Towards Characterizing User Interaction with Progressively Transmitted 3D Meshes

Ransi De Silva Wei Cheng Dan Liu Wei Tsang Ooi Shengdong Zhao
Department of Computer Science
National University of Singapore, Singapore
{ransi, chengwe2, liudan, ooiwt, zhaosd}@comp.nus.edu.sg

ABSTRACT

We collected traces of how 37 users interacted with 9 progressively streamed and rendered 3D meshes. We analyze the traces and discuss the insights that we learned in relation to design of efficient and scalable progressive mesh streaming systems. Our traces indicate that user actions are predictable and exhibit skewed access pattern. This finding could lead to design of efficient pre-fetching and caching techniques for progressive mesh streaming.

Categories and Subject Descriptors I.3.2a **[Graphics Systems]**: Distributed/Network Graphics

General Terms Human Factors, Measurements, Performance

Keywords Progressive Meshes, Interaction, User Behavior

1. INTRODUCTION

Advances in 3D scanning, range data collection, and digital sculpting techniques have eased the creation of high quality 3D meshes. These meshes are increasingly being shared over the Internet through progressive streaming, in applications such as virtual earth, virtual art gallery, and online shops. 3D meshes are typically data rich and demand high network bandwidth and computational power. For example, a regular mesh like the *Thai Statue* (Figure 1, left) has 5 million vertices (122 MB after gzip) takes 16 minutes to download even at 1 Mbps. When these meshes are viewed on portable devices with low bandwidth and slow CPUs, delays in both transmitting and rendering become unavoidable. To minimize negative user experience, the streaming system needs to prioritize data chunks according to user needs and efficiently cache the most frequently viewed data. Designing such systems requires a thorough understanding of user behaviors in viewing 3D meshes.

Most previous work on user interactions with 3D objects focused on design of specific interaction techniques (e.g., the study by Chen et. al [2] and Hinckley et. al [4]). This paper, however, studies user behavior from the system design point of view, such as predictability (for prefetching) and locality of access patterns (for caching), similar to the spirit of the landmark studies for the Web [5] and file system [7]. No such prior study exists for progressive meshes. This paper presents our first step towards filling this gap.

MM'09, October 19–24, 2009, Beijing, China.

We conducted a user experiment with 37 users interacting with 9 meshes. We log the user's actions while they interact and view the meshes in a mock online shop. We present the analysis of these traces in this paper, and highlight findings that are significant to the design of efficient and scalable progressive mesh streaming systems. In particular, we found that (i) in certain scenarios, user actions are highly predictable, making pre-fetching useful in these cases; (ii) users' viewpoint concentrates on part of the meshes, making caching these popular spots useful.

Figure 1: Meshes used in experiment. Left to right: Thai Statue (5 million vertices), Dragon (3.6 million vertices), and Happy Buddha (0.5 million vertices).

2. USER STUDY

Meshes. Three 3D meshes are chosen from the freely available Stanford 3D Scanning Repository[1]: *Happy Buddha*, *Dragon*, and *Thai Statue*. These meshes vary in complexity (amount of vertices), orientation, and symmetry in space from default viewing direction. *Happy Buddha* is the simplest, is vertically oriented, and has a default viewing direction orthogonal from the face of the Buddha. From that direction, the mesh is asymmetric between front and back. The geometric shape of *Happy Buddha* is somewhat representative of all human-like statues. *Dragon* is more complex and is horizontally oriented. The default viewing point is from one side of the body. Unlike *Happy Buddha*, it is front-back symmetric relative to the default viewing direction. The geometric shape of *Dragon* is somewhat representative of most mammals. *Thai Statue* is the most complex and is actually a compound mesh composed of three identical sides, each with three different objects: a Goddess, an elephant, and a dragon, stacking vertically from top to bottom. These three sides connect to form a triangular cylinder. There are three possible default viewing directions, one each from three corners of the triangular mesh. The *Thai Statue* is included as an example of complex compound mesh.

[1]http://graphics.stanford.edu/data/3Dscanrep/

We replicate each mesh above twice, generating nine meshes in total. We added one localized visual defect, by denting a small region, to each replicated mesh without changing its number of vertices or faces. The reason for adding the defects will be explained later. The location and nature of the defects vary between the meshes. As *Happy Buddha* is relatively simplier and has a smooth surface, its defect is more obvious than *Dragon* and *Thai Statue*. Defects for the later two meshes, due to the irregular surfaces, are hard to find unless the user zooms in considerably.

