Abstract—Fast and efficient streaming of detailed 3D model over lossy network has long been a challenge, although progressive compression techniques were proposed long time ago. One reason is that packet loss occurring in unreliable networks is highly unpredictable, and leads to connectivity inconsistency and distortions. In this paper, we address this problem by proposing a receiver-based loss tolerance scheme based on a prediction technique. Our method works without introducing protection bits and retransmission. 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 tests indicate that the decompression is completed quickly, suggesting that it is a practical solution. Moreover, the proposed prediction technique achieves a good approximation of the original mesh with low distortion.

I. INTRODUCTION

Detailed 3D models have been widely used in network multimedia applications such as online games, distributed simulation and Internet shopping. The amount of data representing a high quality 3D mesh may be huge, with thousands to millions of vertices to represent its geometry, and connectivity relationship among vertices to denote the topology. Therefore, it is a challenge to quickly transmit detailed 3D meshes to end users with acceptable quality.

Progressive compression techniques [1, 2, 3, 4] address this problem by sending to users a crude mesh followed by a sequence of refinements to progressively improve meshes’ quality. However, these methods may result in an undesired delay due to the data retransmission mechanism. To alleviate the delay, methods utilize hybrid TCP and UDP connections are suggested in [5, 6, 7]. The use of UDP can lead to packet loss, so methods have been developed to enable packet loss recovery. Unfortunately, existing methods for recovery either introduce extra protection information which leads to network overload, or they simply retransmit the lost packet which causes the time delay.

In this paper, we propose a loss-tolerance streaming strategy in which the packet loss is handled without recovery bits and retransmission. We introduce a prediction method to obtain an approximated decompression result. Since vertex split takes place in a one-ring neighborhood which is a small region, we can project this region in 2D and apply a hole filling technique to insert new vertices and perform triangulation. In addition, we adjust new vertices’ geometric coordinates using the idea of the partition of unity method [8]. Note that vertex split is a local operation and involves small amount of neighbor vertices, our prediction is very efficient and takes $O(n)$ time to predict $n$ lost vertices. We impose a constraint for mesh simplification in addition to the use of reliability bit [9]. Experimental results show that our prediction gets good results and controls prediction error in a very limited area.

Similar to [6, 9], we use hybrid channels to deliver different parts of encoded information separately. A reconstructor is utilized to synchronize hybrid channels, reassemble packets and detect packet loss. During decompression, depending on whether complete

![Figure 1. Architecture of the proposed scheme.](image-url)
refinement data for decoding is available, a regular decompression or our prediction method is applied to obtain a detailed mesh. Figure 1 illustrates the general architecture of our system.

Our work has the following novelties: 1) we propose an efficient receiver-based loss tolerance scheme that works independently of channel behaviors; 2) both the structural data and the geometry data of lost vertex are predicted; 3) decoding can be accomplished successfully without any retransmission and recovery bits; 4) the prediction provides good decompression quality without high computational cost.

II. RELATED WORKS

Our work is associated with the progressive compression techniques and related works in transmission.

Progressive Mesh (PM) [1] as the first progressive compression technique obtains a simplified base mesh by performing a sequence of edge collapse operations. In decompression, a series of vertex split operations are processed. Both Progressive Forest Split (PFS) [2] and Compressed Progressive Meshes (CPM) [3] group edge collapses into batches to achieve a higher compression ratio. Alliez and Desbrun introduce a valence-driven decimating approach [4] for lossless transmission of triangle meshes. From a different point of view, [10] try to save network loads. Considering that users cannot see a whole model’s all parts at one time, they only transmit the particular part of the mesh that is visible for clients. In this method, the mesh needs to be partitioned offline in advance.

Some progressive compression algorithms are based on lossy networks to achieve fast transmission. The unreliability of lossy networks makes it is necessary to consider error handling. AlRegib et al. [5] propose error recovery approaches using redundant bits. Similarly, authors in [11] suggest a receiver-based error control method. AlRegib and Altunbasak propose 3TP protocol [6]. By transmitting part of 3D mesh over TCP and the rest over UDP, they could reduce the distortion caused by packet loss. Cheng et al. [7] suggest an analytical model to measure the effect of dependencies among split vertices before streaming. In this method lost data retransmitted.

