission. Al-Regib and Altunbasak propose 3TP protocol [3]. They notice that the connectivity data, which represents the structure information, affects the quality of decoded mesh more than the geometric data. Therefore, by intelligently selecting the particular parts of 3D mesh to be transmitted over TCP and the rest over UDP, they could minimize the distortion caused by packet loss. Cheng et al. [6] suggest an analytical model to measure streaming of progressive meshes over lossy networks. But they focus on analyzing the effect of dependencies among split vertices to improve the qualities of decoded models, instead of directly dealing with the packet loss during transmission. The lost data in their method will be detected and retransmitted to clients.

All the above protection schemes belong to sender-based error control methods. Similarly, another kind of error handling category is receiver-based error control, which includes the robust transmission method by Bischoff [5].

Some approaches target on building a framework to deliver 3D mesh according to users' rendering capability and network conditions. Martin [17] suggests a framework to convert a 3D object to different modalities like 2D image, depth image, or polygonal meshes, and selects the appropriate modality to deliver. Li et al. [14, 16] use a generic 3D middleware between the 3D application layer and the transport layer. Their approach is a real time decision making algorithm that effectively uses the network resources to reduce both delaying and distortions. Their method can dynamically choose the transport protocol based on the network traffic.

Our error handling method belongs to the receiver-based error control category, and is inspired by some hole-filling techniques which are not used in progressive compression fields but in surface reconstructions. Pfeifle and Seidel [20] fill holes in piecewise polynomial surfaces after finding a spanning triangulation in 2D space. The method in [12] fills a hole based on a cloning technique. We expand these ideas in our loss tolerance scheme to reconstruct decoded meshes with only partial refinements data. We have published a conference version of our work in [22]. In this paper, we introduce the details of a general loss tolerance system with different modules and include two different stream constructions to further improve streaming efficiency. We also present more experimental results.

3 Progressive compression

Before introducing our loss tolerance scheme, we would like to first give an overview of progressive compression technique and related concepts, which are helpful to understand our work. Basically, there are two elementary mesh transformations in the progressive compression, namely vertex split and edge collapse, as depicted in Fig. 1. During compression, a sequence of edge collapse transformations is applied to remove batches of vertices and get a simplified 3D mesh and a series of refinements. The simplified mesh is called the *base mesh* which contains the basic topology of the original mesh. The refinements enclose *split bit*, *structural data* and *geometric data* for all removed vertices. *Split bit* is a bit corresponding to one vertex, indicating whether this vertex will split during decoding process; “1”means the vertex is a split-vertex and “0” otherwise. *Structural data* presents the connectivity of decoded new vertices with their neighbors. *Geometric data* stores the positions of decoded new vertices. During decompressing, vertex split transformations, the reverse operations of edge collapses, are applied to increase model’s level of details (LODs) gradually from base mesh to the original resolution or to the maximum LODs which users can afford based on their network conditions and display capabilities. For detail explanations of the progressive compression techniques, please refer to [10, 19, 23]. Different compression methods may have their own ways to explain those basic concepts, meanwhile they keep the same principles. In our algorithm, we follow the definitions from [10, 19], which is widely used in many applications.

4 Loss tolerance scheme

We propose a receiver-based loss tolerance method for 3D progressive streaming. Instead of protecting data from losing, we suggest a prediction method to recover the lost data. Moreover, we present two streaming construction strategies and one simplification constraint to control errors. Our scheme is able to provide a successful and effective solution of streaming 3D complex models over lossy networks.

4.1 System model

Figure 2 illustrates a general architecture of the proposed scheme, which is applicable to different progressive compression approaches. Besides common steps such as the mesh simplification, we include some additional modules in the architecture: refinements process, streaming process, synchronization and reordering and prediction module. Before explaining the details of those modules, we firstly introduce the system model and streaming strategies.

The server carries the compression procedure, through which we get a base mesh and a sequence of refinements. The mesh simplification process is the first step of compression,

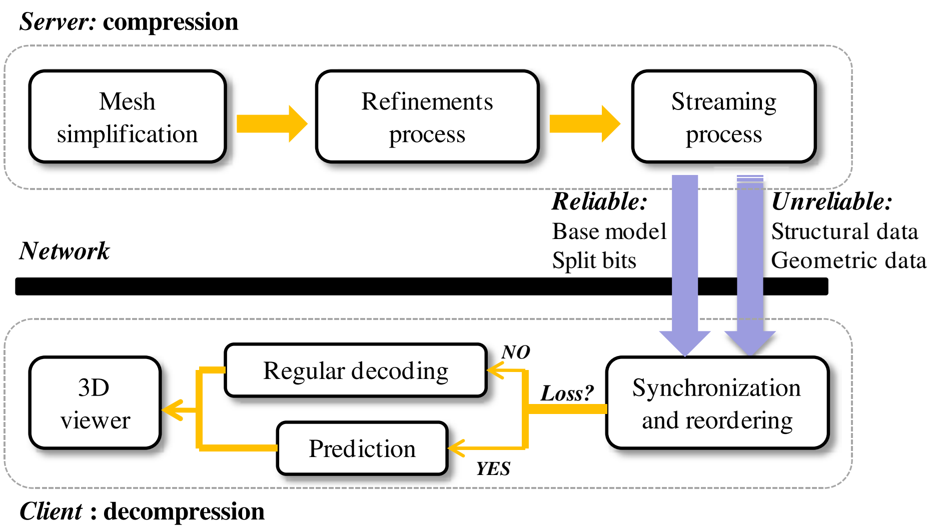


Fig. 2 Architecture of the proposed scheme

and many simplification techniques such as [10, 19, 23] can be applied to complete it. Then, a refinement process module is performed to extract split bits from other refinement data. Subsequently, depending on the network conditions such as the bandwidth, geometric and structural data are further constructed into smaller packets in the streaming process step. And then, base mesh, split bits and geometric and structural data are transferred separately using two channels. On the client's side, a decompression procedure is conducted to improve the 3D model's LODs progressively. Before decoding, we employ a synchronization and reordering process to correct packets orders, synchronize two streaming channels and detect packet loss. Then based on whether we have complete data for decoding or not, a regular decompression or our prediction algorithm is applied to obtain a detailed mesh and send it to 3D viewer.