The meshes are encoded progressively and streamed over a simulated network of 320Kbps and 400ms round trip time.

Design and Procedure Our experiment mimics the following general real world scenario. Customers are shopping in an online antique store. Each product in the store has a number of items, and each item is represented by a 3D mesh closely resembling the corresponding real world item. These items vary in quality, and some have visible defects. Before purchasing, customers will carefully examine the available items for a product in order to pick the best item available for that product category.

We designed our experiment using a simple case of the above scenario. Our online store has three different products, corresponding to the three different meshes mentioned earlier. Each product has three available items in varying quality. Two of them have visual defects (note: defects are created without changing any mesh characteristics). Due to the different complexity of each mesh, the defects are easier to find in the simpler meshes (*Happy Buddha*) compared to the more complex ones (*Thai Statue* and *Dragon*).

The participants were first instructed about the keyboard commands to view and interact with the 3D meshes, and given brief practice of these commands on a simple mesh before starting the experiment. The participants were presented with an user interface mimicking an online catalog with three products. For each product, images of three items (i.e., three versions, one original and two with defects) are shown on the screen. The order of the products are randomized to avoid order effects. The participants' job is to pick the best item among the three. Each item can be viewed in any order and if desired, multiple times. When a participant selects an item, a new viewing window (width of 14cm and height of 15cm, with a resolution of 500x500 pixels) opens, and the 3D mesh corresponding to that item is progressively streamed and rendered in the window. The participants can interact freely with the mesh until they close the window. They would mark the item to be purchased after viewing all three items and move on to the next product. Each participant must go through all three products to complete the experiment.

During the experiment, users' key press actions and view point transformations were logged for offline analysis. The available keys along with its abbreviations are as follows, revolve clockwise (REC) and anti-clockwise (REAN), rotate clockwise (ROC) and anti-clockwise (ROAN), move up (MU), move down (MD), move right (MR), move left (ML), tilt back (TB), tilt forward (TF), zoom in (ZIN) and zoom out (ZOUT). The rotate, revolve, and tilt operations refer to rotating about the z-axis, y-axis, and x-axis, respectively. The axes follow the standard OpenGL convention.

Participants. A total of 25 male and 12 female participants, aged 19 to 36 (mean 23), mostly from the university community participated in the experiment. None had any visual handicaps.

3. RESULTS AND IMPLICATIONS

In this section, we present our analysis on the user behavior. We present only on the analysis of the original (non-defect) statues, unless otherwise specified. In addition, we also discuss what implications these results have on system design.

3.1 Session Length

Session length refers to the time each user spends in viewing a mesh. Generally, the session length is short, with the average values of 107s, 76s, 47s for *Thai Statue*, *Dragon*, and *Happy Buddha*, respectively (see Table 1). Figure 2(a) shows the distribution. The session length decreases with complexity of the meshes, as expected. The session length fits the *log-normal* distribution.

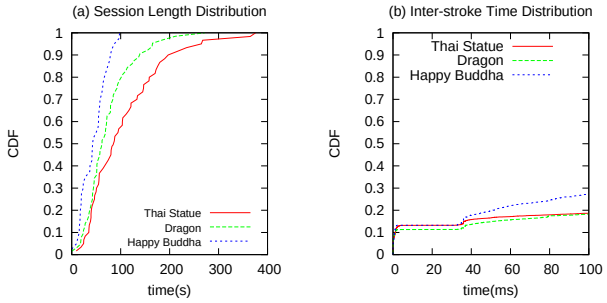

Figure 2: (a) Session Length and (b) Inter-stroke Time

Mesh	Session Length		Think Time	
	Mean(s)	Max(s)	Mean(ms)	Max(s)
Thai Statue	107	376	593	25
Dragon	76	272	574	20
Happy Buddha	47	98	403	13

Table 1: Session Length and Think Time

3.2 Think Time

We refer to the time between two key strokes as *inter-stroke time*. Consecutive key strokes of the same type with inter-stroke time smaller than 20 ms are grouped together as one *operation*. The time between two operations is *think time*. We choose the threshold of 20 ms because there is an obvious gap between 5 ms and 35 ms in the CDF of inter-stroke times for all meshes (see Figure 2(b)).

