An Analytical Model for Progressive Mesh Streaming

Wei Cheng Wei Tsang Ooi
Department of Computer Science, National University of Singapore
{chengwe2, ooiwt}@comp.nus.edu.sg

Sebastien Mondet Romulus Grigoras Géraldine Morin
IRIT, University of Toulouse, France
{sebastien.mondet, romulus.grigoras, geraldine.morin}@enseeiht.fr

ABSTRACT

3D triangular mesh is becoming an increasingly important data type for networked applications such as digital museums, online games, and virtual worlds. In these applications, a multi-resolution representation is typically desired for streaming large 3D meshes, allowing for incremental rendering at the viewers while data is still being transmitted. Such progressive coding, however, introduces dependencies between data. This paper quantitatively analyzes the effects of such dependency on the intermediate decoded mesh quality when the progressive mesh is transmitted over a lossy network, by modeling the distribution of decoding time as a function of mesh properties and network parameters. To illustrate the usefulness of our analytical model, we describe three of its applications. First, we show how it can be used to analytically compute the expected decoded mesh quality. Second, we study two extreme cases of dependency in progressive mesh and show that the effect of dependencies on decoded mesh quality diminishes with time. Finally, based on the model, we propose a packetization strategy that improves the decoded mesh quality during the initial stage of streaming.

Categories and Subject Descriptors

I.3.2a [**Graphics Systems**]: Distributed/Network Graphics; C.2.4b [**Distributed Systems**]: Distributed Applications

General Terms

Performance, Design

Keywords

3D data, streaming, progressive meshes, packetization

1. INTRODUCTION

Advances in 3D scanning technology and mesh reconstruction algorithms have lowered the barrier in creating complex mesh objects and sharing them over the network. For instance, the Stanford's Digital Michelangelo Project [11] digitized statues made by Michelangelo and provided a software, ScanView [10], to allow users to remotely view the 3D version of the statues. Second Life, an online virtual world, allows sharing of user-created 3D objects. In terms of ratio, more users are generating content in Second Life than on the Web [13]. While Second Life only supports constructive solid geometry objects at the moment, there is a user push to import complex, mesh-based objects into Second Life. These signs suggest that an increasing amount of 3D mesh data will be available for remote viewing over the Internet.

The amount of data constituting a high quality 3D mesh can be huge. For example, the statue of David, from the Digital Michelangelo Project, consists of 2 billion polygons. After lossless compression, the total data size is 32 GB. To reduce waiting time when downloading such a huge amount of data, a common technique for remote viewing is to encode a 3D mesh progressively [12], allowing a low resolution version of the mesh to be transmitted and rendered with lower latency. The refinement information is continuously being transmitted, and the quality of the rendered model is incrementally improved over time. Such progressive rendering is also useful in the case of virtual worlds such as Second Life. In such applications, the models are typically smaller and simpler, but due to the existence of multiple models and the interactivity requirements, it is desirable for users to quickly visualize a coarse version of the scene first, instead of waiting for full objects to be downloaded before they can be displayed.

Progressive coding of meshes, much like progressive coding of video, introduces dependencies between data. Dependencies between data units can cause delay in decoding when sent over a lossy network – a data unit cannot be decoded and displayed if one of the other data units it depends on is not received correctly. For example, in the context of video streaming, MPEG-encoded frames are inter-dependent: an I-frame has to be received and decoded properly before being able to decode the subsequent P-frames and B-frames referencing this I-frame (either directly or indirectly). Another example is in layered coded video, where the base layer has to be received before the enhancement layers can be decoded. The effects of dependencies in the context of video streaming have been well studied in the literature. 3D multi-resolution objects also have dependencies between different level of details. For progressive meshes, the mesh is refined by successive *vertex splits* (see Figure 1). Thus, dependencies exist between the original vertex and its one ring (the direct neighbors of the vertex) and the vertices and triangles created by the vertex split. The effects of these dependencies on decoded mesh quality, however, are not well understood. Due to the fundamentally different nature of progressive mesh and video data, what we learn from video streaming research does not apply. This paper aims to study the effect of dependencies in progressive mesh streaming by proposing an analytical model, relating

MM'07, September 23–28, 2007, Augsburg, Bavaria, Germany.

the decoded mesh quality of progressive mesh to packet loss rate, given the dependencies among vertex splits.

The decoded mesh quality is determined by the decoding time of the vertex splits and their contribution to the overall mesh quality. To estimate the decoded mesh quality, our analytical model predicts the decoding time of a vertex split. We first express the expected time for receiving each packet in terms of loss rate, round trip time, and sending rate. A received vertex split has to wait for the vertex splits it depends on to arrive before it can be decoded. Therefore the received time of a vertex split is not always equal to its decoding time. Our model gives an expression for the expected decoding time given the dependencies among the vertex splits. To measure the decoded quality at a given time, we propose a metric based on the contribution of decoded vertex splits to the decoded mesh quality. This decoded mesh quality can be computed once we know the expected decoding time of a vertex split and its contribution.

Our analytical model is useful in several ways. First, we can analytically evaluate different strategies for streaming a progressive mesh. For instance, packetization of vertex splits affects the intermediate decoded mesh quality at the receiver. Evaluating different packetization scheme using simulation is an option, but it may take many experiments to obtain accurate expected values. Our model computes the expected value easily. Second, our model can also help in developing a better sending strategy. The quality of the decoded mesh depends on various factors, which include not only network conditions such as loss rate and round trip time, but also the order, the dependency, and the importance of the data. The last three factors are in turn affected by the packetization strategy. Our model can estimate the effect of each factor on the decoded mesh quality. As such, we can make the right trade-off in packetization during transmission to improve the quality. Finally, we derive closed-form expressions in two extreme cases, giving us insights to the importance of dependencies on the decoded mesh quality.

The rest of the paper is organized as follows. In Section 2, we first describe progressive mesh and how it introduces dependencies. Section 3 presents related work in the area of 3D objects streaming. Section 4 introduces our evaluation metric for intermediate decoded mesh quality. Section 5 introduces our analytical model. We then describe three applications of our model. In Section 6, we show how our model can be used to analytically compute the expected decoded mesh quality. In Section 7, we study two hypothetical extreme cases, and showed how the effect of dependencies on decoded mesh quality. Section 8 describes how we can use our model to devise a packetization strategy for transmitting progressive mesh. Section 9 concludes by reflecting on the insights we gain from our model and its implications.

2. PROGRESSIVE MESH

Progressive transmission and rendering of 3D objects requires multi-resolution representations of data. Progressive mesh is a multi-resolution technique proposed by Hoppe [9] in 1996 to enable progressive rendering of 3D meshes. The technique is based on an operation called *edge collapse*, and its reverse operation, *vertex split*. Given a (non-progressive) 3D mesh, the technique applies a series of edge collapses, simplifying the model by reducing the number of vertices and faces. The final, simplified model obtained after this process is called the *base model*. Given a base model, we can reconstruct the original model by reversing the edge collapse operations through *vertex splits*, incrementally adding new vertices and faces. So, a progressive mesh can be represented by the base model and a series of vertex splits. There are dependencies between vertex splits and the base model, as well as among the vertex splits. A vertex split operation might need a vertex or a face created by another vertex split as input. Figure 1 illustrates edge collapse and vertex split operations.