III. LOSS TOLERANCE SCHEME

We employ both TCP and UDP to deliver different parts of refinement data. Packet loss refers to data failing to reach its destination when transmitted across network. Unreliable channels like UDP have no recovery mechanism for packet loss, resulting in incomplete refinement data for decoding. In this case, the decoder either aborts the decoding process which leads to a significant visual degradation, or makes a random guess which may introduce connectivity inconsistencies and crash. We propose a receiver-based loss tolerance method to recovery lost data and predict decompression results. Our method is explained as follow with Figure 2:

1. Find the least-square best fitting plane of the one-ring region of the split vertex and its neighbor vertices, and project them onto this plane.
2. In this parametric 2D 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 triangulation of the whole polygon.
6. Update the coordinates of new vertex and split vertex as below:
   a. For the selected vertex \( v \), find its neighbor vertex \( v_i \) and assign a weight:
      \[
      w_i = \frac{1}{\| v - v_i \|},
      \]
      where \( \| v - v_i \| \) represents the Euclidean distance in the parametric plane. We have totally \( n \) neighbors with summation equals to unity. The closer neighbor has a higher weight.
   b. New coordinate of the vertex \( v \) is:
      \[
      v_{new} = \sum_{i=0}^{n} w_i v_i .
      \]
   c. Replace the vertex \( v \) with its new coordinates.

The prediction method is inspired by hole-filling techniques [12]. Compared with general hole-filling techniques, our vertex prediction is rather simpler, because only the one-ring "hole" of the missing vertex needs to be filled. We project the one-ring neighborhood in 2D, called the least-square best fitting plane, and use it as the parametric plane to employ a simplified hole filling technique. The computation only relies on the one-ring neighborhood of the missing vertex. Vertices included in this small region are very limited amount, typically 6 to 7. Hence, if we have lost one vertex split information, the corresponding prediction using our method takes only constant time. Assume we need to predict \( n \) vertices during decompression, the prediction requires \( O(n) \) time.

Our scheme provides approximated decompression meshes with good quality. As mentioned in [6], the
structural data affects the quality of a decoded mesh more than the geometric data. The decoded mesh with valid triangulation maintains the same topology as the original 3D model. In addition to the neighbor vertices, the successfully transmitted split bits implies where a higher level of details is needed. Therefore, we can generate a valid triangulation for each split vertex using proposed method. Moreover, we adjust geometric coordinates of new vertices according to the partition of unity method [8]. Follow 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.

During compression, mesh simplification, extract split bits and construct streaming packets steps only need to be performed once offline. The separating process is straightforward: by going through each refinement, we generate two files storing split bits, and structural and geometric data respectively. In addition, for a split vertex, we keep an index such as vertex id and refinement id on both split bit and corresponding structural and geometric data. By doing so, it is easy to detect packet loss and retrieve information during decompression.

In our system, we employ reliable channel to stream the base mesh and split bits, meanwhile use unreliable channel to deliver structural and geometric data. The base mesh which stores the basic topology of the 3D mesh should be delivered with the highest priority. The split bits in each refinement comprise the information of where the higher level of details is going to take place. So they should be transmitted reliably. Even though it is not compulsory to maintain the same number of vertices, we need to keep a similar level of details on the decompressed mesh. Both structural and geometric data are transferred over UDP in our method, because their sizes are large and time-consuming to transfer. Table I shows the bits used to encode a vertex’s structural data and geometric data.

A packet loss may refer to one or more vertices loss depending on the packetization strategy. If several vertex split operations are constructed into one packet, we lose more than one split vertices’ information. Moreover, those vertices may be neighbor vertices, so it can cause distortion propagation on the predicted meshes. In this case, we need to predict more than one new vertex in a small one-ring neighbor region. A vertex split based on a predicted vertex could spread errors. In order to prevent burst loss, we introduce a mesh simplification constraint. During compression, we require that at most one edge be collapsed into one vertex. By doing so, at most two vertices may be collapsed into one. Therefore, one split-vertex in mesh \( M_i \) corresponds to only one edge in \( M_{i+1} \).

