3D Mesh Preview Streaming

Shanghong Zhao, Wei Tsang Ooi, Axel Carlier, Géraldine Morin, Vincent Charvillat

3D Mesh Preview Streaming

Shanghong Zhao
Dept. of Computer Science
National University of Singapore
shzhao17@comp.nus.edu.sg

Wei Tsang Ooi
Dept. of Computer Science
National University of Singapore
ooiwt@comp.nus.edu.sg

Axel Carlier
IRIT-ENSEEIHT
University of Toulouse
axel.carlier@enseeiht.fr

Geraldine Morin
IRIT-ENSEEIHT
University of Toulouse
morin@n7.fr

Vincent Charvillat
IRIT-ENSEEIHT
University of Toulouse
charvi@n7.fr

ABSTRACT

Publishers of 3D models online typically provide two ways to preview a model before the model is downloaded and viewed by the user: (i) by showing a set of thumbnail images of the 3D model taken from representative views (or keyviews); (ii) by showing a video of the 3D model as viewed from a moving virtual camera along a path determined by the content provider. We propose a third approach called preview streaming for mesh-based 3D object: by streaming and showing parts of the mesh surfaces visible along the virtual camera path. This paper focuses on the preview streaming architecture and framework, and presents our investigation into how such a system would best handle network congestion effectively. We study three basic methods: (a) *stop-and-wait*, where the camera pauses until sufficient data is buffered; (b) *reduce-speed*, where the camera slows down in accordance to reduce network bandwidth; and (c) *reduce-quality*, where the camera continues to move at the same speed but fewer vertices are sent and displayed, leading to lower mesh quality. We further propose a keyview-aware method that trades off mesh quality and camera speed appropriately depending on how close the current view is to the keyviews. A user study reveals that our keyview-aware method is preferred over the basic methods.

Categories and Subject Descriptors: [Computer Graphics]: Graphics Systems and Interfaces; [Multimedia Information Systems]: Multimedia Streaming

General Terms: Algorithms, Human Factors, Performance

Keywords: Progressive Mesh Streaming, 3D Preview, Camera Path

1. INTRODUCTION

Advances in hardware and software for 3D acquisition, editing, and viewing, have led to an increasing number of 3D models published online. 3D artists can now easily share the 3D models they build in an online portfolio, using Web sites such as p3d.in[1] or Sketchfab[2]. Cultural artifacts are being scanned in 3D and published freely online through Web sites such as 3D-COFORM[3].

Many of these 3D publishing services provide a way for users to quickly preview a 3D model before a user decides to download, view, and interact with the model. A common method is to display thumbnail images of the 3D model, taken from representative views (or keyviews). Another method is to provide a video depicting the 3D model from a moving camera that navigates a path that connects through the keyviews. Image-based preview requires less transmission bandwidth, but is limited in the amount of information about the model conveyed to the users. Users need to mentally make the connections between the different viewpoints, resulting in higher cognitive load. On the other hand, video-based previews allow smooth transitions between the different viewpoints, but demand higher network bandwidth for transmission.

We consider a third approach to preview a 3D model in this paper, which we term as *3D preview streaming*. 3D preview streaming, like video-based previewing, displays the 3D model at the client as viewed from a moving camera that navigates along a predefined path, connecting the keyviews. Unlike video-based previewing, however, 3D preview streaming downloads and renders part of the 3D model visible from the current viewpoint.

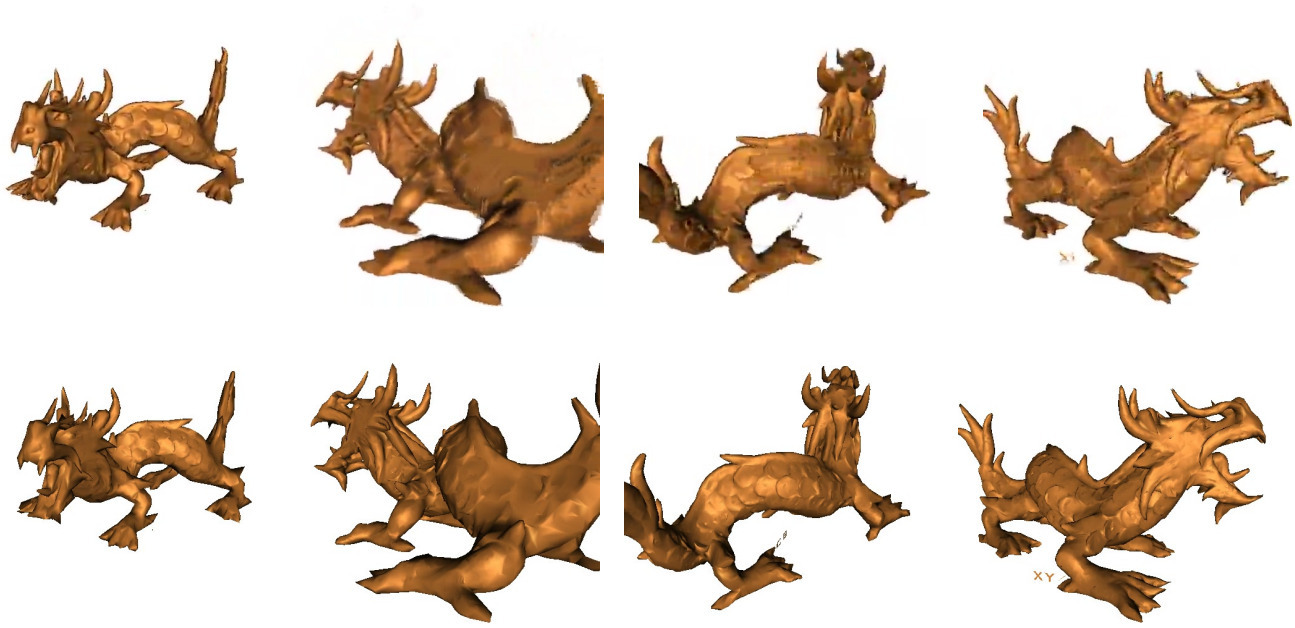
3D preview streaming has many advantages over video-based preview. First, if the user decides to interact with and view the 3D model after watching the preview, the 3D model information has already been (partly) downloaded and can be reused. Whereas for video-based preview, the video frames are useless for 3D rendering. Second, since 3D rendering is done at the client during preview, the user can choose to watch the preview under different rendering parameters (e.g, in different lighting condition, different surface materials, in wireframe or faceted mode). The video-based preview approach would require a new video to be produced for each combination of rendering parameters. Third, depending on the 3D model's geometric complexity and representation, 3D preview streaming may require lower bandwidth than video-based preview. Finally, for a 3D model represented as a progressive mesh, existing progressive streaming and rendering techniques can be used to adapt the quality of the model dynamically depending on the available network bandwidth and on the display resolution. While scalable video coding and rate adaptation can be used for streaming video-based preview as well, the adaptation of quality is done in the 2D domain, not in the 3D domain, leading to lower perceptual quality of the 3D model.

