Extending Progressive Meshes for Use over Unreliable Networks

Zhihua Chen
Dept. of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
zhihua.chen@vanderbilt.edu

Bobby Bodenheimer
Dept. of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
bobbyb@vuse.vanderbilt.edu

J. Fritz Barnes
Dept. of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
J.Fritz.Barnes@vanderbilt.edu

ABSTRACT

Progressive meshes [7] provide an attractive mechanism for transmitting 3D geometry over networks. Progressive meshes (PM) transmit a coarse initial mesh and refinements that can be applied to the initial mesh. However, these techniques assume a reliable network protocol such as TCP/IP is used for data transmission. When transmitting 3D geometry for graphical applications such as concurrent virtual environments distributed over wide area networks with some nodes potentially using wireless networks, many losses will occur. TCP/IP performance degrades in the presence of packet loss and multicast communication mechanisms typically do not provide reliable communication. These applications motivate the issue of transmitting geometric data over unreliable networks. In this paper, we discuss several errors that are caused when reconstructing PM geometries after some packets have been lost. We modify the PM data structures to improve robustness during packet loss. We use these modifications to improve a hybrid transmission technique that uses TCP to transmit the base mesh and portion of the initial mesh and then use UDP to transmit the remainder of the mesh to improve transmission performance.

1. INTRODUCTION

As the Internet expands, demand is growing for high quality 3D geometry in applications, from games such as Everquest [5] to collaborative virtual environments to multimedia and purely web-based applications. Such applications use high resolution 3D meshes to achieve their effect, and the challenge of the growing demand for these meshes is how to store and transmit the large amount of data contained in them.

Hoppe [7] proposed the progressive mesh (PM) technique as an initial solution for the transmission of geometric data over data networks. In this method, the server initially sends coarse shape information to a client that can be reconstructed and rendered very quickly. Then increasing detail in the model is transmitted to the client, allowing the client to progressively refine the initial model into the full resolution model.

Various authors, e.g., [10] and others, have combined mesh compression techniques with progressive transmission to reduce the amount of data that needs to be sent. Typically, there is a trade-off between compression ratio and the *fragility* of the compressed mesh. In prior work, we demonstrated that using an unreliable channel can improve transmission times by as much as forty percent with a negligible drop in visual quality [4]. In this work, we investigate mechanisms that extend the PM technique to handle random losses occurring in the transmission channel. We describe the flaws that occur when packets are lost using the PM technique and modifications and heuristics that can be used to improve the visual quality of models when some packets have been lost. These modifications allow additional improvements in the transmission time over unreliable networks.

The paper is organized in the following manner. In Section 2 we place our work in context of what has been done in this area. Section 3 details the modifications to the basic PM scheme needed when data loss can occur. Section 4 discusses our hybrid transmission protocol. Section 5 presents the results of actual transmission experiments over a lossy network with an analysis of them. Finally, in Section 6 we discuss our results.

2. OUR WORK IN CONTEXT

The PM scheme was devised by Hoppe in [7, 9], and we have implemented the basic versions without the modifications of later work, e.g., [8]. The reader is referred to thos papers for a detailed discussion; here we simply present enough information to familiarize the reader with our notation. The PM representation of a mesh M is stored as a coarse mesh M^0 and a sequence of n detail records called *vertex splits*. These vertex splits indicate how to incrementally refine M^0 so that after the n vertex splits have been processed, the original mesh M is recovered. In fact, the PM representation defines a sequence of meshes $M^0, M^1, \ldots, M^n$ which provide increasingly accurate approximations of M.

A vertex split is a basic transformation that adds a vertex to the mesh. The basic progressive mesh scheme is implemented, but for the purposes of this paper, a vertex split does not contain normal, texture, or material information. Each vertex split is a 30 byte quantity consisting of a face index, $flclw$, an index vs_index ($0 \leq vs\_index \leq 2$), an encoding vlr_rot, and two vertex position deltas, vad_l and vad_s. In our experiments, these vertex splits are packed into 1400 byte packets. Thus, each packet contains roughly 46 vertex splits.