We find that think time follows similar distributions for all of the three meshes (Figure 3(a)). About 90% of the think time is smaller than a second. There is a jump in the curve of think time distribution for all the three meshes – about 5% of the think time clusters around 0.5 seconds. We hypothesize that this is related to the 0.4-second round trip time in our experiments. After a user performs an operation, it takes 0.4 seconds before the user can see progressive refinement of the mesh (although the system responses to the change in viewpoint *immediately*). Some users might wait for the refinement to come before the next operation.

The mean and maximum think time are shown in Table 1. We note that the maximum is up to 50 times larger than the mean.

To investigate the relation between think time and viewpoints, we classify the viewpoints into 4 regions: front-far (FF), front-near (FN), back-far (BF), and back-near (BN), based on *front/back* and *far/near*. We find that the think time distribution is not affected by the viewpoint (see Figure 3(b)).

3.3 Operations

Besides the session length and think time, it is also interesting to see if user actions are predictable. If so, pre-fetching can be used to improve the mesh quality and reduce response time.

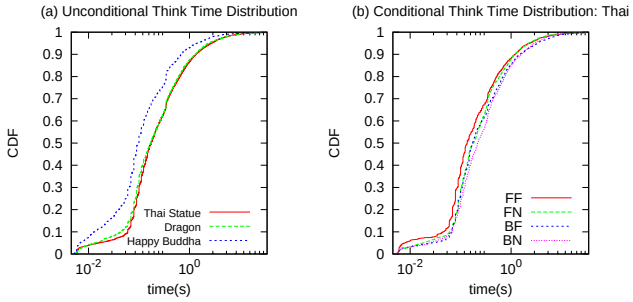

Figure 3: Think Time (x-axis in log scale).

Figure 4(a) shows the probability of occurrences of 12 different operations on the traces from each of 9 meshes, displayed as a bubble chart. The bubble size indicates the probability of an operation occurring for a mesh. We can see that the most frequently used operations are revolving (rotate around y-axis) and zooming. Further, zooming in is used more frequently than zooming out, indicating that users tend to zoom in to a comfortable distance to see the mesh with more details. Indeed, we observe that many users zoom in first, and then revolve around the y-axis.

With these distributions, we can do a simple prediction of the next operation to be taken by a user without any other information. The prediction could be more accurate, however, if we consider the current viewpoint of the user and the previous action of the user.

We first examine whether the probability of an operation relates to the viewpoint of the users. Figure 4(b) shows that the distribution of operations is different in the 16 regions for *Thai Statue*. For example, the probability of zooming in (ZIN) is higher in the right-top, front-far region (RTFF). Since the default viewpoint lies in this region, users tend to zoom in to a comfortable distance first and then move to other regions. *Dragon* (Figure 4(c)), however, shows different behavior. First, zoom-in is not as frequent in the default view, since *Dragon* has less details than *Thai Statue* and can be viewed comfortably with the default distance. Second, users frequently tilt, possibly due to the horizontal orientation of *Dragon*. Our findings indicate that the viewpoint affects the probability of taking a next operation, but this effect depends on the size and shape of the mesh, and thus is difficult to have a general framework for prediction based on viewpoints. Past history of user interactions for a particular mesh, however, could be used for more accurate mesh-specific prediction.

Figure 4(d) shows the conditional probability of taking the next operation given the previous operation for *Thai Statue*. The shaded diagonal shows that the same operation has the significantly larger probability (e.g., more than 0.93 for revolving) of being taken next. The other meshes exhibit the similar pattern. Such high predictability points to the efficacy of pre-fetching as a way to reduce response time and improve the viewing quality.

Finally, we consider the dependencies on viewpoint and previous operation together. Figure 4(e) shows the probability of taking a given operation in a given viewpoint region when the previous operation is zooming in for *Thai Statue*. Figure 4(f) gives such probability when the previous operation of moving down for *Dragon*. Both figures show that the dependence on viewpoint is non-negligible, but is still predictable for a given mesh. For instance, zooming out is more frequent after zooming in, if the viewpoint is nearer to the mesh. This behavior is expected. Finer division of viewpoint regions should yield more accurate prediction.