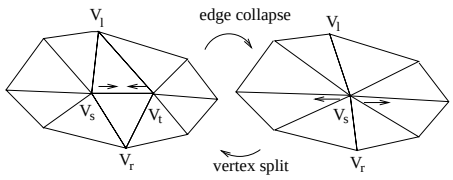

Figure 1: Edge collapse and its reverse vertex split

Progressive meshes are well adapted for streaming since they offer a fine level of progressivity. Arbitrary multi-resolution models provide different levels of detail for the same object. Whereas most multi-resolution models are organized level by level, progressive meshes have the advantage of allowing the refinement steps to be done vertex split by vertex split. This progressivity is crucial for our application since a streamed vertex split will only have to wait for the vertex splits it depends on, and not for other refinement operations of the same level (like e.g. for subdivision surfaces). Therefore, the only dependencies we need to study are the dependencies between vertex splits.

Many extensions to Hoppe's progressive mesh have been proposed (e.g., [16, 14, 6, 4, 3]). The main idea behind these extensions is to combine multiple operations into one, thereby further reducing the redundancy and improving the efficiency. While this paper only considers the original progressive mesh, our model is general enough to model many of these extensions.

Significant differences exist between progressive mesh streaming and video streaming. First, in video steaming, every packet should be received in time, or it will not be played back. Hence, generally sending new data is more important than re-sending old data. On the contrary, in streaming of a progressive mesh, old data is usually more important than new one, since the reconstruction begins with the most significant refinement. Given this observation, retransmission of a lost packet is not only useful, but it should also take priority over transmission of new packets.

Second, in progressive mesh, there is no strict order of vertex splits rendering order, unlike in video where frames must be displayed in sequence. A received vertex split can be rendered immediately as long as all the vertex splits it depends on have been received. When a packet is lost, the subsequent received packets may still be decodable. Those vertex splits received afterwards that depend on the lost packet, however, have to wait until the lost packet is retransmitted successfully before they can be displayed. This observation hinted that we should reduce the dependencies between packets as much as possible, since dependencies cause delay in rendering details generated by vertex splits.

Third, a video frame is usually larger than a packet, causing the streaming application to split the video frame into several packets. On the other hand, in progressive mesh, a vertex split is small (in the order of 10 - 15 bytes). Thus, before the application transmits the mesh, it needs to group vertex splits into packets. This *packetization* process affects the dependencies among the packets, which in turn affects the delay in rendering if some packets are lost.

3. RELATED WORK

Given the background in progressive mesh, we now describe related research in transmissions of progressive mesh over the network. There are three main classes of work in existing literature – error resilient compression, error control, and packetization.

Existing work in robust mesh compression aims to reduce dependencies among the mesh [15, 17]. Similar to introducing key frames or restart marker in video/image coding, mesh segmentation is used to reduce the affected range of one packet loss. In robust mesh compression, a mesh is typically divided into several independent parts and then coded separately. Therefore the effect of one packet loss is confined to the part to which it belongs. The finer the partition is, the fewer the affected vertices are. The coding efficiency, however, will decrease due to increase in redundancy and decrease in correlations.

Al-Regib and Altunbasak [1] proposed an unequal error protection method to improve the resilience of progressive 3D mesh based on CPM algorithm. Forward error correction (FEC) codes are added to different levels of mesh data (base mesh and different levels of detail) such that the decoded mesh quality is maximized. The method is similar to FEC protection of video data. As argued in the previous section, we believe that for streaming of 3D meshes, retransmission is always a better choice (except in cases such as multicast where retransmission is not scalable). Chen et al. [5] also applied FEC to streaming progressive meshes. They analyzed several transmission schemes TCP only, UDP only, TCP with UDP, and UDP with FEC and studied their effects experimentally on transmission time and decoded mesh quality. Al-Regib et al. proposed an application layer protocol, 3TP, for streaming of 3D models [2], combining both TCP and UDP. In 3TP, packets that are of importance are sent using TCP, while the rest are sent with UDP to minimize delay.

Gu and Ooi [7] were the first to look at the packetization problem for progressive meshes. They model the packetization problem as a graph problem where the objective is to equally partition the graph into k partitions with minimum cut size. The problem is shown to be NP-complete and a heuristic is proposed. They, however, assume that every vertex split is equally important. In practice, the importance of vertex splits can vary considerably. A similar work in packetization is done by Harris III and Karvets [8]. They proposed a protocol named On-Demand Graphic Transport Protocol (OGP) for transmitting 3D models represented as a tree of bounding volumes. A key component of the protocol is to decide which bounding volumes to send. OGP begins with packing the largest possible subtree at the root and continues to pack the nodes in the subtree of acknowledged nodes in breadth-first order. This approach again aims to reduce the dependencies, and is similar to the *b-sub* approach. Both these approaches are compared to our method in Section 8.1.

These existing studies are mainly concerned with dependencies (packetization and mesh segmentation) and importance of mesh data (unequal error protection and use of reliable protocol), two factors that affect the quality of decoded meshes. None, however, have looked at both factors and characterize their effect on quality. We aim at achieving this by proposing an analytical model.

4. EVALUATION METRIC

Before we present our model, we first explain how we measure the quality of a progressive mesh at a given time t.

The quality of a simplified mesh represents how close it is compared to the original mesh. Some objective metrics have been proposed. Many of them are based on the Hausdorff distance between a set of sample points on the original mesh and the corresponding ones on the simplified one.

In the case of a progressive mesh, the base mesh has the lowest quality, and each vertex split increases the quality by a certain amount. We define the *importance* of a vertex split as the amount of increase in decoded mesh quality caused by this vertex split. This value can be determined by comparing the quality of the model before and after the vertex split operation. Strictly speaking, the importance value may depend on the order, but, for simplicity, we assume that a refinement always improves the quality of the model by a value, independent of other refinements. Then, the quality of a received model at time t (or, *intermediate quality*) can be represented as the summation of importance of all vertex splits decodable at t. Since the simplification process typically prefer collapsing an edge with low quality loss, edges with lower importance are collapsed earlier during the simplification and split later during the reconstruction. Therefore, in a progressive mesh, vertex splits operations are typically performed in decreasing order of importance.

This model of decoded mesh quality is general – one can define different importance metric for a vertex split depending on the objective metric. For instance, a view-dependent metric can set the importance of a vertex split to zero if the vertex split is outside of the user's viewpoint.