[1] http://p3d.in

[2] http://shw.gl

[3] http://www.3d-coform.eu

MMSys'13, February 26-March 1, 2013, Oslo, Norway.



(a) 10 second (b) 25 second (c) 40 second (d) 55 second

Figure 1: Comparison between video-based preview (top) and 3D-based preview (bottom).

Figure 1 illustrates the last point above. The figure shows four views of the Asian Dragon transmitted using video-based preview (top row) and 3D-based preview (bottom row). Both video and 3D mesh are transmitted at a constant rate of 40kbps. The video depicts the mesh as viewed along a 1-minute camera path (see Figure 5); the 3D mesh is streamed along the same camera path. The snapshot at time 10s, 25s, 40s, and 55s, are shown. The video-based preview is encoded from a rendering of the highest quality mesh along the camera path into H.264 video format[4]. We can observe that blocking artifacts are visible in the video-based preview, and some details on the surfaces of the mesh are lost. In the 3D-based preview, the details of meshes are still visible.

To exploit the adaptability of progressive meshes, we focus our work on preview streaming of 3D progressive meshes.

3D mesh preview streaming has many similarities with video streaming: (i) Users watch a media content passively as it is being downloaded; (ii) The media content changes with time, and there is a deadline on when the downloaded data needs to be displayed; (iii) To ensure smooth playback (no missed deadline), start-up delay can be introduced to buffer sufficient data; (iv) To perform rate control, the quality of the data can be lowered to reduce the data rate (either by lowering the frame rate, frame resolution, or with a higher compression ratio).

There exist, however, several intrinsic differences between 3D mesh preview streaming and video streaming. Firstly, the display rate of a video is constrained by the frame rate. While one can reduce the data rate of a video stream by exploiting temporal scalability and reducing the frame rate, this leads to reduction of display rate and undesirable stuttering of video during playback. On the other hand, for 3D mesh preview, the display rate of the 3D data is controlled by the rendering rate, whereas the data rate is controlled by the speed of moving camera. To reduce the data rate in the case of 3D mesh preview streaming, we can reduce the camera speed while keeping the rendering rate constant, thus maintaining smooth playback.

Secondly, in video streaming, the content of every video frame is normally assumed to be equally, semantically, important. In 3D mesh preview streaming, however, the keyviews are more important and convey more important content than the intermediate viewpoints along the camera path.

Thirdly, in video streaming, the rate control decision can be done independently for a frame or a group of frames. For 3D mesh preview streaming, different viewpoints may share the same mesh surfaces, reducing the amount of bits sent for a given viewpoint could therefore increase the number of bits required for a subsequent viewpoint, leading to non-trivial rate control algorithms.

These unique characteristics of 3D mesh preview streaming lead to new techniques (and challenges) for rate control (and the closely associate buffering control). In this paper, we explore three such basic techniques and propose a fourth, hybrid, technique. The first technique, STOP-AND-WAIT, stops the camera at a keyview when the sending rate reduces, and starts moving the camera again when enough data is buffered. This technique is similar to how video playback pauses when there is insufficient data in the buffer. However, by stopping at keyviews that are semantically important (and not arbitrary video frame in video streaming case), users can spend more time examining the 3D model from the important views. The second technique, REDUCE-SPEED, slows down the camera movement when the sending rate reduces, and resumes normal camera speed again when enough data is buffered. This technique is analogous to reducing the video frame rate, but as explained above, is able to maintain smooth display rate due to decoupling of rendering rate and data rate. The third technique, REDUCE-QUALITY, exploits the property of progressive meshes and lowers the quality of the mesh (by sending fewer refinements) when sending rate reduces (but maintains constant camera speed). This technique is analogous to sending fewer layers in a scalable video, but with the reduction of quality done in the 3D domain.

[4]using FFmpeg 0.11.2 with arguments `-vcodec libx264 -b 40k -s 500x500`

STOP-AND-WAIT can tolerate the most amount of reduction in bandwidth, but is most disruptive to the user viewing experience. We found that for both REDUCE-SPEED and REDUCE-QUALITY, they can adapt to bandwidth reduction up to a certain threshold without causing perceptually obvious differences.