In order to reduce distortions caused by packet loss during transmission, while keep a fast transmission speed, both TCP and UDP protocol are used in our system to deliver different parts of refinement data separately. Specifically, the base mesh and split bits are transmitted reliably while geometric and structural data are streamed over unreliable networks. A reliable channel using TCP can guarantee successful transmission and correct packet orders, but it may lead to longer delay due to the retransmissions and congestion control mechanisms. Unreliable channel, which follow UDP protocol, is fast and real-time streaming friendly but may suffer from packet loss. We choose to use both channels simultaneously to take their advantages. The base mesh which stores the basic topology of the 3D model is the foundation of the whole progressive compression procedure, so it is important to ensure its reliable delivery. Meanwhile, split bits carry the information of where the higher level of details is going to take places. By safely transmitting them to the client side, we are able to know which vertex should split so that we could keep the decoded mesh in a similar level of details as the original one. In our architecture, both structural data and geometric data are transferred over unreliable channel, because both of their data sizes are large and heavy to transfer. As shown in Table 1 [15], the bits which are used to encode a vertex's structural data and geometric data is not small. Therefore, we deliver both of them using UDP to achieve a high streaming speed.

Table 1 Bits used to encode structural and geometric data

Compression algorithm	Bits of structural data/vertex	Bits of geometric data/vertex
PFS	10	30
CPM	7.2	15.4

4.2 Stream construction

The last step of our compression procedure is the streaming process, which generate network packets before transmission starts. Since both the geometric and structural data are streamed unreliably, the packetization strategy over the UDP channel can influence the result of a packet loss and consequently affect the decompression outcomes. Thus we focus on the stream construction over UDP channel and propose two different ways to divide refinement information into small streaming packets, as illustrated in Fig. 3.

1. For each vertex split transformation, the corresponding structural and geometric data are bundled together as one basic *streaming fragment* (SF). Each SF contains the refinement and split-vertex id. Based on the SF, we divide each refinement into several packets, and each of them has one or more SF depending on network conditions. An example is presented in Fig. 3(a).
2. For a vertex split transformation, we treat its corresponding structural and geometric data as two basic SFs separately. Both of them have the refinement and split-vertex id. Stream construction based on this kind of SF can generate packets with any number of structural data and geometric data SF, as shown in Fig. 3(b).

The goal of using small streaming packet is to distribute the packet loss risk and improve streaming efficiency. Moreover, the packetization strategy may also influence the streaming results, and affect the decompression procedure accordingly. The first stream construction method put both structural and geometric data for a vertex split together, so whenever there is a packet loss, we lose both of them. On the other hand, in the second case, we may get different number of structural and geometric streaming fragments.

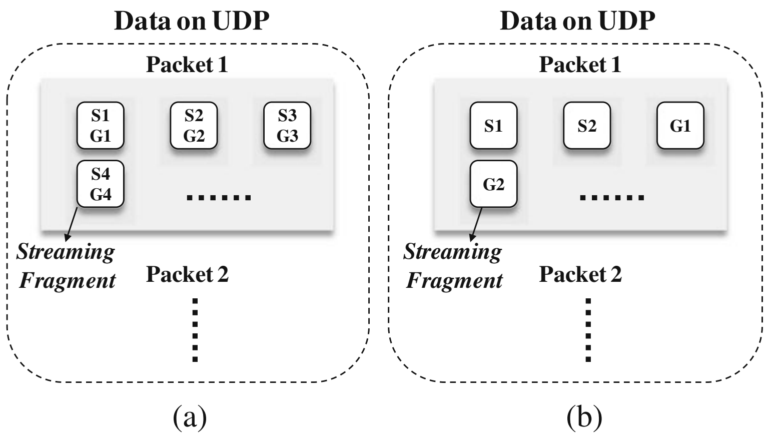


Fig. 3 Streaming packet on UDP

4.3 Synchronization, ordering and loss detection

On the client side, we include a module to deal with the synchronization, packet reordering and detection of packet loss before starting decompression. Based on the results from this module, we perform regular decoding or follow our prediction algorithm.

While two channels are used simultaneously in the system, we need to synchronize them and detect the end of the each refinement transmission on both channels. The synchronization between two channels can be accomplished by sending a flag message. For example, at the end of reliable part of each refinement, an END message containing the refinement id is sent to indicate the delivery end of this refinement. Similar message is also used on the unreliable channel. A simple acknowledgement-timeout-retransmission scheme with packet sequence number is used to handle possible loss of END messages over unreliable channels. We could start the reordering and loss detection process when both channels receive the END message.

Packets may arrive in different orders in unreliable channels, so we use a *stream reconstructor*, which works as a buffering pipeline, to reassemble them in the correct order. These packets can be either the structural data or the geometric data for a refinement. As described previously, we insert corresponding split-vertex id and refinement id on each part, so that the reconstructor can group the packets for the same refinement, and rearrange them in the correct orders using these ids. Another responsibility of the reconstructor is the packet loss detection. We compare both split bits and assembled structural and geometric data for the same refinement to detect the packet loss. Packets loss happens when a vertex's split bit is valued one which indicates it is a split-vertex, but there is no available corresponding structural data or geometric data or both. Depending on whether there is packet loss or not, we apply our prediction method or follow regular decompression procedure. After reordering and loss detection, the refinement can be delivered to the decoder for decompression, and the stream reconstructor can clean up its buffering for the next refinement.

Considering two different packetization strategies, after reordering and loss detection, we may face following four different cases to decompress one vertex split transformation: (1) both the structural and the geometric data arrive correctly at the client side; (2) both the structural and the geometric data are lost during transmission; (3) the structural data is lost but the geometric data arrives correctly; (4) the geometric data is lost but the structural data arrives safely. We apply the regular decompression procedure for the first case, and for the second case we use our prediction algorithm for both structural and geometric data. Case 3 and 4 appear only when we use the second packetization method. For case 3, we could use method similar to [8]. Specifically, based on successfully received geometric data, we obtain the coordinates of new vertices and then calculate their neighbors. Subsequently, neighboring vertices are used to derive the structural data, which is considered as a triangulation procedure. In case 4, only the geometric data is lost, so we only need to apply partial of our prediction method, namely the geometric prediction.

4.4 Prediction algorithm

We employ hybrid streaming channels to deliver different parts of refinement data separately, and to deal with the packet loss occurs over UDP, we propose a prediction algorithm. Packet loss refers to data failing to reach its destination when transmitted across networks. Unreliable channels using UDP have no recovery mechanism for packet loss, resulting in incomplete streaming data. So we may only have partial refinement data. Then, the decoder either aborts the decoding process which leads to a significant visual degradation during rendering, or makes a random guess which may introduce connectivity inconsistencies and cause decoding crash. Therefore, we propose a receive-based loss prediction algorithm to recover lost data and predict decompression results, as illustrate in Fig. 4.