We evaluate and compare different streaming strategies of the same progressive mesh by examining the intermediate quality. Note that as long as the sending rate is the same, the complete mesh will be received at the same time. We are more interested in intermediate quality, since in progressive transmission, we want the users to view a mesh with the best possible quality before the whole mesh is transmitted. This intermediate quality is important especially in interactive applications, where what a user sees initially matters. One simple metric is to compare the intermediate quality at a given time t; the other is to compare the time it takes for two streaming strategies to reach an intermediate quality. These simple metrics are nonetheless too restrictive. For instance, in Figure 2(a), although the quality at time t is the same for both strategies, we think that Strategy 2 is better since it can achieve a higher quality earlier.

Based on this observation, we propose an evaluation metric that measures the intermediate quality over a period of time (rather than instantaneous). The metric sums up the intermediate quality of the mesh between a given time period. Imagine plotting the quality of the received mesh versus time, as in Figure 2. This metric is equivalent to the area between the curve and the time axis. There are two ways we can interpret the meaning of this metric. We view the metric as the sum of decoded quality from time 0 to t. Discretizing the time, we let q_t be the decoded quality of the progressive mesh at time t, and let a_t be the area under the curve at time t. Then, $a_t = \sum_{i=0}^{t} q_i$. Here, the area under the curve is computed as the area sum of vertical slices. We can also compute the area as the area sum of horizontal slices (see Figure 2). Thus, to compute the area, we can sum up the product of a vertex split's importance and the amount of time since it was decoded. Let the importance of a vertex split i be w_i and the time when it was decoded be D_i. Then

$$a_t = \sum_{i \in K_t} w_i (t - D_i), \tag{1}$$

where K_t is the set of vertex splits that have been decoded at time t. To compute the expected rendering quality using Equation 1, it is crucial to compute the expected decoding time of a given vertex split. We show how we model this value next.

5. ANALYTICAL MODEL

We now develop an analytical model for transmission of progressive mesh over lossy networks. We first need to model the progressive mesh and the network. Since typically the base model is small (less than 1% of the total size) and is transmitted using reliable transport protocol [2], we assume that the base model has been received by the client. We will focus on modeling the vertex splits,

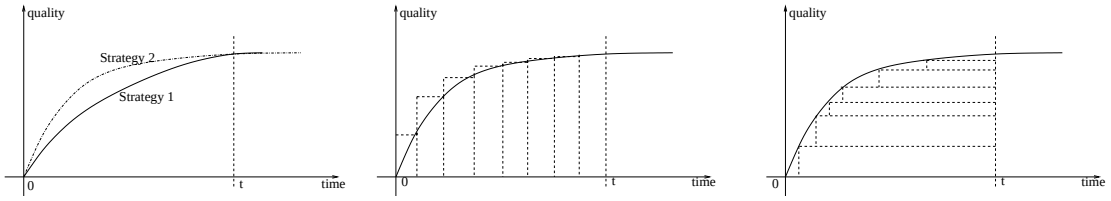

Figure 2: Intermediate quality of decoded mesh. From left to right: (a) Strategy 2 is better than Strategy 1 since the area under the curve is larger. (b) Area sum of vertical slices. (c) Area sum of horizontal slices.

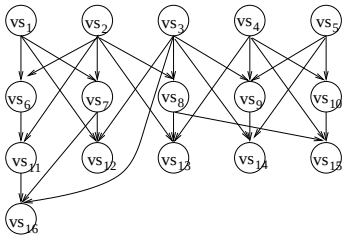

Figure 3: Dependency between vertex splits represented as a DAG.

which can be modeled as a directed acyclic graph (DAG) (see Figure 3). In the DAG, nodes represent the vertex split operations, and edges represent the dependency between these operations. We assign each node a weight, which corresponds to the importance of the vertex split. We will use the terms node and vertex split interchangeably in this paper.

On the server, the vertex splits are grouped into packets. We assume that each packet contains the same number of vertex splits. We say packet P is a *parent packet* of packet Q if Q contains a vertex split that depends on a vertex split in P. Therefore, a vertex split can be decoded after the packet that contains it and all its parent packets are received.

We denote B and p as the average sending rate[1] and packet loss rate respectively. We assume a retransmission-based protocol – when packet loss is detected at the receiver, a retransmission request (NACK) is sent back to the sender, triggering a retransmission. Let T_d be the average period between the time a packet P is sent and the time P's NACK is received (P's loss is detected) at the sender – T_d is the round trip time plus some constant.

We discretize the time into slots. Each slot is the time to send one packet. Thus, one time unit is L/B seconds, where L is the packet size. The time T_d is normalized to this time unit. Thus, T_d can also be interpreted as the number of packets transmitted between sending a packet P and detecting loss of P. Moreover, at the server, we define the time the first packet is sent as time 0, whereas, at the client, we define the time the first packet is received (if it is not lost) as 0. Therefore, a packet sent at time t will be received at time t if it is not lost. We use a different timeline at the receiver to avoid an additional term, $RTT/2$, in equations related to the received time and decoded time. We summarize these notations as well as other major notations we use in this paper in Table 1.

With that background, we will now explain how we obtain the expected value of sending time, received time, and decoding time of a vertex split. Due to space constraints, we will only sketch the proofs in this paper.

[1]decided either by the available bandwidth or by TCP friendly requirement

L	packet size
B	sending rate
p	packet loss rate
T_d	time between sending a packet and receiving its NACK
S_i	sending time of packet i (wrt. sender time)
R_i	received time of packet i (wrt. receiver time)
D_j	the time vertex split j is decoded at the receiver
w_j	the importance of vertex split j
$N_{d,t}$	number of packets decoded at time t
$N_{r,t}$	number of packets received at time t
Δ_t	increase in expected number of decodable packet at time t

Table 1: Notations used in our model.

5.1 Sending Time and Receiving Time

Let S_i be the time when the i-th packet is sent. Lemma 1 computes the distribution of S_i and the expected value $E[S_i]$.

Lemma 1 *If $i \geq T_d$, then for any $k \geq 0$,*

$$Pr(S_i = i + k) = \binom{i - T_d + k}{k} p^k (1-p)^{i-T_d+1}$$

$$E[S_i] = (i - T_d + 1)\frac{1}{1-p} + T_d - 1.$$

Otherwise, if $i < T_d$, then $S_i = i$.

Proof Sketch Whether a packet is sent successfully or not can be known only after T_d. At time $i + k$ ($k \geq 0$), the result of first $i + k - T_d + 1$ transmissions are known (we call them *known transmissions*), but the result of the following $T_d - 1$ transmission remains unknown (we call them *unknown transmissions*). For $t \geq T_d$, a new packet is sent at t only when the transmissions at $t - T_d$ is known to be successful. Hence, if packet i is sent at $i + k$, then among the $i + k - T_d + 1$ known transmissions, $i - T_d + 1$ transmissions are successful and k transmissions fail. The variable k has a negative binomial distribution, and the expression for $Pr(S_i = i + k)$ follows.