3.4 Access Pattern

Proxy caching of meshes is useful in reducing the service load. In this section, we show that the user traces exhibit access patterns that can lead to efficient caching.

Caching for Mesh Streaming. In mesh streaming, vertices are often grouped into chunks [3] for transmission. These chunks can be cached at a proxy to reduce server overhead. To study the usefulness of proxy caching, we look at the access pattern to chunks.

We first replay the log of operations from the users, and generate a list of chunks accessed by users during the experiment. From these chunk traces, we count the number of times each chunk is accessed. We then sort the chunks in decreasing order of the access count, and plot the cumulative access count versus rank in Figure 5(a). We normalize both axes to between 0 and 1 so that we can plot all three meshes on the same graph. Figure 5(a) shows how many requests we can satisfy (i.e., hit rate) by storing the most frequently requested chunks in a proxy. The x-axis denotes the number of chunks stored in the proxy, as a fraction of total number of chunks requested. It can be observed that by building a static cache that stores 20% of the most frequently accessed chunks, the proxy can achieve more than 70% hit rate for *Thai Statue* and *Dragon*, and 55% for *Happy Buddha*.

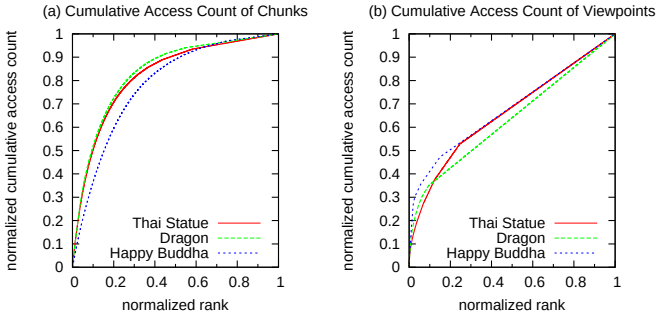

Figure 5: Cumulative Access Count versus Rank

Caching of Remote Rendering. For rendering on mobile devices [1] or for protection of mesh data [6], the server could send a rendered image directly according to the users' viewpoint. In this scenario, the caching proxy can cache the rendered images. We can similarly find the viewpoints "visited" by the most users. A viewpoint visited multiple times by the same user is only counted once, since the user can keep the received image locally and need not request it the second time. Figure 5(b) shows a plot similar to Figure 5(a), but for access frequency of viewpoints. The figure shows the hit rate at the caching proxy if we choose to store pre-rendered images corresponding to the most frequently accessed viewpoints. The distribution is not as skewed as access count for chunks, but still, caching the rendered images for 20% of the most frequently accessed viewpoints can yield 40 - 50% hit rate.

Caching of Vertices and Pixels. Caching mesh data in graphic card memory (e.g. using VBO (Vertex Buffer Object) and PBO (Pixel Buffer Object) supported in OpenGL), could significantly increase the rendering speed when the memory bandwidth is the bottleneck. For graphic cards without enough memory to store the whole mesh, we could just store the most frequently viewed part of the mesh in the graphic card memory.

We replay all the user traces, and count the number of times each face is viewed. We normalized the number of views of each face and visualize them with a heat map (Figure 6(a)). We can see that