To handle larger changes in the sending bandwidth, we propose to combine both techniques to simultaneously slow down the camera movement and reduce the mesh quality. This hybrid technique allows the preview to adapt to the importance of the viewpoint and trades off mesh quality with camera speed. As the camera is near the keyviews, the mesh quality is reduced as little as possible, while the camera movement is slowed down. Conversely, the camera speed is maintained as much as possible when viewing other less important regions of the mesh, but the mesh quality is reduced.

This paper is the first to propose the notion of 3D mesh preview streaming. Our contributions are: First, we introduce 3D mesh preview streaming as an approach to preview 3D models (in Section 3). Second, we explore three basic approaches to adapt to bandwidth fluctuation, exploiting the unique characteristics of 3D mesh preview streaming (in Section 4). Finally, we propose a hybrid approach to adapt to bandwidth fluctuation and view importance (Section 5) and show that it leads to more perceptually pleasing previews than the baseline approaches in a user study (Section 6).

2. RELATED WORK

2.1 3D Mesh Streaming

We now present previous work on 3D mesh streaming. Previous work considers situations where users are free to interact with the model. None has considered the problem of mesh preview streaming, where the camera moves but no interaction is possible. Meeting playback deadline is not a central issue in these works, but is a core consideration of our approach. The concerns in these previous works include how to improve the quality of the received mesh as fast as possible, mitigate distortion of the rendered mesh in the presence of packet losses, and scale to a large number of users.

Al-Regib and Altunbasak [1], Li et al. [20], and Chen et al. [7] investigated how to choose between different standard transport protocols for transmissions, trading reliability and delay. Harris III and Kravets [16] designed a new transport protocol that exploits loss tolerance and partially-ordered property of 3D objects that are organized into trees of bounding volumes.

Several existing works also consider error control techniques at the application layer. Park et al. [23] and Yan et al. [34] segmented a progressive mesh into smaller partitions that are loss resilient. Al-Regib et al. [2] and Chen et al. [7] studied error correction for 3D transmission, while Park et al. [22] and Tang et al. [29] investigated error concealment methods. Cheng et al. [10] and Tian [30] used retransmissions for error control. We do not consider error control in this paper. We believe many of these methods can be adapted into 3D preview streaming easily, by additionally considering playback deadline in the error control decision. Such situation is similar to video streaming and well-known solutions exist.

Cheng et al. studied two ways to scale 3D mesh streaming to many users. First, a receiver-driven approach with stateless server is proposed to reduce the computational overhead at the server [9]. Second, a peer-assisted approach is used to reduce the bandwidth overhead at the server [8]. We consider a client-server architecture in this work. The computational overhead of the server, however, is less since the camera path is pre-determined: the server does not need to compute which vertices are visible given a viewpoint as when the user interacts with the 3D model. The use of peer-assisted approach for 3D preview streaming remains an open problem.

The final set of previous work we want to highlight is that of De Silva et al. De Silva et al. studied 3D progressive mesh streaming from the user perspective. They characterized how users would interact with a 3D mesh [11] and measured how well users would tolerate slow mesh refinements and high response time [12]. The finding on high tolerance to response time (up to 1s) is relevant to our work, as it indicates that users are willing to view lower quality meshes and supports our REDUCE-QUALITY approach to 3D preview streaming. Our user study results validate this finding as well.

2.2 3D Object Preview

The related work on 3D object preview focuses on automatic determination of keyviews and generation of camera paths for previews. Many papers have addressed the problem of finding the best view, or a set of best views for a scene. We focus here on techniques for a single object, and discard the treatment of a whole scene. Early work by Kamada and Kawai [18] chose non-degenerated views. Some recent work focuses on maximizing a criterion based on the 3D geometry, like maximizing the number of polygons seen or the area covered [25], the total curvature [28], the viewpoint entropy [32], or the view stability [31]. Dutagaci et al. [13] gave a nice comparison of state of the art methods. Perceptual criteria have also been proposed: gathering user experience [35], or tracking eye motion [4, 24]. Some of these techniques generalize naturally to more than one view; such as the work of Fiexas et al. [14] which used the entropy of the same object.

Creating a virtual camera path is a common problem in robotics, gaming, or virtual walkthrough. The goal, however, is different. In both contexts, the virtual environment is usually complex scene, and criteria like collision detection are main concerns, whereas visual quality may not be as important. The navigation in complex worlds proposes the use of cinematographic rules [26], or dedicated language [21] or interactions for the camera control [19]. We indeed have a simplified framework. Here we start from a set of keyviews, selected by an expert or the content provider for their aesthetic and relevance for describing the 3D object. We need to seamlessly, or at least smoothly, link the keyviews corresponding to the chosen views with an unsupervised approach (we do not want the user to interact for choosing the path). In [15], Han et al. chose among the visited keyviews using a shortest path algorithm. We actually require to visit each chosen keyview and thus need order the chosen keyviews. We also consider a different cost function between the keyviews; whereas their cost function depends on the similarity of the keyviews (weighted distance between position, viewing angles and saliency), our cost function will take into account the streaming cost. Moreover, their camera path lacks smoothness between keyviews, which leads to a shaky effect in the resulting videos. Both Burtnyk et al. [5] and Andújar et al. [3] used smooth paths, respectively modeled by Bézier or Hermite polynomial curves. Similarly, we use Catmull-Rom splines, in order to interpolate our keyviews and have a smooth path.