Figure 1 illustrates a PM vertex split transformation. Each vertex split operation introduces a new vertex v_t and two new faces, as shown. The location of a vertex split is parameterized by v_s, v_l, and v_r. By default, the PM data structure does not include incidence information in the vertex to face direction. The vertex values are determined through the fields $flclw$, vs_index, and vlr_rot. To determine the vertex being split (v_s), the three vertices of the face $flclw$ are sorted by their index values and stored into an ordered list. They are indexed by vs_index. Vertex v_l is the next vertex clockwise on the face $flclw$. The vertex v_r is determined by vlr_rot, the number of clockwise rotations about v_s from v_l to v_r.

Other techniques for mesh decomposition with an idea towards

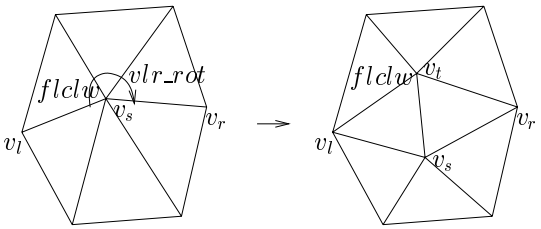

Figure 1: Vertex split transformation.

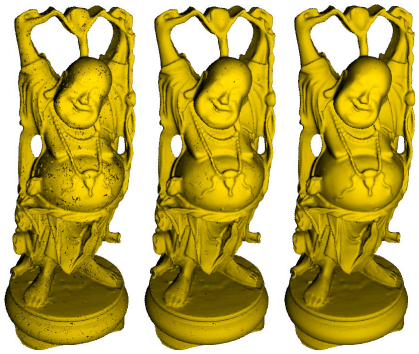

Figure 2: The Buddha model after transmission over a lossy network: (a) no change to PM data structures; (b) using absolute vs_index; (c) using both absolute vs_index and altered vlr_rot field.

robust transmission exist, e.g. [3, 10, 1]. Most of these techniques have not been tested in as rigorous an environment as our methods. In the context of progressive transmission, much of the work has focused on better methods of transmission, e.g., [10, 2, 6]. These techniques are complementary to our own, in that they could fit within the techniques proposed here to achieve higher transmission rates.

3. MODIFICATIONS TO THE PM DATA STRUCTURE

When the transmission channel is lossy, vertex splits will be

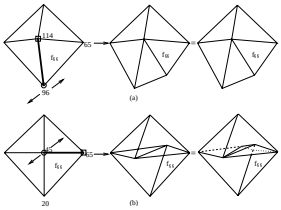

Figure 3: Illustration of self-intersection occurrence caused by vs_index: (a) No packet loss, intermediate vertex splits change the three vertices of face f_{66} to 65, 96 and 114; (b) Packets containing the intermediate vertex splits are lost, the three vertices of face f_{66} remain unchanged as 65, 20, 45.

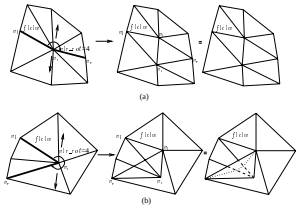

Figure 4: Illustration of self-intersection occurrence caused by vlr_rot: (a) No packet loss, intermediate vertex splits change the number of faces around v_s to 8; (b) Packets containing the intermediate vertex splits are lost, the number of faces around v_s remain unchanged at 6.

lost when packets are lost. As a result, the geometry of the reconstructed model can be corrupted. This corruption most often takes the form of surface self-intersection, i.e., parts of the surface intersecting with the surface itself. The self-intersections destroy the manifold property that the surfaces in the original model have and create visual artifacts in the reconstructed models, as shown in Figure 2 (a). In this section we discuss how lost vertex splits cause self-intersections to occur and describe modifications to the PM data structure that improve the situation.

First, in addition to a packet containing only vertex split information, each packet contains a 4 byte header that stores the index number of the first face of the mesh that will be introduced by the first vertex split in this packet. The client renderer uses this number as a "poor man's" error correction, to give faces generated by vertex splits after lost packets their index number in the full resolution mesh. This process is done so that future splits whose face index, $flclw$, reference these faces will be able to find them. This header is a monotonically increasing number, and can also be used to detect out-of-order packets.