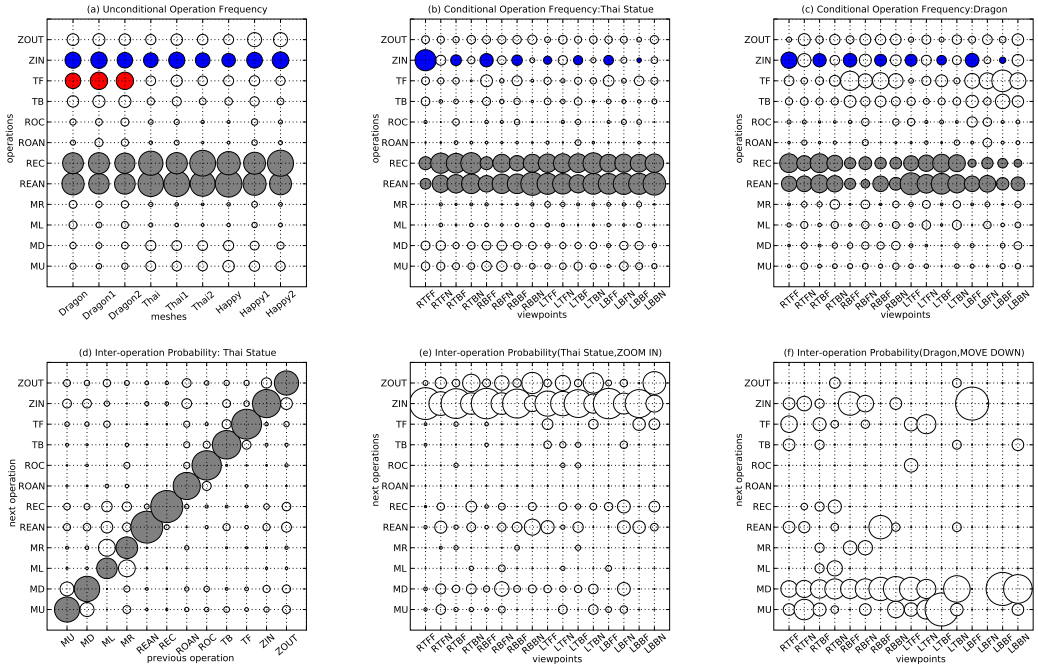

Figure 4: Frequency of User Actions

the most frequently viewed region of *Happy Buddha* (viewed 4205 times) is the base between the two legs because it is visible from both the front and the back. Figure 6(b) plots the normalized cumulative view count of faces versus rank, similar to Figure 5. We can see that the locality is slightly less than that in the previous two scenarios, but for *Happy Buddha* and *Thai Statue*, hit rate of 40% can be achieved by storing 20% of the most frequently viewed faces in the graphic card memory. The mesh *Dragon* has the least locality. We hypothesize that this is because people tend to view *Dragon* at many different viewpoints due to its complex shape, leading to more evenly distributed viewpoints around the mesh.

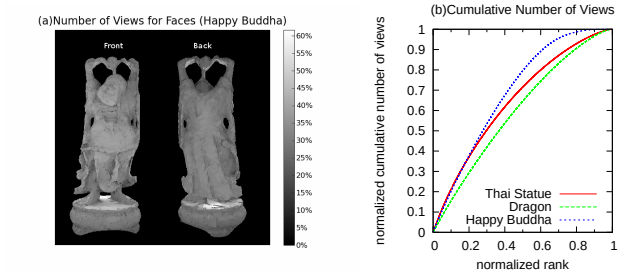

Figure 6: (a) The normalized number of views for each face. (b) The hit rate if we partially cache faces on the graphic card.

4. CONCLUSION

The analysis of user traces reveals that user actions are predictable to certain extent. The prediction based on previous action is simple and effective. Further, locality exists in both data and viewpoint access. By storing the most popular mesh data in caching proxies, the server overhead could be significantly reduced.

We plan to conduct another user study on the effect of network parameters on user interaction with the progressively streamed meshes. In particular, we are interested in understanding the user's tolerance level to delay and bandwidth in the network, the two factors that affect the time and rate at which a mesh refines its resolution.

Acknowledgement: This project is supported by National University of Singapore Academic Research Fund R-252-000-306-112.

5. REFERENCES

[1] P. Bao and D. Gourlay. A framework for remote rendering of 3-D scenes on limited mobile devices. *IEEE Transactions on Multimedia*, 8(2):382–389, 2006.

[2] M. Chen, S. Mountford, and A. Sellen. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of SIGGRAPH'88*, pages 121–129, Atlanta, GA, 1988.

[3] W. Cheng and W. T. Ooi. Receiver-driven view-dependent streaming of progressive mesh. In *Proceedings of NOSSDAV '08*, pages 9–14, Braunschweig, Germany, 2008.

[4] K. Hinckley, J. Tullio, R. Pausch, D. Proffitt, and N. Kassell. Usability analysis of 3D rotation techniques. In *Proceedings of ACM UIST'97*, pages 1–10, Banff, Canada, 1997.

[5] B. A. Huberman, P. Pirolli, J. Pitkow, and R. Lukose. Strong regularities in world wide web surfing. *Science*, 280(5360):95, 1998.

[6] 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, 2004.

[7] J. K. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of ACM SOSP'85*, pages 15–24, Orcas Island, WA, 1985.