3. A STREAMING FRAMEWORK FOR 3D MESH PREVIEW

3.1 Background

Before we describe the framework for 3D mesh preview streaming, we need to describe some background terminologies and relevant techniques that enable preview streaming.

Progressive Mesh. We begin by describing how a 3D mesh is progressively represented, following Hoppe's model [17]. A 3D

mesh can be progressively represented by repeatedly performing the *edge collapse* operation on the edges in the mesh. Each edge collapse operation merges two neighboring vertices into one, therefore simplifying the mesh by reducing the number of vertices by one. The simplified mesh after a sequence of operations is called the *base mesh*. We can obtain the original mesh from the base mesh by reversing the edge collapses, using an operation called *vertex split* that takes a merged vertex and splits it into two vertices.

A progressive mesh can be represented with a forest of binary trees, where the root of each tree is a vertex in the base mesh. Every intermediate node is a vertex split, with its two children as vertices that are split from the parent. The leaf nodes in the forest form the vertices of the original mesh.

View-dependent Progressive Mesh Streaming. Streaming of progressive mesh is done by transmitting over the base mesh, followed by a sequence of vertex splits. As the client receives the vertex splits, it updates the base mesh, and renders the updated mesh with increased levels of detail. As a result, the user sees the mesh refining itself as more vertex splits are received.

In view-dependent progressive mesh streaming, only vertex splits that are visible from the current viewpoint are sent. Further, only vertex splits that cause changes larger than one pixel (in the screen space area) are transmitted.

There are two ways view-dependent streaming is achieved. In server-driven approach, the client sends the current viewpoint to the server. The server determines which vertex splits are visible and streams them to the client. In receiver-driven approach, the client estimates the vertex splits that are visible and requests them from the server.

3.2 Mesh Preview Streaming

We now introduce the notion of 3D mesh preview streaming, which is a specific form of view-dependent progressive mesh streaming. In 3D mesh preview streaming, there is no interaction between the user and the 3D model. The sequence of views (viewpoints and synthetic camera parameters) is predetermined in the form of a parametric camera path $P(\cdot)$, which is pre-computed by the server and communicated to the client. In addition to the camera path, the server also precomputes the in-view vertex splits along the path before streaming, and transmits the vertex splits in sequence to the client. The client buffers the base mesh and all the vertex splits of the initial view before it starts playing back the preview.

The client renders the 3D mesh according to the camera path $P(t) = (Pos(t), C(t))$: at time t, the vertices visible from viewpoint $Pos(t)$ are rendered using the synthetic camera parametrized by $C(t)$. $C(t)$ is a set of camera parameters specifying near plane, far plane, field of view, color model, etc. Ideally, the client has received all the vertex splits needed to refine the visible part of the mesh to its full level of detail by time t. The server determines which vertices to send based on the camera path. Ideally, the server always sends all vertices visible from viewpoint $P(t_1)$ before the vertices visible only from $P(t_2)$ if $t_1 < t_2$. Under the above ideal conditions, the client would playback the preview of the mesh smoothly at uniform camera speed and at the highest level of detail as the viewpoint moves along the camera path.

3.3 Bandwidth-Aware Camera Path

We now present our method to compute the camera path. We assume that a set of representative views, called *keyviews*, is given as input. These keyviews can be set manually by the content provider, or can be algorithmically determined using the existing methods (see Section 2.2). Figure 2 shows some examples of keyviews selected for Lucy (details about this model are given in Section 6).

Figure 2: Four of the chosen keyviews for Lucy.

A good camera path needs to have the following properties: (i) the camera path must pass through the keyviews, (ii) the path should travel directly and smoothly between two neighboring keyviews on the path. For preview streaming, we add two more properties: (iii) the amount of data shared between two successive keyviews should be as much as possible, reducing the fluctuations in bandwidth requirements as the viewpoint moves along the camera path, and (iv) the camera path must be smooth with no sudden changes in the gradient of the path, especially near the keyviews.

We construct a weighted complete graph $G = (V, E, w)$ where each vertex in the graph is a keyview. Every two keyviews are connected with an edge (u, v) with the cost $w(u, v)$ defined to be $|S_m| - |S_p(u, v)|$, where S_m is the size of the mesh, and $|S_p(u, v)|$ is the size of vertices visible from both keyviews u and v. The cost function favors visiting a neighboring keyview sharing a maximum of visible vertices.

The problem of finding the sequence of keyviews to visit that would minimize the total cost is then equivalent to the traveling salesman problem, which is NP-complete. For our use case, the number of keyviews ($|V|$) is small (e.g., we use 12 keyviews for Lucy) and the path/tour is computed offline, so we use a brute force approach. We could alternatively, for more complex paths, use a heuristic by Rosenkrantz et al. [27] to find a tour in G. This tour yields the ordering of the sequence of keyviews to visit.

We did not consider the distance between two neighboring keyviews in the camera path in our graph model. Two keyviews can be relatively close and still very little overlap in data if the camera is pointing at two very different directions at the keyviews. The converse is true: two keyviews with positions relatively far from the object may be far apart and still share large overlaps in the model. Using distance as a cost to minimize during path construction would lead to view sequences that swing wildly. Figure 3 shows an example. We note that, in previous work that constrain the camera position to be on a bounding sphere with the orientation of the camera pointing towards the center, such issue does not exist and thus it suffices for them to find the minimal distance between the neighboring viewpoints when determining the visiting order (such as in [33]).