1. Find the least-square best fitting plane of the one-ring region which consists of the split-vertex and its neighbor vertices, and project them onto this plane. We take this plane as a parametric plane.
2. In the parametric plane, find the farthest edge from the split-vertex, denoted as *e*.
3. Insert a vertex in the triangle formed by the split-vertex and vertices of edge *e*. The inserted vertex is the new vertex.
4. Connect the new vertex to the split-vertex and two vertices of edge *e* to form three triangles.
5. Apply edge flipping to ensure Delaunay triangulations of the whole polygon including the new vertex, the split-vertex, neighbor vertices and all corresponding edges.
6. Update the coordinates of new vertex and split-vertex as following:

   a. For a selected vertex v, find its neighbor vertex v_i and assign a weight to its neighbor using Eq. 1:

   $$w_i = \frac{1/\|v - v_i\|}{\sum_{j=0}^{n} 1/\|v - v_j\|}, \tag{1}$$

   where $\|v - v_i\|$ represents the Euclidean distance of vertex v and v_i in the parametric plane. We have n neighbors in total, with the summation of all weights being unity and the neighbor vertex which is closer to v_i having a higher weight.

   b. New coordinates of the vertex v can be computed by Eq. 2:

   $$v_{new} = \sum_{i=0}^{n} w_i v_i \tag{2}$$

   c. Update the vertex v with its new coordinates.

Our prediction method is inspired by some hole filling techniques [12, 20]. Compared with general hole-filling techniques which typically deal with large holes and complicated boundaries, our vertex prediction is a rather simpler problem, because only a one-ring "hole" of the missing vertex needs to be filled. The calculation only relies on the one-ring neighborhood information of the missing vertex, so it takes constant time for predicting a

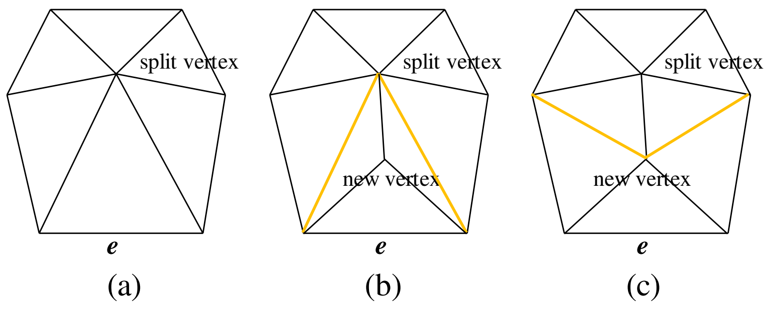


Fig. 4 Loss prediction algorithm: step 2 to step 5. **a** On the best fitting plane, find out the split vertex, and the farthest edge *e*. **b** The new vertex, the split vertex and vertices of edge e form three triangles. **c** The polygon after applying Delaunay triangulation

vertex. Observing that an edge collapse and its corresponding vertex split transformation are local operations and take place in the one-ring neighborhood of the split-vertex which is normally a very small region, we can project this region in 2D space, called the least-square best fitting plane. This plane serves as the parametric plane, on which we employ a simplified hole filling technique to predict new vertices. Vertices included in this small region are fairly limited amount, typically 6 to 7, which means for each vertex prediction there is only 6 to 7 vertices involved. Thus, if we lose information of one vertex split, the corresponding prediction following our method can be completed in constant time. Assume there are n vertices which needs to be predicted during decompression, the prediction procedure will take $O(n)$ time. Therefore, our approach is very effective and satisfies the requirement of real time streaming.

In addition, we are able to obtain decompressed meshes with high qualities using the proposed prediction algorithm. As mentioned in [3], the structural data affects the quality of a decoded mesh more than the geometric data does. The decoded mesh with valid triangulations still maintains the same topology as the original 3D model. Our streaming strategy guarantees that split bits are received successfully, notifying us where a higher level of details is needed on the current 3D mesh. Moreover, the neighbor vertices of split-vertex also give additional information. Consequently, we can generate a valid triangulation for each split vertex using proposed method. Furthermore, we adjust geometric coordinates of new vertices according to the partition of unity method [18]. Following a linear combination, we calculate the location of the new vertex as a weighted sum of its neighbors. The neighbors closer to the new vertex play a more important role in evaluating the coordinates. Thus, our prediction generates a good approximation of decompressed mesh.

4.5 Control of error propagation

In the last stage of the decompression procedure, all predicted new vertices are marked as unreliable [13]. For every vertex, we use one bit, called reliability bit, to indicate whether this vertex is from prediction or regularly decoding. By doing so, we are able to track the vertices with predicted connectivity and coordinates to ensure validity of our further decoding, and also to prevent the prediction errors from propagating to other regions on the mesh.

A packet loss may refer to one or more vertices loss depending on the packetization strategy. If we treat the refinement of a vertex split transformation as a packet and streaming it over UDP, when a packet loss occurs, we lose information of one vertex split. If refinement information of several continuous vertex split operations is constructed into one packet and streamed over unreliable channel, one packet loss can lead to losing of a group of neighbor vertices which may cause distortion propagation on the predicted meshes. Therefore, to prevent burst loss and error propagation during compression, we introduce a mesh simplification constraint: at most one edge be collapsed into one vertex. This constraint ensures that at most two vertices may be collapsed into one. As a result, one split-vertex in mesh M_i corresponds to only one single edge in M_{i+1}. Assume that there are more than one edge collapsed into one vertex, and these vertex split transformations are packetize into one packet and streamed over UDP. When packet loss happens, we need to predict more than one new vertex in a small one-ring neighbor region. Subsequently, a vertex split based on a predicted vertex could spread errors from the previous prediction. We explain this prediction error propagation problem using an example, as depicted in

Fig. 5. In a small region on the mesh, two edges *AB* and *BC* collapse into one vertex *C*, as shown in Fig. 5(a) and (b). So during decompression, we sequentially insert two new vertices, vertex *B* and vertex *A*, based on vertex *C*. When packet loss occurs on unreliable channels, we lose refinement information of both vertex *A* and *B*. Thus, the prediction method is applied to get the new vertex *B* (Fig. 5(c)), and then based on the predicted vertex *B*, a subsequent prediction is made to get the vertex *A* (Fig. 5(d)). Note that the prediction error of the vertex *B* is propagated to the vertex *A*. Hence, in order to prevent this kind of error propagation, we always follow the above constraint. By doing so, we can ensure only one edge collapses into one vertex, and a packet loss will not bring burst loss. This constraint can be simply accomplished by a checking step: during mesh simplification, when we select an edge to perform edge collapse transformation, we first check whether it shares a vertex with another collapse-edge. If not, it can be a candidate collapse-edge, and edge collapse happens only on those candidate edges.