The total number of transmissions until a packet is successfully sent is a random variable following geometric distribution with expected value of $1/(1-p)$. Therefore, the expected value of total transmission number until $i - T_d + 1$ packets are successfully sent is $(i - T_d + 1)/(1-p)$. Including the subsequent $T_d - 1$ unknown transmissions, we have the expression for $E[S_i]$. □

Let R_i denote the time a packet i is received. The probability that a packet i is received at time t is given as follows.

Lemma 2

$$Pr(R_i = t) = \begin{cases} (1-p)p^{n_{i,t}} & \textit{if } (t - S_i) \mod T_d = 0 \\ 0 & \textit{otherwise} \end{cases}$$

where $n_{i,t} = \lfloor (t - S_i)/T_d \rfloor$ is the number of times packet i was lost when $R_i = t$.

Proof Sketch The lemma follows from the fact that a packet can only be received at a time that is a multiple of T_d after the first time it was sent. □

Note that here S_i is a random variable but we approximate the decoding time by using the expected value of S_i computed from Lemma 1. This approximation is accurate enough (as shown in the next subsection) as long as the variance of S_i is small[2].

Lemma 3

$$Pr(R_i \leq t) = 1 - p^{n_{i,t}+1}.$$

Proof Sketch Given t, the probability that a packet is received strictly after time t is the same as the probability that the packet has been lost $n_{i,t} + 1$ times, or $p^{n_{i,t}+1}$. Lemma 3 follows. □

5.2 Decoding Time of a Vertex Split

Once we expressed both sending time and received time, we can approximate the decoding time of a vertex split v, denoted as D_v.

Let $\mathcal{P}(i)$ be the set of all parent packets of packet i and itself, then the probability that vertex split v can be decoded at time t is given by the probability that one of the packet in $\mathcal{P}(i)$ is received at time t and all other packets in $\mathcal{P}(i)$ is received before time t.

$$Pr(D_v = t) = \sum_{j \in \mathcal{P}(i)} \frac{Pr(R_j = t)}{Pr(R_j < t)} \prod_{k \in \mathcal{P}(i)} Pr(R_k < t) \qquad (2)$$

Lemma 2 and 3 give the expression for $Pr(R_i = t)$ and $Pr(R_i < t)$ respectively. Once we have the probability distribution of D_v, we can estimate the expected decoding time of a vertex split v with

$$E[D_v] = \sum_{j=t}^{\infty} jPr(D_v = j), \qquad (3)$$

Since the probability $Pr(D_v = t)$ decreases exponentially as t increase, in practice we can numerically estimate the expected decoding time by considering only the first few terms of the sum. In this paper, we consider j from S_i to $S_i + 3T_d$, which we found to be accurate enough for practical purposes. That is, a packet is considered to be lost at most 3 times in a row. For larger loss rate, one can consider more terms to trade-off computation time and accuracy.

5.3 Validation

To validate the accuracy of our estimation of $E[D_v]$, we compared our analytical results with simulation results. The simulation considers the transmission of a progressive mesh model (we use the horse model with 48258 vertex splits) over a lossy network with random, independent loss rate ($p = 0.1$). Each simulation run registers the exact decoding time of a vertex split. The average decoding time of a vertex split over a number of runs is compared with the analytical results. Table 2 shows the average absolute difference and the maximum absolute difference in time unit (recall that one time unit is the time to send one packet). When we increase the number of simulation runs, the difference reduces further. The accuracy remains in the same range when we run the simulation using a real packet loss trace collected over 16 minutes between Toulouse, France and Singapore. Figure 4 shows the actual decoding time for one specific run. Our model fits the actual decoding time reasonably well. Table 3 explores the accuracy of our analytical model when T_d and p changes. We see that the difference remains low even when p increases beyond practical values.

[2]This approximation leads to the possibility of two packets arriving at exactly the same time. In this case, we break ties by randomly reordering the arrival of the packets

Number of runs	Average difference	Maximum difference
1000	0.474434	3.192292
10000	0.160840	1.566875
100000	0.122337	1.308333
trace	0.176589	2.183958

Table 2: Average and maximum absolute differences in decoding time between analytical model and simulation.

	$T_d = 20$	$T_d = 40$	$T_d = 60$
$p = 0.01$	0.064106	0.128288	0.227679
$p = 0.05$	0.155994	0.287506	0.454681
$p = 0.1$	0.304333	0.474434	0.709541
$p = 0.2$	0.480877	0.775148	1.033872
$p = 0.3$	0.676666	1.097639	1.504085

Table 3: Average absolute differences in decoding time between analytical model and simulation, for different p and T_d, after 1000 runs.

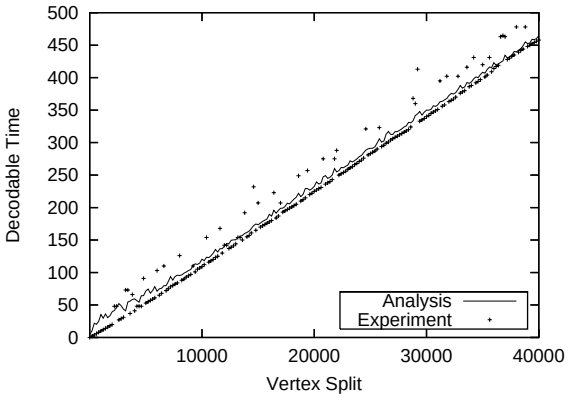

Figure 4: Comparison between decoding time obtained by one simulation run and decoding time computed from our analytical model. For clarity, we only plot 1 out of every 100 points in this figure.

The analytical model is useful in several ways. Besides allowing us to compute the expected decoded quality analytically, which is a faster alternative to simulation (see Section 6), our model can lead to some simple close form equations in some special cases. These equations can aid us in understanding the effect of dependency (see Section 7). Moreover, a better packetization algorithm can be designed based on this analytical model (Section 8). We introduce these applications of our model next.

6. EXPECTED DECODED MESH QUALITY

Using our analytical model, we can now analytically compute the expected decoded mesh quality given the dependencies among the vertex splits and the order the packets are sent.

In Section 4, we explain how we can view the area under the curve at time t, a_t, as either the area sum of vertical slices or horizontal slices. If we use the area sum of vertical slices, then $a_t = \sum_{i=0}^{t} q_i$, where q_i is the quality of the mesh at time i. Define

an indicator variable $x_{i,t}$ such that

$$x_{i,t} = \begin{cases} 1 & \text{if } D_i \leq t \\ 0 & \text{otherwise} \end{cases}.$$

Then we have $E[x_{i,t}] = Pr(D_i \leq t)$. Since $q_t = \sum_{i=0}^{t} x_{i,t} w_i$, we have $E[q_t] = \sum_{i=0}^{t} w_i Pr(D_i \leq t)$. Therefore,

$$E[a_t] = \sum_{i=0}^{t} \sum_{j=0}^{i} w_j Pr(D_i \leq j) \quad (4)$$

Using the area sum of horizontal slices, Equation 1 can be rewritten as $a_t = \sum_{i=0}^{t} x_{i,t} w_i (t - D_i)$, and the expected value becomes

$$\begin{aligned} E[a_t] &= \sum_{i=0}^{t} w_i E[x_{i,t} t - x_{i,t} D_i] \\ &= \sum_{i=0}^{t} w_i [t Pr(D_i \leq t) - E[x_{i,t} D_i]] \end{aligned}$$

We define $E[D_i|_{D_i \leq t}]$ as $\sum_{k=0}^{t} Pr(D_i = k)k$. Then, $E[x_{i,t} D_i] = E[D_i|_{D_i \leq t}]$, and the expected decoded mesh quality at time t is

$$E[a_t] = \sum_{i=0}^{t} w_i [t Pr(D_i \leq t) - E[D_i|_{D_i \leq t}]] \quad (5)$$

Equations 4 and 5 can be shown to be equivalent. Using Equation 2, which gives the distribution of D_i, we can analytically compute the expected decoded mesh quality at any time t using either equations. Equation 5, however, is more convenient in some cases. In Section 8, we will introduce an algorithm based on Equation 5.