One cause of self-intersection is that the vs_index field in a vertex split record is the $relative$ numerical order of the index value of the split vertex v_s among the three vertices on $flclw$ (see Figure 1). When vertex splits are lost, the index values of the vertices on $flclw$ may become different from what they were when the progressive mesh was generated. Thus, the ordering of the vertices may be wrong. This change of ordering causes the PM reconstruction to find the wrong v_s, which results in geometric corruption. For example, in Figure 3 (a), without packet loss, intermediate vertex splits change the three vertices of face f_{66} to 65, 96 and 114. A subsequent vertex split with $flclw$=66 and vs_index=1 splits vertex 96 because its index is the second smallest. When the intermediate splits are lost, as shown in Figure 3 (b), the three vertices of face f_{66} remain unchanged as 65, 20, 45. Now vertex 45 has the second smallest index among the vertices of f_{66}. The same split with $flclw$=66 and vs_index=1 then splits vertex 45 and causes the faces connected with the added vertex to pierce the top right face.

The PM data does not store how the three vertices of a face to be reconstructed were ordered in the vertex list of the face in the original model. Therefore, the ordering of vertices in the vertex list of a face in the reconstructed model may be different from that in the original model. This fact is part of the reason why the vs_index was defined as a $relative$ order of index value rather than the $absolute$ placement of v_s in the vertex list of $flclw$. To alleviate this problem, we add a second pass over the PM data to replace each $relative$ vs_index with the $absolute$ placement of v_s at the time of the vertex split. The time the second pass takes

is negligible compared with the time to generate the progressive meshes. When this change is incorporated a significant number of self-intersections can be eliminated, as shown in Figure 2 (b).

The vlr_rot field in a vertex split record can also cause self-intersections to occur. The vlr_rot field, as originally defined, is the number of clockwise rotations from v_l to v_r about v_s. When vertex splits are lost, the number of rotations from v_l to v_r may be different from what it was in its original context, or v_r may not exist at all. This difference can cause the reconstruction program to pick the wrong v_r, resulting in geometric corruption as shown in Figure 4. To help the reconstruction program detect this case without changing the size of the PM data structure, we modify the vlr_rot field. Instead of using 16 bits for the vlr_rot field, we use 8 bits, and use the remaining 8 bits to encode a quantity called $cycle\_size$. The $cycle\_size$ field stores the total number of faces around v_s. The reconstruction program checks the $cycle\_size$ at run time and discards splits whose $cycle\_size$ do not match those in the partially reconstructed meshes. When this change is incorporated, additional self-intersections can be eliminated, as shown in Figure 2 (c).

4. HYBRID TRANSMISSION

The progressive mesh format creates an alternative representation of the 3D geometry. One significant advantage of this representation is that some packets containing mesh data are more important than other packets. In particular, initial splits provide significant improvements as measured by the Hausdorff metric, while later splits provide decreasing benefits.

Our hybrid technique leverages the inherent differences that vertex splits have on the resulting visual accuracy of the reconstructed mesh. Thus, in this scheme, we begin by transmitting data using the TCP protocol. Part-way through the transmission, the hybrid sender closes the TCP connection and transmits the remaining data using UDP. Details on these protocols can be found in [4]. This technique allows the sender to reliably transfer the base mesh and some of the initial splits and use a more aggressive technique to transfer less important splits.

TCP controls the rate that packets are sent to maintain flow control and minimize congestion. As a result, TCP does not provide the end-user control of the sending rate. When we consider using UDP for transmission of data, the end-user has significant influence over the send rate. Therefore, our application must carefully select the send rate. If the send rate exceeds the capacity of the channel, we will increase the number of packets lost. However, if we decrease the send rate, we will not lose any packets but the transfer of information will take much longer than a TCP connection might take. We experimentally determine UDP send rates to use in applications. In future work, we will investigate automatic mechanisms for setting the UDP send rate.