IV. IMPLEMENTATION

As demonstrated in Figure 1, our system includes some extra modules in addition to the common steps. During compression, mesh simplification, extract split bits and construct streaming packets steps only need to be performed once offline. The separating process is straightforward: by going through each refinement, we generate two files storing split bits, and structural and geometric data respectively. In addition, for a split vertex, we keep an index such as vertex id and refinement id on both split bit and corresponding structural and geometric data. By doing so, it is easy to detect packet loss and retrieve information during decompression.

In our system, we employ reliable channel to stream the base mesh and split bits, meanwhile use unreliable channel to deliver structural and geometric data. The base mesh which stores the basic topology of the 3D mesh should be delivered with the highest priority. The split bits in each refinement comprise the information of where the higher level of details is going to take place. So they should be transmitted reliably. Even though it is not compulsory to maintain the same number of vertices, we need to keep a similar level of details on the decompressed mesh. Both structural and geometric data are transferred over UDP in our method, because their sizes are large and time-consuming to transfer. Table I shows the bits used to encode a vertex’s structural data and geometric data.

On the client side, we add a module to deal with the synchronization, packet ordering and detection of packet loss. Since packets may arrive in different orders in unreliable channel, we use a stream reconstructor, which works as a buffering pipeline, to reassemble them in the correct order before decompression. As mentioned above, we insert vertex id and refinement id on each part, so that the reconstructor can group the packets for the same refinement, and rearrange them in correct orders. Another responsibility of the reconstructor is the packet loss detection. Packets loss happens when a vertex’s split bit valued one which indicates it is a split vertex, but there is no available corresponding structural and geometric data. Depending on whether there is packet loss or not, we apply our prediction method or follow regular decompression procedure. Another function of this module is channels synchronization. While two channels are used simultaneously in the system, the reconstructor has to know the end of the each refinement transmission on both channels in order to start decompressing. The synchronization between two channels can be accomplished by sending a flag message. For example, at the end each refinement, an END message containing the refinement id is sent to indicate the end of the delivery of this refinement. A simple acknowledgement-timeout-retransmission scheme with packet sequence number is used to handle possible loss of END messages in unreliable channels.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bits of Structural Data/Vertex</th>
<th>Bits of Geometric Data/Vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFS</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>CPM</td>
<td>7.2</td>
<td>15.4</td>
</tr>
</tbody>
</table>

Table I. Bits Used to Encode Structural and Geometric Data
In the last stage of the decompression procedure, all predicted new vertices are marked as unreliable [9]. For every vertex, we use one reliability bit, to indicate whether it is from prediction or regularly decoding. In this way, we can track the vertices with predicted connectivity and coordinates to ensure validity of our further decoding, and to prevent the prediction error from propagating to other regions on the mesh.

V. PERFORMANCE EVALUATION

A. Error metrics

Distortion is used to evaluate the decoding performance in our system. We focus on two types of distortion, geometric distortion like difference on vertex positions, and visual difference such as curvature dissimilarity. Two widely used techniques, Hausdorff Distance [13] and Laplacian Operator [14], are utilized to capture these two kinds of distortion separately. We compare the completely decoded 3D mesh with and without packet loss and consequent prediction. Equation (3) gives the Hausdorff Distance (HD) between two meshes \( M \) and \( M' \):

\[
H(M, M') = \max \{ h(M, M'), h(M', M) \},
\]

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

The Laplacian Operator (LO) captures the visual difference such as curvature and smoothness between two meshes. The Visual difference between mesh \( M \) and \( M' \) is given by equation (4):

\[
||M - M'|| = \frac{1}{2n} \left( ||v - v'|| + ||GL(v) - GL(v')|| \right),
\]

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

\[
GL(v_i) = v_i - \frac{\sum_{j \in N(i)} l_{ij} v_j}{\sum_{j \in N(i)} l_{ij}},
\]

where \( l_{ij} \) denotes the geometric distance between vertices \( i \) and \( j \); and \( N(i) \) is the neighbors of vertex \( i \).

B. Simulation results

The proposed loss-tolerance scheme is applicable to different progressive algorithms, and we implement it based on CPM for experiments. Our system is running on a network with packet loss ratio \( l \) between 2%-12%, and we follow random packet loss pattern. In the testing, 35%-40% vertices are added in every refinement. We use the Stanford “Bunny” model (with 34843 vertices and size of 2.96MB) and the Cyberware “Horse” model (with 48485 vertices and size of 4.74MB) to illustrate our results.