From the ordered sequence of keyviews, we compute the parametric camera path P as an interpolating curve (so the property (i) is insured). Catmull-Rom splines are a classical model of smooth

Figure 3: Two sequences of keyviews selected by using different costs. Top: using distance between keyview positions. Bottom: using the number of overlap vertices. The top sequence leads to swinging views, while the bottom sequence is more coherent.

interpolatory curves, since such curves are C^1-continuous. Recently, Yuksel et al. [6] have proposed different parameterizations of Catmull-Rom splines leading to different tension in resulting curves. For an arbitrary starting time t_0, the i-th keyview is visited at time $t_i = t_{i-1} + ||Pos(i) - Pos(i-1)||^{\alpha}$, where $Pos(i)$ is the viewpoint position corresponding to the i-th keyview, $\alpha = 0$, 0.5, and 1 correspond respectively to uniform, centripetal and chordal parameterizations. Figure 4 shows the Catmull-Rom curves for these three different parameterizations. We choose the chordal parameterization on the position of the viewpoints. There are three advantages in choosing this parameterization. First, the behavior does not depend on the relative difference in the positions of the keyviews. In particular, the keyviews are far from being uniformly distributed (since they are chosen for their visual interest), the chosen interpolating curve does stay close to the line joining only successive nearby positions. Second, between two far away positions corresponding to two successive keyviews, a relatively long path is derived, leading to intermediate keyviews that stay at a relatively constant distance to the object. The chosen model quite naturally avoids collision with the model. Finally, the parameterization is fairly regular, which is important for us as we sample views along the parametric path: regular parameters thus lead to quite regularly sampled positions on the path. We use this property in the following to get an approximate arc-length parameterization.

3.4 Basic Streaming Algorithm

We now describe how the server and the client work in tandem to enable 3D mesh preview streaming.

First of all, we need an arc-length parameterization of the camera path, that is, such that the position $Pos(t)$ is at a distance on the path of $t - t_0$ from the starting point $Pos(t_0)$. To do that, we compute dense samples on the curve from equally-spaced samples in the parameter space. The distance on the curve is approximated by the length of the piecewise linear approximation of the sample points. This approximated arc length parameterization is used in the following.

We discretize the camera path into N sample viewpoints corresponding to parameters taken at an equal distance d. The time taken for the camera to travel between two sampled views is 1 unit time. The camera speed is therefore d. We assume the camera speed is constant in the ideal streaming algorithm (unlimited bandwidth).

The server computes $M(t)$, the set of vertices visible from any point along the curve between $P(t)$ and $P(t+1)$ (inclusive) that have not appeared before in any set $M(t')$ for $t' < t$. These are the vertices that have not been seen before if the client follows the camera path from $P(0)$ to $P(t)$. The size of the data in $M(t)$ can also be precomputed. We denote the size of $M(t)$ as $B(t)$ (in bits). The playback rate, the rate in which the data are consumed and display, is $B(t)$ (in bits per unit time) in this ideal case.

The server algorithm is simple: for each time $t = 0, 1, \ldots, N-2$, the server sends $M(t)$ with the data rate of $B(t)$. Suppose we start the clock $t = 0$ at the client one unit time later (plus some delay to account for network latency and jitter) than the server. Then, assuming that the data rate of $B(t)$ is sustainable, this algorithm ensures that the client will receive all data of $M(t)$ by time t for rendering within the next unit time.

Now we consider the case where the server has a maximum sending rate of r. Let $B_{max} = \max\{B(i)|0 \leq i \leq N-2\}$. If $r < B_{max}$, it is possible that at some time t where $B(t) = B_{max}$, the client would not have received enough data to display the mesh at full resolution. To counter the mismatch between the playback rate and the sending rate, the client can wait for an additional B_{max}/r time before starting to render. This start-up delay ensures that the client would continue to playback the 3D preview smoothly, and is analogous to the start-up delay introduced in video streaming.

4. HANDLING BANDWIDTH VARIATIONS

We now consider what would happen if r reduces during streaming. This condition would happen for several reasons. First, there could be a congestion in the network, causing the server to clamp down the congestion window, effectively reducing the value of r.

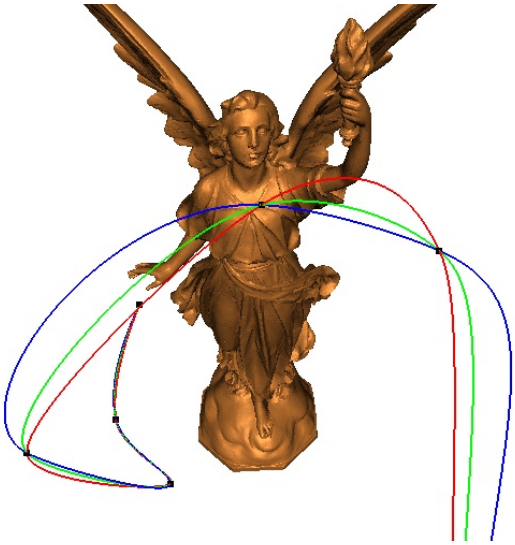


Figure 4: Path given by Catmull-Rom splines, interpolating the camera positions of keyviews (black squares). Uniform (red curve), centripetal (green curve) and chordal (blue curve) parameterization lead to different tension parameters. Chordal is chosen since tension is applied on shorter segments (close by keyviews) and the camera path is less tense on longer segments (far apart keyviews).