We compare the decoded quality obtained analytically with the actual values obtained through simulation, where the Horse model is sent using FIFO packetization (see Section 8) with $p = 0$ and $T_d = 40$. Table 4 shows the results for eight time instances, averaged over 1000 runs. Our computed decoded quality closely matches the simulation results.

7. EFFECTS OF DEPENDENCIES

In this section, we investigate the effects of dependencies on the number of decodable packets, as a function of p and T_d. As mentioned in Section 2, packetization at the sender might affect the decoded mesh quality. We quantify this effect by estimating the number of decodable packets at a given time for two extreme cases of dependencies. The effects of any other dependencies will be bounded by what we obtained for these two extreme cases.

In the first case (ideal case), there is no dependency among packets. In the second case (worst case), a packet depends on all other packets sent before it. Note that these two dependency models are independent of any packetization scheme. Moreover, we are computing the number of decodable *packets* rather than *vertex splits*. But this simplification does not matter since in the ideal case, we can assume that the all vertex splits in a packet is decodable as long as it is received (no dependencies among packets). In the worst case, if a packet's dependency is not satisfied, then all the vertex splits contained in that packet are not decodable. Thus, the number of decodable packets is proportional to number of decodable vertex splits in this two extreme cases.

Let $N_{r,t}$ be the number of received packets at time t and $N_{d,t}$ be the number of decoded packets at time t. We show how to compute the expected value of $N_{d,t}$ for the two extreme cases in the next two subsections.

7.1 Ideal Case

Theorem 1 *In the case with no dependencies among the packets,*

$$\begin{aligned} Pr(N_{r,t} = k) &= \binom{t+1}{k} p^k (1-p)^{t+1-k}, \\ E[N_{r,t}] &= E[N_{d,t}] = (t+1)(1-p). \end{aligned}$$

Proof Sketch In the ideal case, each packet can be decoded independently regardless of other packets. Thus, at any time t, $N_{d,t} = N_{r,t}$. The number of successful transmission at time t is equivalent to the number of successful Bernoulli trials out of $t+1$ transmission[3]. Theorem 1 follows. □

7.2 Worst Case

In the worst case, a packet depends on all previous packets (packets with smaller index). The expected number of packets decodable at time t, is given by the following equation:

$$E[N_{d,t}] = \sum_{i=0}^{t} \prod_{j=0}^{i} Pr(R_j \leq t) \quad (6)$$

Obtaining a close form formula for $E[N_{d,t}]$ is more tricky. We will express the expected value as a recursive function.

The base case for the expression is when $t = 0$. In this case, the expected number of decodable packets $N_{d,0}$ is $1 - p$. Now, assume that we know $E[N_{d,t}]$. We now compute $E[N_{d,t+1}]$.

Suppose packet 0 is successfully received. We remove packet 0 from the dependency graph, reducing the dependency graph to another graph with the same dependency structure but with one less packet. The expected number of decodable packets from time 1 to time $t+1$ is the same as the expected number of decodable packets from time 0 to time t. Therefore, we have

$$E[N_{d,t+1}|_{\text{Packet 0 received}}] = E[N_{d,t}] + 1. \quad (7)$$

Now consider what happens if packet 0 is lost. We will consider the cases where $t + 1 < T_d$ and $t + 1 \geq T_d$ separately.

Let Δ_t be the increase of expected decodable number at time t, that is, $\Delta_t = E[N_{d,t}] - E[N_{d,t-1}]$. We let Δ_0 be $(1 - p)$ since the expected decodable number at time 0 is $(1 - p)$.

Lemma 4 *Suppose* $t + 1 < T_d$.

$$\Delta_{t+1} = (1-p)^{t+2}.$$

Proof Sketch If packet 0 is lost, packet 0 would not have been retransmitted by time $t + 1$. In the worst case, all other packets depend on packet 0 and cannot be decoded. That is,

$$E[N_{d,t+1}|_{\text{Packet 0 is lost}}] = 0 \quad (8)$$

Combining Equation 7 and Equation 8, we have

$$E[N_{d,t+1}] = (1-p)E[N_{d,t}] + (1-p)$$

Therefore, when $t > 0$, we have $\Delta_{t+1} = \Delta_t(1 - p)$. Since $\Delta_0 = 1 - p$, Lemma 4 follows. □

Case 2: $t+1 \geq T_d$**.** We first consider the subcase when $t+1 = T_d$ and packet 0 is lost in its first transmission and is retransmitted at time T_d. The expected number of decodable packets in this case is given by the following lemma.

Lemma 5 $E[N_{d,T_d}|_{\text{Packet 0 lost}}] = E[N_{d,T_d-1}]$

[3]Recall that the first transmission is at time 0

	t=20	t=40	t=60	t=80	t=100	t=150	t=200	t=400
Analytical result	89.84	236.40	491.20	832.37	1246.73	2516.83	4077.19	12399.4
Simulation result	90.29	237.67	487.33	828.28	1240.34	2520.09	4076.73	12394.8

Table 4: Comparison of evaluation with analytical model and simulation. Model: horse, $T_d = 40$, $p = 0.1$, averaged over 1000 runs.

Proof Sketch This lemma follows from Equation 6 and the fact that $Pr(R_0 < T_d) = Pr(R_0 = T_d|_{\text{Packet 0 is lost}})$. □

Now we consider the subcase when $t + 1 > T_d$ and packet 0 is lost. We introduce a new notation, $M_{d,t}(i, j)$, which is the number of packets with index from i to j inclusive, that are decodable at time t. We have the following two lemmas.