5 Performance evaluation

5.1 Error metrics

Distortion is served as a metric to evaluate the decoding performance of our system. We focus on two types of distortions, namely the geometric distortion like differences at vertex positions, and the visual difference such as curvature dissimilarities. Two widely used techniques, *Hausdorff Distance* [7] and *Laplacian Operator* [11], are utilized to capture these two kinds of distortions separately. Even though packet loss occurs in different segments of a refinement, instead of individually comparing the distortion of each fragment [14], we compare the completely decoded 3D mesh with and without packet loss and consequent prediction. Equation 3 gives the definition of the *Hausdorff Distance* between two meshes *M* and *M*′:

$$H(M,M') = max\{h(M,M'),h(M',M)\}, \tag{3}$$

where $h(M,M') = max_{a \in M} min_{a' \in M'} ||a - a'||$, and $\|a - a'\|$ represents the Euclidean distance between vertex a and vertex a'.

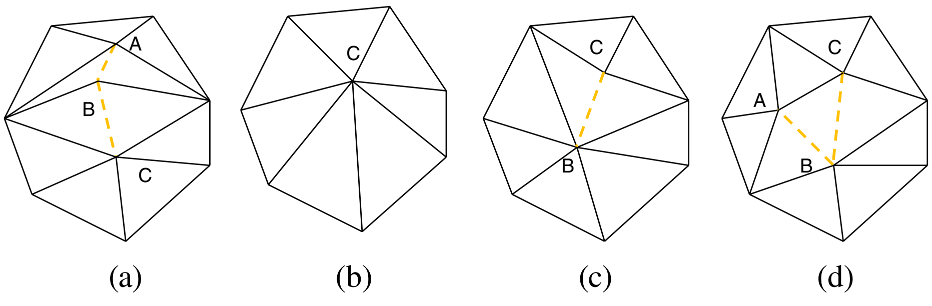


Fig. 5 Prediction error propagation. **a** Two edges *AB*, *BC* are collapse edges; **b** Collapse edges *AB*, *BC* from (**a**) to vertex *C*; **c** Using prediction method to get vertex *B*; **d** Based on *B* using prediction method to get vertex *A*

The *Laplacian Operator* captures the visual difference such as curvature and smoothness between two meshes. It requires that the two compared meshes have the same vertex number, which is fulfilled in our approach. The Visual difference between mesh M and M' is given by the following Eq. 4:

$$\|M - M'\| = \frac{1}{2n}(\|v - v'\| + \|GL(v) - GL(v')\|), \tag{4}$$

where v is the set of vertices on mesh M, and v' is the set of vertices on mesh M'. $GL(v)$ is the value of the geometric Laplacian of v, and similarly, $GL(v')$ is the geometric Laplacian of v'. n is the number of vertices on the mesh. For any vertex v_i, the geometric Laplacian is defined as the Eq. 5:

$$GL(v_i) = v_i - \frac{\sum_{j \in N(i)} l_{ij}^{-1} vj}{\sum_{j \in N(i)} l_{ij}^{-1}}, \tag{5}$$

where l_{ij} denotes the geometric distance between vertices i and j, and $N(i)$ is the set of indices of neighbors of vertex i.

5.2 Simulation results

The proposed loss-tolerance scheme is generally applicable to different progressive algorithms, and we choose to implement it based on CPM for experiments. Our system is tested on a network with packet loss ratio l between 2% and 12%, and to be more realistic, we follow random packet loss pattern. In our experiments, 35%–40% more vertices are added in every refinement. For illustrating our simulation results, we use the Standford "Bunny" model (34843 vertices, 2.96 MB) and the Cyberware "Horse" model (48485 vertices, 4.74 MB).

Our experiments focus on testing how the prediction method works during the decompression under different data loss ratios. We present the distortion values and screen shots of decoded meshes in different level of details and with various loss ratios. The performance of experimental results is evaluated using both the geometric distortion and the visual distortion simultaneously. Figures 6 and 7 explain the geometric distortions of the Bunny and the Horse respectively, using Hausdorff Distance. Figures 8 and 9 demonstrate the results' visual distortions using *Laplacian Operator*. Mesh sequence in Fig. 10 shows the decoded progressive meshes of the Bunny model using proposed method. In Fig. 11, we use increased percentage of decoding time to explain the computational cost of our approach, based on the regular decoding time without predictions for packet loss.

From Figs. 6 to 9, we can see that with the increasing of loss ratio, the distortion increases and the mesh surface becomes less smooth. This phenomenon is because large loss ratio implies more packets lost during transmission, then more new vertices are predicted which leads to higher possibility of distortions. When the refinement increases, distortion also raises accordingly. There are some packets losses in each refinement, more refinements means more packets losses totally. As shown in those figures, the distortion caused by our prediction is only a very small percent of the whole model. Therefore, we can conclude that distortions caused by prediction are very limited. As shown in Figs. 6 and 7, distortions do not change linearly. This may relate to vertices' importance. As mentioned in [4, 6], every split-vertex has an importance value and there are dependencies between split vertices. Once random loss occurs on those split vertices with high importance, results may have more distortion.

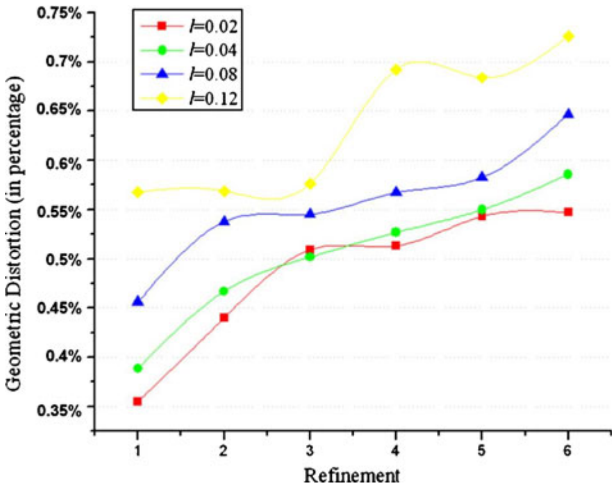


Fig. 6 Geometric distortion (*Hausdorff Distance*) between the zero-loss decoded and the proposed decoded Bunny model over different loss ratios *l*. Distortion is presented w. r. t. the size of the model

Therefore, taking into account the importance of split vertices before streaming may further improve our results.