Second, there could be new clients in the system, causing the server to reduce the value of r for one client to re-allocate the bandwidth for the new clients.

A reduction in r would cause the client not to download enough data, since the start-up delay is no longer sufficient to absorb the difference in playback rate and sending rate. While it is possible for the client to be conservative and have a higher start-up delay than B_{max}/r to absorb the variation in r, higher start-up delay sacrifices user experience. Furthermore, if r drops significantly for a longer period of time, there is no guarantee that the higher start-up delay is sufficient.

As such, we need to support variation of the sending rate r in 3D preview streaming. In the rest of this section and the next section, we introduce three basic techniques and a more sophisticated technique that is a hybrid of the three, answering the following question: what should the client (and the server) do when there is insufficient data to display the mesh at full resolution?

4.1 Pause During Buffering

An obvious solution for the client is to pause the camera at viewpoint $P(t)$ until all data in $M(t)$ are completely received. This method, however, may lead to frequent pausing and jittery camera movement, and is therefore undesirable.

An alternative is for the client to pause and rebuffer enough data to ensure smooth playback for the rest of the camera path. Assuming that the current sending rate $r(t)$ stays above a value r_{min} for the rest of the playback. Since $B_{max}(t) = \max\{B(i)|t \leq i \leq N-2\}$, if the client stalls for $B_{max}(t)/r_{min}$ before starting to move the camera again, then there would be sufficient data for the client to render along the rest of the camera path.

The above techniques, however, do not exploit a unique property of 3D mesh streaming: some viewpoints are more important and interesting than others. With such semantic information available, we note that, if we need to pause, it is better to pause at one of the keyviews. This idea has led to our first method to handle bandwidth variation. The method works as follows: let the sequence of keyviews along the camera path be $P(R_1), P(R_2), ..., P(R_k)$, where $P(R_1) = P(0)$. After the initial start-up delay, the client first waits for all vertices for $P(R_2)$ to arrive (which also implies that all vertices on preceding viewpoints have arrived). The client then starts moving the camera. Upon reaching the viewpoint $P(R_i)$, if the vertices for $P(R_{i+1})$ have not been received, then the client pauses and waits until the vertices for $P(R_{i+1})$ has been received, before start moving again. This strategy might lead to longer buffering time initially, but if it needs to pause for rebuffering, it always does so at a keyview that is interesting and important.

We collectively call the above three methods as STOP-AND-WAIT, and call the variations as "naive", "rebuffer", and "at-keyview" respectively.

Note that, for this technique, the server still sends all the vertex splits even when $r(t) < B(t)$. The consequence is that the server will take longer than one unit time to transmit $M(t)$, increasing the duration of the preview.

4.2 Reducing Mesh Quality

Another obvious strategy for the client, if it has not fully received $M(t)$ by time t, is to render the set of received vertices anyway, resulting in a mesh of lower level-of-detail. We call this technique REDUCE-QUALITY. Since progressive mesh is used and the client already has the base mesh, the client always has some data to display. The advantage of this approach is that the camera moves continuously at full speed, leading to smooth movement and preserving the preview time of the ideal path.

Unlike the previous technique, the server stops sending $M(t)$ after one unit time and starts sending vertices in $M(t+1)$. The server's camera speed in this case remains constant.

Note however that at the end of the preview, the model will not be available at full resolution, as indicated in Table 2.

4.3 Reducing Camera Speed

The previous two basic techniques are minor variations of well known techniques in the domain of video streaming. We now consider the third technique, which capitalizes on the unique property of 3D mesh preview streaming. Since the rendering rate and playback rate are decoupled, we slow down the playback rate by slowing down the camera when the sending rate $r(t)$ reduces. We call this technique REDUCE-SPEED.

To control the speed of the camera at the client, the client keeps track of the latest viewpoint received and rendered. Suppose the client is currently rendering viewpoint $P(t_{disp})$ and the latest complete viewpoint received is $P(t_{recv})$.Consider what happens when the sending rate reduces. Just like in STOP-AND-WAIT, the server slows down its camera movement to ensure that full mesh quality is received at the client, allowing t_{disp} to catch up with t_{recv}. To control the camera speed, the client computes periodically, the *catch-up time*, t_{catch}, defined as the time it takes before $t_{disp} = t_{recv}$.

If t_{catch} falls below a threshold, the client slows down the camera. To recover, the client speeds up the camera if t_{catch} increases beyond another threshold (up to the original camera speed). We limit the camera speed to fall within the range $[S_{min}, S_{max}]$, where S_{max}, the maximum speed, is the original camera speed specified for the path.

4.4 Discussion

There are a few points we want to highlight about these basic algorithms before we propose a more sophisticated algorithm.

First, for ease of exposition, we described STOP-AND-WAIT and REDUCE-SPEED as resulting in full quality mesh being rendered at the client. As such, there is actually no need for progressive meshes to be used. In our system, however, we included an optimization that adapts the rendering of mesh to the screen resolution (no triangle with screen area below one pixel is sent). We also support out-of-core rendering for large 3D models, where the out-of-view region of the mesh is collapsed to reduce memory requirement. Progressive mesh is needed to support these functionalities.

Second, if $r(t)$ drops significantly for a long period of time, REDUCE-SPEED could slow the camera to halt, reducing it to a STOP-AND-WAIT-like scheme. In both STOP-AND-WAIT and REDUCE-SPEED the duration of the preview is not bounded. At the opposite, REDUCE-QUALITY could end up showing only the base mesh, as little or no data is being received. Neither of which is desirable.