An issue of concern in implementing this idea is that the sender can begin sending UDP data to the receiver while the receiver is still reading data from the TCP connection. This behavior causes a problem since the initial data will fill up the operating system's buffer for incoming network data and start dropping UDP packets that arrive. Therefore, the hybrid protocol adds a delay before sending the UDP data until the receiver has received all TCP data.

5. EXPERIMENTAL RESULTS

We tested our transmission methods by running a set of experiments with varying levels of background load using a variety of models. The experimental setup is identical to that described in [4]; for reasons of space, the reader is referred there for details. Also for reasons of space, we report results for only one model, the "happy Buddha" model. The Buddha model contains 1.08M faces in its full model and 1998 faces in its base mesh. The suite of experiments was run using different send rates for the UDP portion of the model transmission.

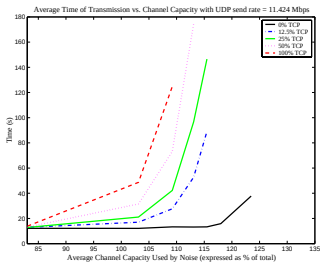

Figure 5: The average time of transmission of the Buddha model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 11.424 Mbps.

5.1 Results for Buddha Model

Our extensions to the PM data structure cause additional splits to be rejected because they cause geometric corruption, as discussed earlier. The first modification generally does not cause much impact on the total number of faces reconstructed. For example, over ten trials using the Buddha model, transmitting 25% of the data with TCP in addition to the base mesh causes less than 1% of the splits to be rejected (893k faces reconstructed on average versus 889k faces with the absolute index modification in place). The second modification, changing the vlr_rot field, is more severe, and on average causes 24% of the splits to be rejected over the method without modification (679k faces reconstructed on average with both modifications in place). Nonetheless, we believe the improvement in visual quality justifies the changes.

Figure 5 shows the average transmission time for the various transmission schemes, and Figure 6 shows the average number of faces received versus channel capacity. In these figures, the UDP send rate was 11.424 Mbps, and TCP was used to transmit the base mesh plus 0 to 50% of the rest of the model. The important result in Figure 5 is that the transmission time to send the model using pure TCP is the longest. Moreover, as the ratio of packets sent by TCP decreases, the transmission time improves for all given levels of noise. To explain this behavior, one must consider what happens when we increase the amount of noise. When the noise exceeds the channel capacity, the channel is full, and buffers in the network become saturated. This saturation results in packet loss. Packet loss causes TCP to incur timeouts or apply the fast retransmission scheme. Thus, the transmission time using TCP increases. As a final note, the 0% TCP plot in this figure is not quite constant because, as mentioned, the base mesh is transmitted reliably using TCP.

This transmission time improvement comes at a cost, however. Figure 6 shows the number of faces in the received model for each of these transmission methods. Note that TCP, since it is a reliable transport protocol, always transmits all the faces. When a portion of the packets are sent using UDP, significant packet loss can occur, and this loss becomes worse as the noise increases. Therefore the selection of an appropriate hybrid protocol will depend on the tradeoff between the transmission time and the visual degradation that occurs when packets are lost. The visual degradation of this process is shown in the Buddha model in Figure 7. This fig-

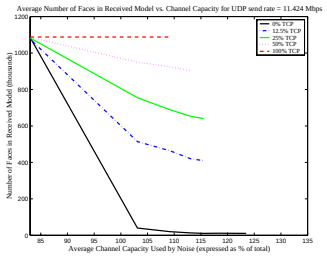

Figure 6: The number of faces received of the Buddha model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 11.424 Mbps.