Lemma 6 *If $t + 1 > T_d$, then*

$$E[M_{d,t+1}(T_d, t+1)|_{\textit{Packet 0 is lost}}] = E[M_{d,t}(T_d, t)]$$

Lemma 7 *Let $t = nT_d + b$, where $n, b \in \mathbb{Z}^+$ and $0 \le b < T_d$, then*

$$E[M_{d,t+1}(0, T_d - 1)|_{\textit{Packet 0 is lost}}] - E[M_{d,t}(0, T_d - 1)] = \begin{cases} 0 & \textit{if } b = 0 \\ -\dfrac{1-p}{p}[1 - (1-p^{n+1})^{b+1}] & \textit{otherwise} \end{cases}$$

Proof Sketch This lemma follows from Equation 2, Lemma 2, and Lemma 3. □

The intuition behind Lemma 6 and Lemma 7 is as follows. Take two possible scenarios where the sequence of packet loss events are exactly the same, except that in one scenario, the first transmission of packet 0 is lost, and in the other, the results of this transmission is unknown. Now compare the sequence of packet transmitted in timeslots 1 to $t + 1$ when transmission 0 is lost with timeslots 0 to t when the result of transmission 0 is unknown, the only difference is the sending order of the first T_d packets.

Lemma 6 implies that to express $E[N_{d,t+1}|_{\text{Packet 0 is lost}}]$ in terms of $E[N_{d,t}]$, it suffices to consider packet 0 to $T_d - 1$, whose expected number of decodable packets is given by Lemma 7.

Combining the results in this section, we have the following.

Theorem 2

$$\Delta_t = \begin{cases} (1-p)^{t+1} & \textit{if } t < T_d \\ 1 - p & \textit{if } t \ge T_d \textit{ and } t = nT_d \\ (1-p^{n+1})\Delta_{t-1} & \textit{if } t \ge T_d \textit{ and } t = nT_d + b \end{cases},$$

where $n, b \in \mathbb{Z}^+; 0 < b < T_d$.

7.3 Insights

We have derived the expected decodable numbers of packets for two cases of dependencies, the ideal case where there are no dependencies between the packets, and the worst case, where a packet always depends on previously sent packets. Any packetization algorithms will lead to a dependency structure between the packets that lies between these two extreme cases. The difference between the number of decodable packets for these two cases therefore gives us an indication of how much improvements we can get if we intelligently group the vertex splits into packets. From our model, we make the following observations.

Observation 1 In Theorem 2, if t is large, and p is small enough, then the factor $1 - p^{n+1}$ tends to 1. Hence, as time progresses, the increase in number of decodable packets for the worst case becomes close to $1 - p$ for each time slot. If we plot two curves, showing expected number of decodable packets versus time, for the ideal and worst case, then the two curves becomes almost parallel for large t. In fact, if we consider the difference between the curve at time xT_d, then the difference tends to a constant that depends only on T_d and p as x tends to infinity. Figure 5 shows an example of such two curves, with $p = 0.1$ and $T_d = 40$.

This observation leads us to believe that optimization of packet dependencies only matters during the first few T_ds, after that, the dependencies among the packets will not affect the number of decodable packets. Further, in progressive mesh, the importance of the vertex splits decreases as the model becomes incrementally refined. Thus, the contributions of the later vertex splits to the decoded quality of the mesh are less important than those initial ones.

This observation is good news – regardless of how large the progressive mesh is, only dependencies among vertex splits sent during the first few T_ds matter. Thus, any packetization algorithm only needs to focus on the vertex splits sent during this initial period, reducing the computational costs significantly.

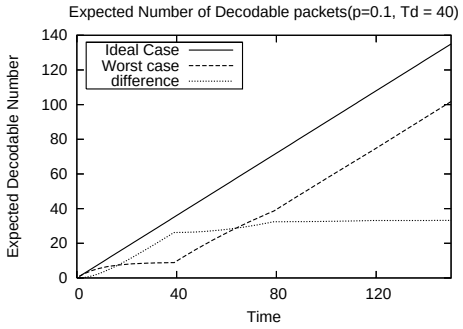

Figure 5: The number of decodable packets for ideal case and worst case. $T_d = 40$, Loss rate $p = 0.1$.

Observation 2 The next question is how big is the gap between the two extreme cases. Consider time $t = T_d - 1$. For the ideal case, the expected number of decodable packets is $T_d(1-p)$, while for the worst case, the expected number is

$$E[N_{d,T_d-1}] = \sum_{t=0}^{T_d-1}(1-p)^{t+1} = (1-p)\frac{1-(1-p)^{T_d}}{p}$$

We see that the gap between the two cases is $(T_d - \frac{1-(1-p)^{T_d}}{p})(1-p)$. Recall that T_d is the time between sending a packet and its lost is detected at the sender, expressed in units of a packet transmission time. Suppose with a typical RTT of 250 ms, a packet size of 1500 bytes, and a sending rate of 1.5Mbps, The value of T_d is 30. With a 5% loss rate, the gap between the two cases are about 15 packets, If we have about 100 vertex splits in each packet, then there is a 1500 vertex splits difference between the two cases, not a small number. The gap, however, decreases if the sending rate is smaller or RTT is smaller (reduces T_d). The gap also decreases when p decreases.

This observation surprises us. It says that dependencies in progressive meshes has smaller effect than dependencies in audio and

video when transmitted over a lossy network. This contradicts what we observe in practice, where dependencies among packets matter.

The reason for this mismatch is that our model gives us the *expected* number of decodable packets. Therefore, the average quality of the decoded mesh over a large number of streaming sessions will not differ too much. But, the *variance* for the number of decodable packets is large. Thus, a viewer who receives a progressive 3D mesh stream (a single sample) might still observe a significant differences initially if different packetization strategies are used. This realization motivates us to apply our model to packetize vertex splits during streaming of progressive meshes.

8. A PACKETIZATION ALGORITHM

Packetization refers to the process of grouping vertex splits into packets before transmitting them over the network. Due to dependencies among nodes, packetization may introduce dependencies among packets (if a vertex split and its parent are packed in different packets). The dependencies then affect the decoding time of the vertex splits and the intermediate quality of the decoded model.

To improve the intermediate quality of the decoded model, we need to consider two factors in the packetization: the importance of each vertex split, or node, and the dependencies among nodes. One strategy is to always send the most important node first; The other is to packetize the packets to minimize its dependency [7]. If there is no packet loss, these two objectives can be achieved at the same time. Since the vertex split operations in a progressive mesh are typically executed in the decreasing order, we can simply send the vertex splits in the first-in-first-out (FIFO) order, in the reverse order of the edge collapse operations. Since a node's parent packets are always sent before the packet containing the node, a node can be decoded as soon as it is received. When packet loss exists, however, there is a conflict between maximizing importance and minimizing dependencies. FIFO satisfies the first objective, but may increase dependency among packets; whereas to reduce dependency, one may send a node with lower importance before a more important node. To trade-off between these two objectives, we need to know the exact effect of both factors. In this section, we show how we use our model developed in Section 5 to help us select which node to pack.