As described in Fig. 11, the decoding time increases when packet loss ratio is larger. This is because high loss ratio implies that we have more vertices which need to be predicted. Figure 11 also proves that the proposed scheme is very efficient. Even when the loss rotio *l* is high as 12%, which happens rarely in real applications, we only introduce about 10% more decoding time. Note that it takes longer time to decode the Horse model than the Bunny model, which is only because the previous has more vertices. The decoding time is largely related to the hardware performance, so it can be further reduced if the client has more powerful computing hardware.

As mentioned previously in Section 4.2, two different stream construction methods are proposed, so we compare our prediction results based on these two packetization strategies.

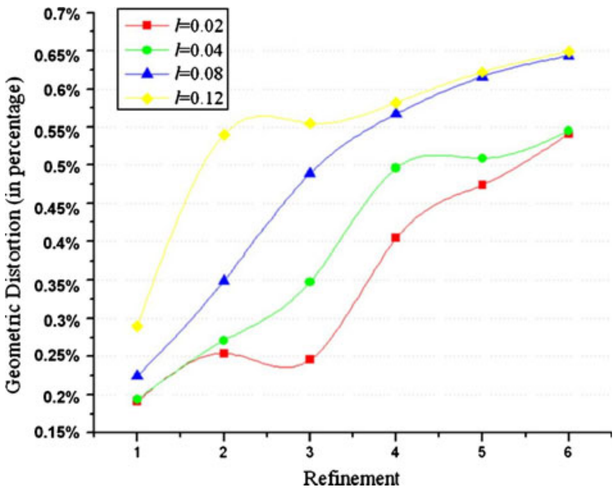


Fig. 7 Geometric distortion (*Hausdorff Distance*) between the zero-loss decoded and proposed decoded Horse model over different loss ratios *l*. Distortion is presented w. r. t. the size of the model

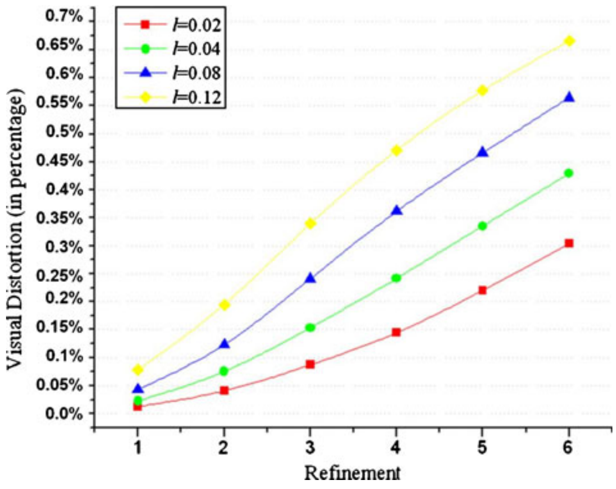


Fig. 8 Visual distortion (*Laplacian Operator*) between the zero-loss decode and proposed decoded Bunny model over different loss ratios *l*. Distortion is presented w. r. t. the size of the model

The geometric distortions and visual distortions between the zero-loss decoded and proposed decoded bunny model over loss ratio 12% (l=0.12) are shown in Figs. 12 and 13 separately. We can see that the second packetization strategy is better, because structural and geometric data are packetized separately, so when packet loss happens, we may still get partial refinement information of a split-vertex, as case 3 and 4 in Section 4.3. Lost both structural and geometric data for a vertex split transformation in the second strategy does not occur very frequently.

5.3 Comparison with Priors works

We have compared the proposed method with some existing mesh streaming techniques. Comparison results confirm that our algorithm is able to achieve high quality decoded meshes with lower distortions.

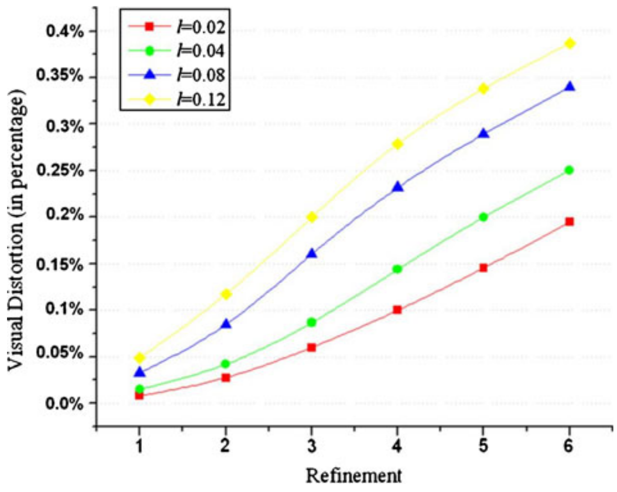


Fig. 9 Visual distortion (*Laplacian Operator*) between the zero-loss decode and proposed decoded Horse model over different loss ratios *l*. Distortion is presented w. r. t. the size of the model