The experiments focus on testing the prediction under different data loss ratios. Our results are evaluated using both the geometric distortion and the visual distortion. Figure 3 explains the geometric distortions of the Bunny. Figure 5 demonstrates the results’ visual distortion of horse model. Mesh sequence in Figure 7 shows the Bunny model achieved using proposed method. In Figure 8, we show additional decoding time with respect to the regular decoding time to explain the computational cost.

From Figure 3 to Figure 6, we can see that with the increasing of loss ratio, the distortion increases. Large loss ratio implies more packets lost during transmission, so more new vertices are predicted which leads to higher possibility of distortions. When refinement increases, distortion also raises accordingly. As shown in those figures, the distortion caused by our prediction is only a very small percent of the whole model. Therefore, we conclude that the distortion is dominated by the simplification error not by the prediction error. Noticing that in both Figure 3 and Figure 4, distortions do not change linearly, we believe it is related with vertices’ importance. As mentioned in [5, 7], every split vertex has an importance value and there are dependencies between split vertices. Once random loss occurs on split vertices with high importance, results have more distortion.

The decoding time increases when packet loss ratio is larger, because high loss ratio implies that we have more vertices which need to be predicted. Figure 8 proves that the proposed scheme is very efficient, and the decoding time can be further reduced if the client has more powerful computing hardware.

![Fig. 3. Geometric distortion (HD) between the zero-loss decoded and the proposed decoded Bunny over different loss ratios l.](image)

C. Comparison with priors works

As mentioned previously, the easiest way of handing packet loss is simply abort the vertex split and
all the subsequent vertex split corresponding the lost data. We call this method *Loss-without-prediction* (LwoP). We compare the LwoP method and proposed prediction method and show the results in Figure 9 and Figure 10. Experiments show that around 70% of our prediction results are better than the LwoP method. 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 $M_{\text{max}}$ can have parts in different level of details. 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, the decoded meshes have more distortions using LwoP. Moreover, if the base mesh is in a very low resolution, which means we need more refinements data to decompress base mesh into original mesh, the results using LwoP are not satisfying.

**Fig. 6.** Visual distortion ($LO$) between the zero-loss decode and proposed decoded Horse over different loss ratios $l$.

We compare our approach with [11]. Geometric distortion comparison results are depicted in Figure 11. Obviously distortions of the decoded meshes using proposed method are smaller than [11]. In comparison to their method, we have few advantages. Since split bits are transmitted reliably in our algorithm but not in theirs, we know better 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 can be wrong to always select the closest triangle to insert new vertex. We try to compare the efficiency of our approach with [7], but their decoding time includes data retransmission. Our work focuses more on the prediction scheme and decoding time does not have retransmission delay. Thus, it is not appropriate to perform direct comparison.

**Fig. 8.** Increased decoding time of the Bunny and the Horse using the proposed scheme on Pentium IV machines.

**Fig. 9.** Geometric Distortion ($HD$) between decoded Bunny with zero-loss and with loss ratio 0.12.

**VI. CONCLUSION AND FUTURE WORKS**

In this paper, we propose a receiver-based loss-tolerance scheme to deal with the packet loss. We utilize reliable channel to transmit the base mesh and the split bits, meanwhile employ lossy channel to deliver both structural and geometric data. During decompression, a stream reconstructor is used to synchronize two channels, reorder packets, and detect packet losses. Whenever there is packet loss, our prediction method is applied to reconstruct the new vertices’ connectivity and estimate their coordinates. 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. Our method does not need any additional data from server to recover packet loss, and it is not necessary to retransmit lost data. The proposed prediction method
provides good decompression quality without incurring high computational cost.

In the future, we would like to integrate the partition methods [9] to further reduce the distortions in our results. Moreover, the importance of vertices will be included to further improve our results.

![Fig. 10. The decoded Bunny model using Loss-without-prediction and the proposed method over loss ratios l=12%.
](image)

![Fig. 11. Geometric Distortion (HD) decoded Bunny model with loss ratio 0.12.
](image)

VII. REFERENCES


![Fig. 7. Sequence of meshes for the Bunny model over loss ratio](image)