Third, REDUCE-SPEED is more appropriate for when $P(t)$ is close to a keyview, as it gives users more time to view the mesh near the interesting regions. On the other hand, REDUCE-QUALITY is not at all appropriate, as it reduces the quality of the mesh around the interesting regions.

Fourth, we observe that both REDUCE-SPEED and REDUCE-QUALITY do not decrease the perceptual quality of the mesh significantly, as long as $r(t)$ does not decrease too much.

The last three points have led us to design a new keyview-aware scheme that combines all three basic schemes above, to trade off between speed and quality.

5. KEYVIEW-AWARE STREAMING

Let $I(t)$ be the importance of view $P(t)$. We set $I(t)$ to 1 at each keyview and 0 at the midpoint between two successive keyviews. For other points along the camera path, we interpolate $I(t)$ linearly between 1 and 0.

The server algorithm for the keyview-aware scheme is fairly simple: for each time $t = 0, 1, ..., N-2$, the server computes its camera speed $S_s(t)$ as the following:

$$S_s(t) = S_{min} I(t+1) + S_{max}(1 - I(t+1)).$$

With the above adaptation, the server's camera moves at the slowest speed S_{min} at the keyviews. Note that the control of the server's camera does not (directly) affect the client's camera speed. Rather, by slowing down the server's camera, the server has more time to transmit more vertices, leading to a higher mesh quality.

To slow down the camera at the client near the keyviews, the client controls its camera speed $S_c(t)$ in a similar way as the server:

$$S_c(t) = S_{mid} I(t+1) + S_{max}(1 - I(t+1))$$

except that S_{mid} is a speed threshold chosen to be higher than S_{min}. We use S_{mid} instead of S_{min} here, since a large change in the client's camera speed is directly perceivable by the user, we keep the variations in the camera speed smaller.

The control above, however, does not adapt to sending rate $r(t)$. We need to further adapt the camera speed at the client by considering the gap between t_{disp} and t_{recv}, similar to REDUCE-SPEED in Section 4.3. Unlike REDUCE-SPEED, however, the client's camera speed is also dependent on the importance, complicating the calculation of t_{catch}. We therefore use a slightly different method. We let l_{gap} be the distance (along the camera path) between $P(t_{disp})$ and $P(t_{recv})$, and k be an input parameter that determines how speed scales with l_{gap}. As l_{gap} becomes narrower due to a lower $r(t)$, the camera slows down naturally. The updated formula to compute the camera speed is thus:

$$S_c(t) = \min\{k \times l_{gap}, S_{mid} I(t+1) + S_{max}(1 - I(t+1))\}$$

Note that the above camera speed is continuous. If $r(t)$ is too low, then KEYVIEW-AWARE would pause and rebuffer whenever the buffer becomes empty, reducing it to STOP-AND-WAIT method.

6. EVALUATION

In this section, we present the evaluation results of different 3D preview streaming schemes proposed in this paper, as well as a user study to measure the perception of different solutions.

6.1 Experimental Set Up

We used three 3D models for evaluation: Asian Dragon, Thai Statue, and Lucy, from the Stanford 3D Scanning Repository. We converted the models into progressive meshes for streaming. For Asian Dragon and Thai Statue, the vertex coordinates are encoded with a lossy compression algorithm [9] and are of reduced size. We rendered each model on a 500×500 pixels canvas with OpenGL. Table 1 summarizes the meshes we used and some basic performance metrics of preview streaming. All measurements are done with a Linux PC equipped with Intel Core 2 Duo 2.40 GHz CPU, 4 GB memory, and NVIDIA Quadro FX 3500 graphics card.

We conducted three experiments in total. The first two experiments used Lucy, the largest one of the three models. We used Thai Statue in the third experiment to further validate the results from the second experiment.

We picked 12 keyviews manually for Lucy and 10 keyviews for Thai Statue (four of the keyviews for Lucy are shown in Figure 2), and constructed the camera path as described in Section 3.3.

For evaluation, we implemented six schemes in 3D preview streaming: GROUND-TRUTH, STOP-AND-WAIT naive, STOP-AND-WAIT at-keyview, REDUCE-QUALITY, REDUCE-SPEED, KEYVIEW-AWARE schemes. GROUND-TRUTH assumes the outgoing bandwidth is unlimited and therefore the preview always has the best quality and constant camera speed.

To simulate the network using TCP with a maximum outgoing bandwidth $r(t)$, we set up both server and client on the same machine. $r(t)$ is tuned by `tc`, Linux traffic controller. The data rate $r(t)$ for Lucy starts from 100 KBps and switches between 25 KBps and 100 KBps every 30 seconds (see Figure 7). We use this range of values as it roughly matches the variation in bandwidth required to transmit the Lucy model. Similarly, the data rate $r(t)$ for Thai Statue starts from 50 KBps and switches between 10 KBps and 50 KBps every 20 seconds.

Before we present the evaluation, we first illustrate these schemes in greater details. Table 2 shows the duration of the preview and the amount of received data for different schemes for Lucy. The duration indicates the amount of time the user would need to watch through the whole preview (in other words, the time the camera takes to move along the whole camera path). REDUCE-SPEED leads to preview that lasts more than twice the duration of the GROUND-TRUTH, while REDUCE-QUALITY leads to three times fewer vertices being received, reducing the mesh quality. KEYVIEW-AWARE has duration and quality that fall in between both schemes. Figure 6 illustrates the quality of three selected keyviews when streamed using the three non-ideal schemes. Note the mesh quality is significantly and noticably lower in the middle column.