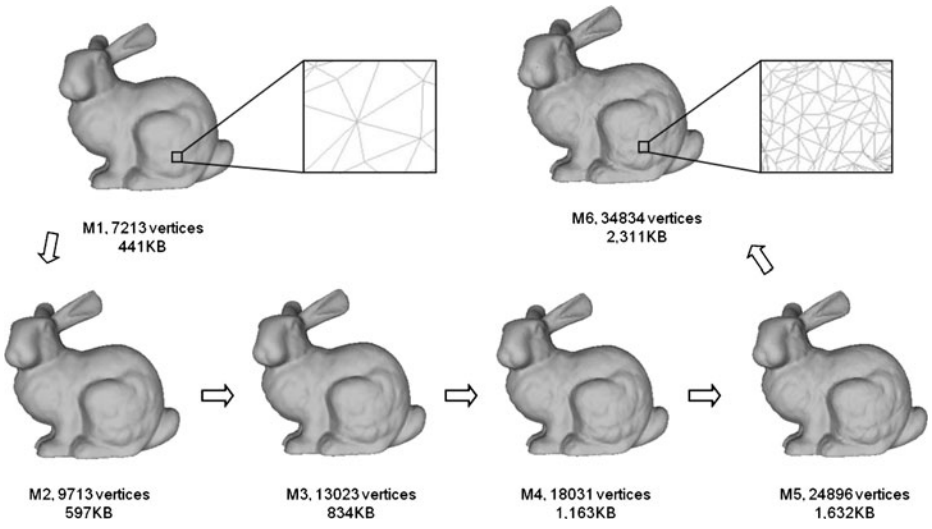
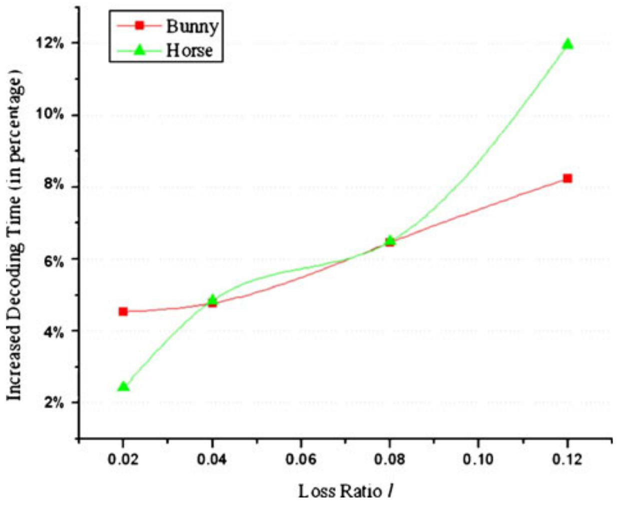


Fig. 10 Sequence of meshes for the Bunny model over loss ratios 0.12

An extreme and easy way to handle packet loss is simply abort the vertex split transformations wherever the corresponding refinements are lost during transmission. Moreover, to avoid crashing at later decompression phases, all the subsequent vertex split transformations based on the lost ones need to be abandoned as well. We name this method *Loss-without-prediction* (LwoP) and compare it with our method. Experiments show that around 70% of our results are better than the LwoP method, and we can improve decompression results and present smoother meshes. The comparison results are illustrated in Figs. 14 and 15. In LwoP, whenever a vertex's refinement information is lost in some level, the consequent mesh area stops at that resolution. Therefore, after decoding, the decoded mesh $M_{\max}$ can have different parts in different level of details. Moreover, the

Fig. 11 Increased decoding time of the Bunny and the Horse models using the proposed scheme over different loss ratio l on Pentium IV machines. Increased decoding time is presented w. r. t. the zero-loss decoding time



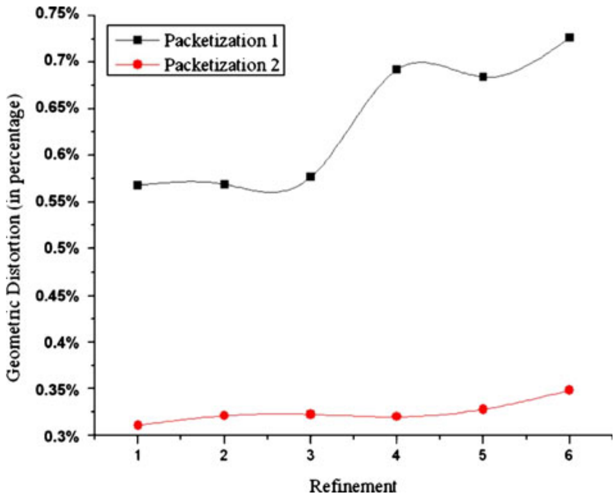


Fig. 12 Using two packetization methods, geometric distortion (*Hausdorff Distance*) between the zero-loss decoded and proposed decoded Horse model over loss ratios $l=12\%$. Distortion is presented w. r. t. the size of the model

decoded meshes have less vertex number than the original ones. For example, after three refinements, the decoded mesh with size 755 KB has 12063 vertices using LwoP, while using the proposed technique the mesh has 13023 vertices with size 834 KB. In our method, the prediction results in smooth decoded meshes, provides better visual effect and keeps same number of vertices. In addition, when the data loss ratio is higher, which implies more packet loss occurs during transmission, the decoded meshes have more distortions using LwoP. Furthermore, if the base mesh is in a very low resolution, which means we need more refinements data to decompress the base mesh into the original mesh, the results using LwoP are not satisfying.

We compare our approach with another receiver-based progressive streaming method proposed in [5]. Based on geometric distortion using *Hausdorff Distance*, comparison results are depicted in Fig. 16. Obviously, distortions of the decoded meshes using proposed

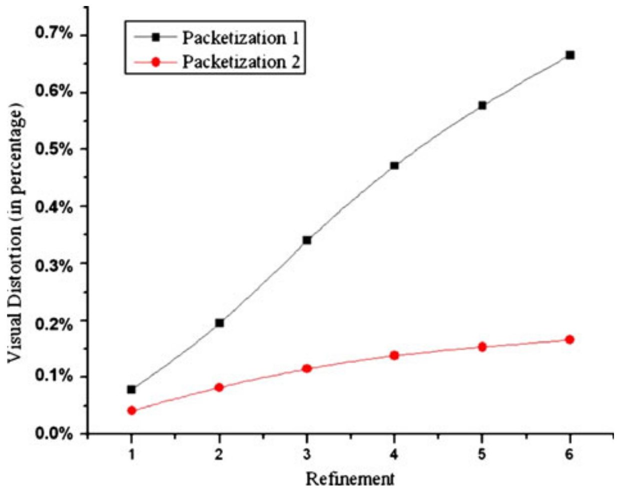


Fig. 13 Using two packetization methods, visual distortion (*Laplacian Operator*) between the zero-loss decoded and proposed decoded Horse model over loss ratios $l=12\%$. Distortion is presented w. r. t. the size of the model