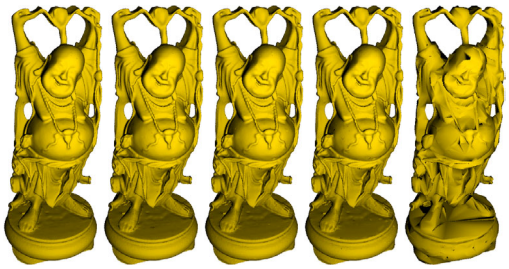

Figure 7: The Buddha model after transmission over a lossy network: (a) the full model as sent by TCP; (b) model received when 50% transmitted by TCP; (c) model when 25% sent by TCP; (d) model when 12.5% sent by TCP; (e) model when fully sent via UDP.

ure shows the models that were received in one experiment using a UDP send rate of 11.424 Mbps and where the noise consumed 109% of the channel capacity. In particular, the visual quality of the model is excellent when 12.5% of it is transmitted via TCP. TCP took on average 131 seconds to transmit at this noise level, while the average transmission time for the 12.5% scheme was 27 seconds, a considerable savings.

A significant concern in any transmission scheme is that the bandwidth of the scheme is minimal. In Figure 8, we compare the number of bytes used to transmit the Buddha model by both TCP and UDP as we increase the amount of background noise. UDP is constant by design—it is equivalent to the TCP 0% line in the previous figures. Note that TCP uses more bandwidth as the noise increases, from 1.3 to 4.6%. This result is unsurprising since TCP must resend lost packets, requires the use of acknowledgments, and has a larger header size than UDP packets. Note that the comparison in Figure 8 does *not* include the bandwidth of packets used to acknowledge receipt of data by the PM receiver.

6. CONCLUSION

In this paper, we have modified the progressive mesh data structure to enable it to more robustly transmit 3D geometric data over lossy channels. We demonstrated this modification in the context of a hybrid transmission scheme, and showed that the hybrid approach improves the transmission performance. This improvement in performance comes at the cost of lost packets. The progressive mesh representation allows us to minimize the visual impact when packets are lost, although some surface corruption is visible. The visual results indicate that geometric corruption has minimal impact on the visual quality of the mesh.

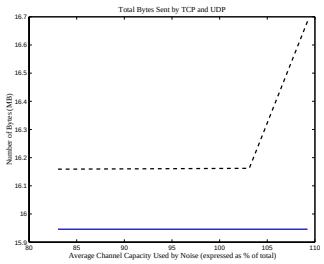

Figure 8: Total bytes sent by TCP (dashed line) and UDP (solid line) versus noise in the channel. Note that the UDP send rate is constant, but the number of bytes sent by TCP increases as it must resend data due to lost packets.

We are currently investigating methods of representing and reconstructing the progressive mesh that eliminate this corruption with less overhead than employing a forward error correction (FEC) code arbitrarily on the data.

REFERENCES

[1] AL-REGIB, G., AND ALTUNBASAK, Y. An unequal error protection method for packet loss resilient 3-d mesh transmission. In *Proceedings of INFOCOM 2002* (2002).

[2] ALLIEZ, P., AND DESBRUN, M. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of ACM SIGGRAPH 2001* (July 2001), Computer Graphics Proceedings, Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 195–202.

[3] BISCHOFF, S., AND KOBBELT, L. Streaming 3d geometry over lossy communication channels. In *Proceedings of the IEEE International Conference on Multimedia and Expo* (2002).

[4] CHEN, Z., BODENHEIMER, B., AND BARNES, J. F. Robust transmission of 3d geometry over lossy networks. In *Web3D 2003 Symposium Proceedings* (St. Malo, France, Mar. 2003), ACM SIGGRAPH, pp. 161–172.

[5] EVERQUEST. http://everquest.station.sony.com.

[6] GANDOIN, P.-M., AND DEVILLERS, O. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics 21*, 3 (July 2002), 372–379. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[7] HOPPE, H. Progressive meshes. In *Proceedings of SIGGRAPH 96* (New Orleans, Louisiana, August 1996), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 99–108. ISBN 0-201-94800-1.

[8] HOPPE, H. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH 97* (Los Angeles, California, August 1997), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, p. I198. ISBN 0-89791-896-7.

[9] HOPPE, H. Efficient implementation of progressive meshes. *Computers & Graphics 22*, 1 (February 1998), 27–36. ISSN 0097-8493.

[10] PAJAROLA, R., AND ROSSIGNAC, J. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (January - March 2000), 79–93. ISSN 1077-2626.