Consider a node i. We need to decide whether we should pack i into the current packet. First, note that if there exists a parent of i that has not been packed, then we should not have packed i (if a parent of i arrives later than i, i cannot be decoded anyway). Thus, we only consider nodes whose parents have all been packed. Now, consider what would happened if we pack i into the current packet, versus the subsequent packet. From Equation 1, the difference in quality, δ_i, between the two cases is given by

$$\delta_i = w_i(E[D_i^{next}] - E[D_i^{curr}]), \tag{9}$$

where D_i^{curr} and D_i^{next} are the decoding time of i if i is packed in the current packet and next packet respectively. We call the metric δ_i as the *penalty*. To compute the penalty, we use Equation 3 from our model, which give us the expected decoding time for these two cases. Minimizing the penalty maximizes the difference in decoded mesh quality (Equation 5).

Equation 9 succinctly gives an expression that captures the trade-off between the importance of i and the dependencies. The penalty increases as w_i increases. Consider the case where i has a parent in the current packet. If we pack i in the next packet, then the (expected) decoding time for i increases – not only because it will arrive later, but also because this packing introduces a dependency between the current and the next packet. From Equation 3, we can see that additional dependencies increase the decoding time since both packets have to be received before i can be decoded. Therefore, increasing dependencies increases the penalty.

We can now describe a greedy algorithm to packetize progressive meshes. The algorithm simply packs the node with highest penalty at each step[4]. Algorithm 1 shows the pseudocode for packing one packet using our algorithm.

Algorithm 1 Greedy Packetization

```
for all node i whose parents are already packed do
    calculate its penalty δ_i if it is moved to the next packet;
    insert i into a maximum heap H with δ_i as key;
end for
while H is not empty and packet is not full do
    Pop j from H;
    Pack j into current packet;
    for all children k of j whose parents are already packed do
        calculate δ_k if k is moved to the next packet;
        insert k into H with δ_k as key;
    end for
end while
```

To reduce computation cost, we approximated δ_i by computing $E[D_i^{next}]$ and $E[D_i^{curr}]$ up to a limited number of terms. We chose to use up to $3T_d$ terms in our implementation. Further, from our observation in Section 7, since dependencies do not matter as time increases, we stop running the algorithm after time $3T_d$ and simply send the vertex splits in decreasing order of their importance.

8.1 Evaluations

We compare this greedy algorithm with three other algorithms: *FIFO*, *b-sub*, and *breadth first*. In FIFO, we send nodes in the original order, which is the reverse of the collapse order. The idea of this strategy is to send the most important node first. In b-sub, we send the most important node, then choose its descendent, whose parents are all packed, in a breadth first manner. It is an effective way to reduce the dependency between packets. In breadth-first strategy, we send the nodes in the breadth-first order in the DAG. The idea is to send the nodes in a way that maximizes the distance between parents and children.

We try the four algorithms on two meshes, the Horse model (48258 nodes) and the Happy Buddha model (538944 nodes), taken from Stanford 3D Scanning Repository. We perform experiments with various values of T_d and loss rate p. In these experiments, we use the length of collapsed edge as the importance of a vertex split. The results indicate that this greedy algorithm can improve the intermediate decoded quality for both models and for different network conditions. More importantly, it reduces the variance in decoded quality and ensures that the worst case seldom happens.

Figure 6 compares the intermediate decoded quality of the two meshes using different algorithms and using $T_d = 40$, averaged over 1000 runs. The loss rate p is either 0.05 ((b) and (e)) or 0.1 ((a),(c),(d), and (f)). Figure 6(a) and (b) show the average quality of the Happy Buddha model received versus time. The bump at $t = 40$ corresponds to the case where packet 0 (which affects the quality the most) is lost, retransmitted, and received at time $t = T_d = 40$. Moreover, when packet 0 is received, it triggers the decoding of other packets depending on it (directly or indirectly), causing a jump in expected decoded quality. We can see a bump of expected decodable packets as well in Figure 5.

[4]Technically this is a heuristic since it does not guarantee an optimal packetization. The packetization problem has been shown to be NP-complete [7].

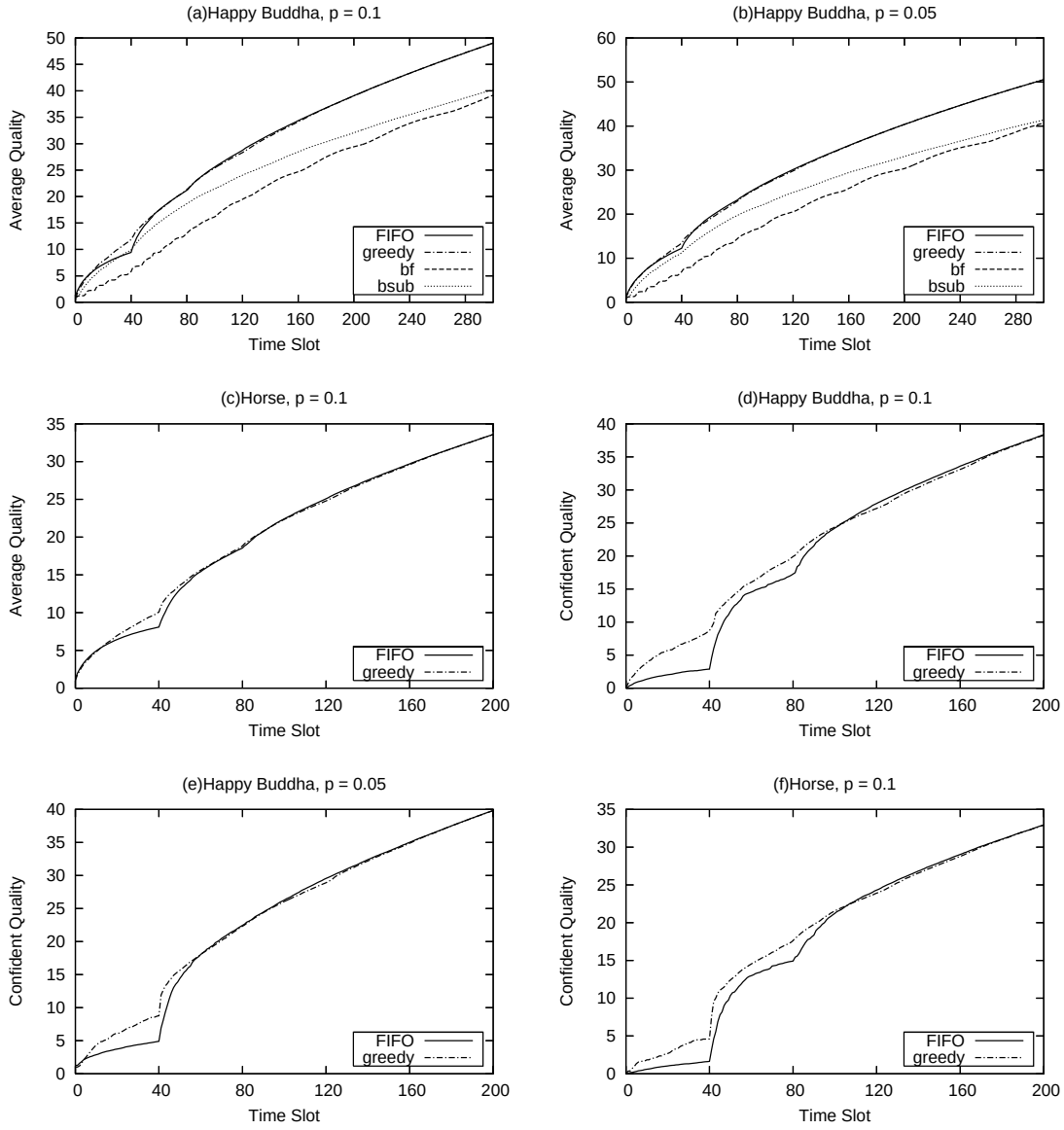

Figure 6: Comparison of decoded quality with $T_d = 40$.