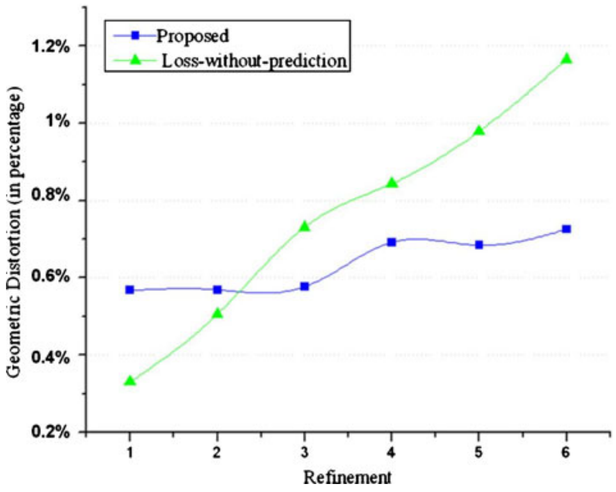


Fig. 14 Geometric distortion (*Hausdorff Distance*) between decoded Bunny model with zero-loss and with loss ratio 0.12. Proposed: decoding using proposed method; Loss-without-prediction: decoding using loss-without-prediction method

technique are much smaller than the ones following [5]. In comparison to their method, we have few advantages. Since split bits are transmitted reliably in our algorithm but not [5], we have a better understanding of where the high level of details are needed, and our approach tends to have more vertices and triangles in decoded mesh. They only transfer the geometric data, so it cannot ensure the correctness of always selecting the closest triangle to insert new vertex.

We compare our streaming strategy with CPM [19]. In our streaming strategy, both reliable and unreliable channels are included to distribute the transmission loads which are all over the reliable channel in CPM. So our streaming procedure can be speeded up. We are able to effectively reduce at least half of the data which needs to be delivered over reliable channel, as shown in Table 2.

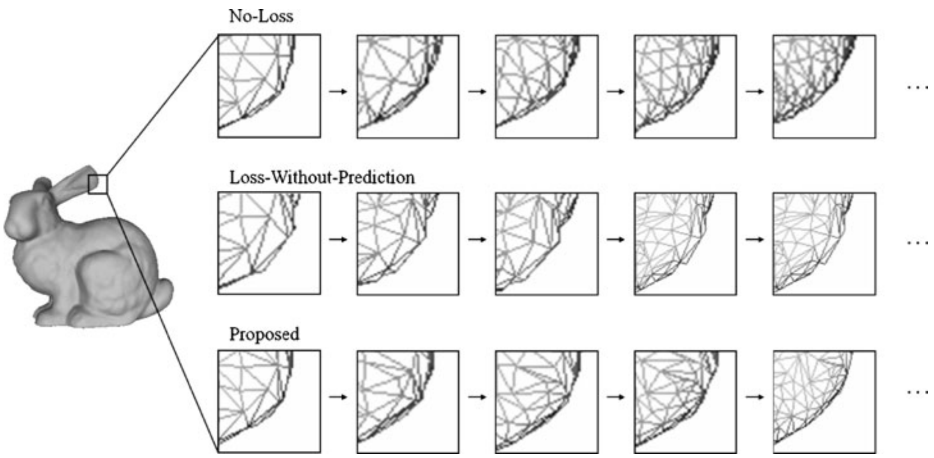


Fig. 15 The decoded Bunny model using Loss-without-prediction and the proposed method over loss ratios l=12%

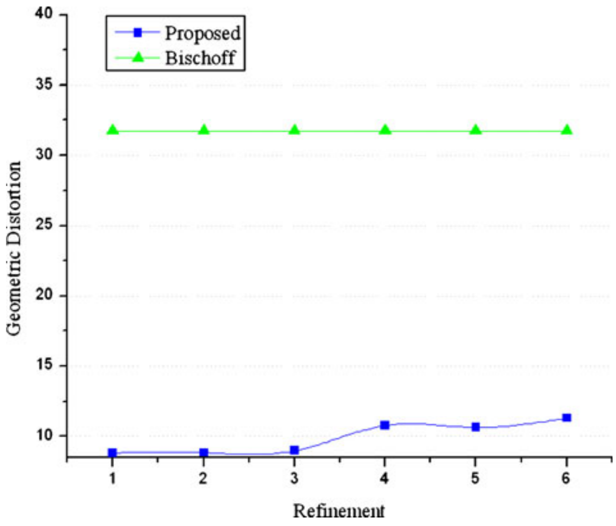


Fig. 16 Geometric distortion (*Hausdorff Distance*) between decoded Bunny model with zero-loss and with loss ratio 0.12. Proposed: decoding using proposed method; Bischoff: decoding using Bischoff algorithm

We have also tried to compare our results with methods in [2, 4], however, they illustrate their errors (*Hausdorff Distance*) using absolute values without explicitly indicating the size of their models. Thus, we are not able to directly perform comparisons. We try to compare the efficiency of our approach with [6, 25]. Their decoding time includes data retransmission time when packet loss happens. Our work focuses more on the modified compression scheme (on server side) and the prediction scheme to tolerate packets losses (on the client side), so our decoding time does not have retransmission delay and is used to measure the effectiveness of our prediction scheme and decompression. Thus, it is not appropriate to perform direct comparison between [6, 25] and our approach.

6 Conclusion and future works

In this paper, we have proposed a receiver-based loss-tolerance scheme to deal with the packet loss when 3D progressive meshes are transmitted over lossy networks. We utilize a reliable channel with TCP protocol to transmit the base mesh and the split bits, and follow a fast channel using UDP to deliver both structural and geometric data. To

Table 2 Data size (Bits/Vertex) over reliable and unreliable channels. CPM: data size in CPM algorithm; Proposed: data size in our proposed method

Mesh	CPM	Proposed	
	Reliable	Reliable	Unreliable
Bunny	8.98	4.3	4.68
Horse	7.51	2.87	4.64

further improve streaming efficiency, in the encoding procedure, refinements are subdivided into smaller packets according to network bandwidths. During decompression, we use a stream reconstructor to synchronize reliable and unreliable channels, reorder packets, and detect packet losses. Whenever there is packet loss, the proposed loss tolerance scheme is applied to reconstruct the new vertices' connectivity and estimate their coordinates. Our method does not require any additional data from server to recover packet loss, and it is not necessary to retransmit lost data for decoding so there is no extra transmission load over networks and no extra transmission delay. We are capable of predicting both the structural data and the geometry data for a lost vertex split transformation. Our prediction is able to provide meshes with good decompression quality without incurring high computational cost. We suggest a simplified hole-filling technique to effectively perform prediction so that it takes only $O(n)$ time to accomplish prediction of n vertices loss.