We see that both FIFO and our greedy algorithm outperform breadth first (bf) and b-sub. The quality of greedy algorithm slightly outperforms FIFO in the first 40 slots. Figure 6(c) shows the average quality for the Horse model. We exclude the results of breadth first and b-sub since both resulted in poorer quality.

We further observe that, even though both greedy and FIFO resulted in similar average quality, in practice we encounter instances where greedy resulted in significantly better quality than FIFO. This observation is due to the variance in the mesh quality when we send using FIFO. Figure 7 shows the cumulative distribution of quality over 1000 runs at a particular time t = 39. We see that for FIFO, the quality is almost uniformly distributed and ranges from 0 to 14, whereas for greedy, the range is narrower. We therefore plot the 90% *cumulative quality* Q versus t – i.e., 90% of the runs resulted in quality higher than Q at time t. Figure (d), (e), and (f) compares the 90% cumulative quality for FIFO and greedy. We can see that greedy has a much better quality than FIFO – consistent with what we observe on individual runs of our algorithms.

To give our readers a sense of the visual difference in rendered quality, Figure 8 shows four intermediate versions of the Happy Buddha model with quality 2.82, 8.26, 9.25, and 11.44. These quality values correspond to the 90% cumulative quality of FIFO, 90% cumulative quality of greedy, average quality of FIFO, and average quality of greedy in Figure 6(a) and (d) at time 39.

9. CONCLUSION

We conclude by reflecting on what we learn from our model.

The most important insight we gain is that *the effect of dependencies among vertex splits on decoded mesh quality when transmitted over a lossy network is limited to the first few T_ds.* This has several implications. Since T_d is typically small, the effect of dependencies is not going to matter for non-interactive applications. Thus,

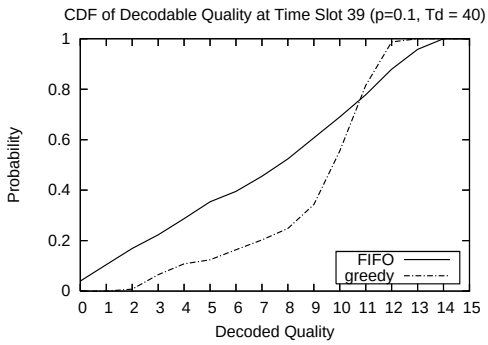

Figure 7: The CDF of the decodable quality at time slot 39 for the Horse model over 1000 runs with T_d = 40 and p = 0.1.

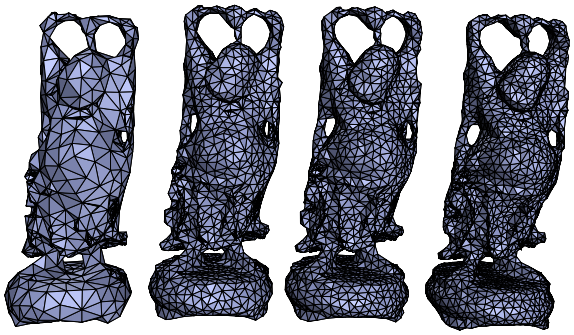

Figure 8: Partially rendered Happy Buddha. From left to right, the qualities are 2.82, 8.26, 9.25, and 11.44.

we believe that existing research to reduce dependencies (for instance, mesh segmentation [15, 17] and packetization [7]) is not relevant in this context if retransmission is used.

For interactive applications, the decoded mesh quality in the first few T_ds is crucial. We showed that FIFO ordering gives good enough average quality – even though a more intelligent greedy packetization can give better average quality with smaller variance.

Besides packetization, our insight that only first T_ds matters can lead to other transmission schemes. For example, the sender can protect the vertex splits sent during the first few T_ds using enough FEC coding, ensuring that there is no lost; or the sender can increase the sending rate temporarily during this brief period to improve the initial quality. Our model is still useful here. For instance, sending FEC would decrease the loss probability of some packets but would delay transmission of new data. Temporarily increasing the sending rate may also increase the loss rate due to congestion. Our model can characterize this trade-off and thus can guide the sender in deciding whether to send a FEC packet or new data. Our formula for expected decodable quality can similarly guide the sender in deciding how fast it should send during the first T_d period.

10. REFERENCES

[1] G. Al-Regib and Y. Altunbasak. An unequal error protection method for packet loss resilient 3D mesh transmission. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 743–752, June 2002.

[2] G. Al-Regib and Y. Altunbasak. 3TP: An application-layer protocol for streaming 3D models. *IEEE Transactions on multimedia*, 7(6):1149–1156, December 2005.

[3] P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 195–202, New York, NY, USA, August 2001.

[4] C. L. Bajaj, V. Pascucci, and G. Zhuang. Progressive compressive and transmission of arbitrary triangular meshes. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 307–316, Los Alamitos, CA, USA, October 1999.

[5] Z. Chen, J. F. Barnes, and B. Bodenheimer. Hybrid and forward error correction transmission techniques for unreliable transport of 3D geometry. *Multimedia Systems*, 10(3):230–244, March 2005.

[6] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 67–72, Los Alamitos, CA, USA, October 1999.

[7] Y. Gu and W. T. Ooi. Packetization of 3D progressive meshes for streaming over lossy networks. In *Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN)*, San Diego, CA, October 2005.

[8] A. F. Harris(III) and R. Kravets. The design of a transport protocol for on-demand graphical rendering. In *Proceedings of the 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 43–49, Miami Florida USA, May 2002.

[9] H. Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, August 1996.

[10] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3D graphics via remote rendering. *ACM Trans. Graph.*, 23(3):695–703, August 2004.

[11] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D scanning of large statues. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 131–144, July 2000.

[12] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, July 2002.

[13] C. Ondrejka. Escaping the gilded cage: User created content and building the metaverse. *New York Law School Law Review 81*, 49(1), January 2004.

[14] R. Pajarola and J. Rossignac. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, January 2000.

[15] S.-B. Park, C.-S. Kim, and S.-U. Lee. Error resilient 3-D mesh compression. *IEEE Transactions on Multimedia*, 8(5):885–895, October 2006.

[16] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 123–132, New York, NY, USA, July 1998.

[17] Z. Yan, S. Kumar, and C.-C. Kuo. Error-resilient coding of 3-D graphic models via adaptive mesh segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(7):860–873, July 2001.