The technique discussed in this paper is a generic solution for different compression algorithms, and we have explained our result by integrating with CPM [19]. As shown in the performance evaluation, the distortion introduced by prediction is fairly small and the computational cost for predicting lost data is low. In our system, we are able to achieve high transmission speeds for streaming progressive meshes due to the use of hybrid protocols, so it is suitable for real-time applications. Moreover, we suggest two different packetization methods to further improve streaming efficiency and distribute loss risks. And we have illustrated the differences of two methods in the experiments.

In the future, we will include an intelligent transmitting mechanism to control communications at various network conditions and avoid transmitting unnecessary refinements which will not be used during decompression due to predictions. Moreover, we will consider integrating the partition methods [13] to further reduce the distortions in our results. The importance of vertices will also be included to further improve our results in the future.

References

1. Alliez P, Desbrun M (2001) Progressive compression for lossless transmission of triangle meshes. In: Proceeding of SIGGRAPH'01, pp 195–202
2. Alregib G, Altunbasak Y(2002) An unequal error protection method for packet loss resilient 3D mesh transmission. In: Proceedings of the IEEE International Conference, vol. 2. Orlando, FL, pp 2041–2044
3. Alregib G, Altunbasak Y (2004) 3TP: 3D models transport protocol. In: Proceedings of the 9th International Conference on 3D Web Technology, pp 155–162
4. Alregib G, Altunbasak Y, Rossignac J (2005) Error-resilient transmission of 3D models. ACM Trans Graph 24:182–208
5. Bischoff S, Kobbelt L (2002) Towards robust broadcasting of geometric data. Comput Graph 26:665–675
6. Cheng W, Ooi WT, Mondet S, Grigoras R, Morin G (2007) An analytical model for progessive mesh streaming. In: Proceedings of 15th ACM International Multimedia Conference, September 24–29, pp 737–746
7. Cignoni P, Rocchini C, Scopigno R (1998) Metro: measuring error on simplified surfaces. Comput Graphics Forum 17(2):167–174
8. Devililers O, Gandoin P (2000) Geometric compression for interactive transmission. In: Proceedings of IEEE Visualization, pp 319–326

9. Guan W, Cai J, Zheng J, Chen C (2008) Segmentation based view-dependent 3D graphics model transmission. IEEE Trans Multimedia 10(5):724–734
10. Hoppe H (1996) Progressive meshes. In: Proceedings of SIGGRAPH'96, pp 77–108
11. Karni Z, Gotsman C (2000) Spectral compression of mesh geometry. In: Proceedings of SIGGRAPH'00, pp 279–286
12. Kraevoy V, Sheffer A (2005) Template based mesh completion. In: Proceeding of Eurographics Symposium on Geometry Processing (SGP), pp 13–22
13. Li H (2007) Streaming 3D progressive meshes over lossy networks. Doctoral Thesis. University of Texas at Dallas
14. Li H, Li M, Prabhakaran B (2006) Middleware for streaming 3D progressive meshes over lossy networks. ACM TOMCCAP 2(4):282–317
15. Li H, Shan P, Prabhakaran B (2004) Smart decision module for streaming 3D meshes over lossy networks. In: Proceedings of DMS/VLC'04, pp 275–278
16. Li H, Tang Z, Guo X, Prabhakaran B (2008) Loss tolerance scheme for 3D progressive meshes streaming over networks. In: Proceedings of IEEE ICME, Germany, pp 501–504
17. Martin I (2000) Adaptive rendering of 3D models over networks using multiple modalities. Technical Report, IBM T. J. Watson Research Center RC 21722, 97821 (April)
18. Ohtake Y, Belyaev A, Alexa M, Turk G, Seidel H (2003) Multi-level partition of unity implicits. ACM Trans on Graphics 22(3):463–470
19. Pajarola R, Rossignac J (2000) Compressed progressive meshes. IEEE Trans Vis Comput Graph 6 (1):79–93
20. Pfeifle R, Seidel H (1996) Trianglar B_Splines for blending and filling of polygonal holes. In: Proceeding of Graphics Interface '96, Morgan Kaufmann Publisher, pp 186–193
21. Southern R, Perkins S, Steyn B, Muller A, Marais P, Blake E (2001) A stateless client for progressive view-dependent transmission. In: Proceedings of the Web3D Symposium
22. Tang Z, Guo X, Prabhakaran B (2009) A comprehensive approach for streaming 3D progressive meshes. In: Proceedings of IEEE International Symposium on Multimedia (ISM 2009), pp 263–268
23. Taubin G, Gueziec A, Horn W, Lazarus F (1998) Progressive forest split compression. In: Proceedings of SIGGRAPH'98, pp 123–132
24. Yan Z, Kumar S, Kuo C (2001) Error resilient coding of 3D graphic models via adaptive mesh segmentation. IEEE Trans Circuits Syst Video Technol 11(7):860–873
25. Yang B, Pan Z, Li F, Wang X (2008) An effective error resilient packetization scheme for progressive mesh transmission over unreliable networks. In: Proceeding of Computer Graphics International, pp 18–26

Ziying Tang received her B. S. degree in computer science from the Zhejing University and M. S. degree from the International Univesity in Germany. She is currently a Ph. D. candidate in the Department of Computer Science, University of Texas at Dallas, Richardson. Her search interests include multimedia systems, computer graphics and computer networks, with an emphasis on streaming of 3D animations and deformable 3D models, and haptic related problems.

Xiaohu Guo is an assistant professor of computer science at the University of Texas at Dallas. He received the PhD degree in computer science from the State University of New York at Stony Brook in 2006. His research interests include computer graphics, animation and visualization, with an emphasis on geometric and physics-based modeling. His current researches at UT-Dallas include: spectral geometric analysis, deformable models, centroidal Voronoi tessellation, GPU algorithms, 3D and 4D medical image analysis, etc. For more information, please visit http://www.utdallas.edu/_xguo. He is a member of the IEEE.

Balakrishnan Prabhakaran is currently a Professor with the Department of Computer Science, University of Texas at Dallas, Richardson. He received the Ph.D. degree in computer science from the Indian Institute of Technology Madras, Chennai, India. He has been involved in the area of multimedia systems: animation and multimedia databases, authoring and presentation, resource management, and scalable Web-based multimedia presentation servers. He was a Visiting Research Faculty with the Department of Computer Science, University of Maryland, College Park. He was also a faculty member in the Department of Computer Science, National University of Singapore, as well as in the Indian Institute of Technology Madras.