6.2 Quantitative Results

Figures 7 and Figures 8 (a)-(d) illustrate how different parameters of REDUCE-SPEED, REDUCE-QUALITY, KEYVIEW-AWARE for Lucy change over time. Since the schemes lead to different camera speed, the x-axes of the graphs are shown in units of the path length. The vertical lines indicate the positions of the keyviews.

Measure	Asian Dragon	Thai Statue	Lucy
Number of Vertices	3.6 million	5 million	14 million
Number of Triangles	7.2 million	10 million	28 million
Size of Transmitted Mesh	58.7 MB	81.2 MB	428.1 MB
Size of Base Mesh	7.9 KB	10.3 KB	7.0 KB
Number of Vertices in Base Mesh	38	16	264
Size per Vertex-Split	17 bytes	17 bytes	32 bytes
Precision of Vertex Coordinates	17 bits	17 bits	32 bits
Render Frame rate	15-20 fps	10-18 fps	12-20 fps
Network Overhead	244 bytes	244 bytes	340 bytes

Table 1: Measurements of 3D mesh preview streaming along a camera path (see Figure 5)

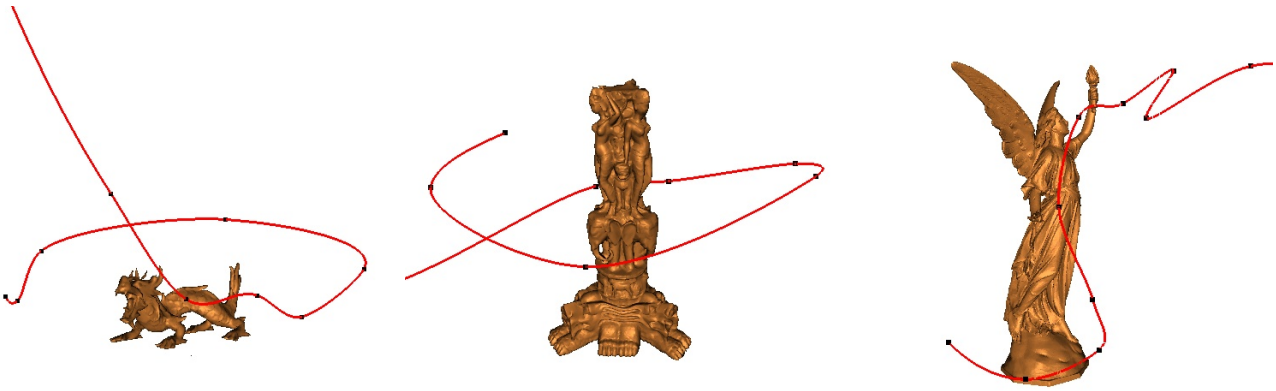


Figure 5: Camera paths of Asian Dragon, Thai Statue, and Lucy. The black points are the camera positions of keyviews.

Scheme	Duration	Received
GROUND-TRUTH	84 seconds	14.6 MB
STOP-AND-WAIT naive	202 seconds	14.6 MB
STOP-AND-WAIT at-keyview	210 seconds	14.6 MB
REDUCE-QUALITY	88 seconds	5.3 MB
REDUCE-SPEED	219 seconds	14.6 MB
KEYVIEW-AWARE	148 seconds	8.8 MB

Table 2: Statistics of preview streaming schemes for Lucy.

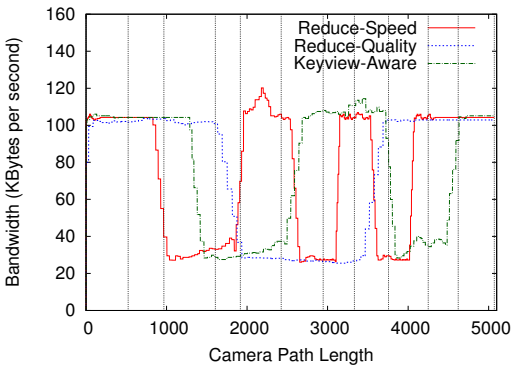


Figure 7: Bandwidth of preview streaming schemes for Lucy, estimated on the client side. The camera path length is measured with OpenGL coordinate units. The simulation of bandwidth for Lucy starts from 100 KBps and switches between 25 KBps and 100 KBps every 30 seconds.

We first focus on the REDUCE-SPEED scheme (in red). Since there are much data to transmit initially (Figure 8(d)), the gap is small (Figure 8(c)), causing the camera to slow down significantly (Figure 8(a)) compared to other schemes. The quality of REDUCE-SPEED (Figure 8(b)), however, is the highest.

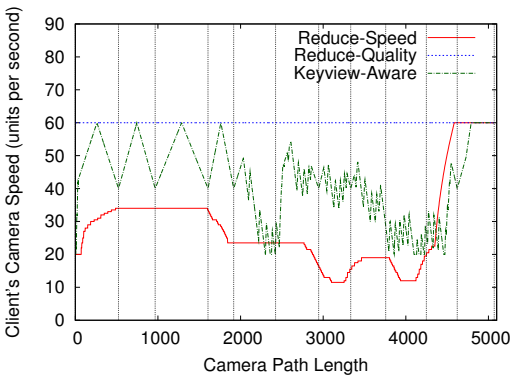
Next, we illustrate the REDUCE-QUALITY scheme (in blue). The camera speed is constant (Figure 8(a)), but the quality is the lowest (Figure 8(b)), and therefore the least amount of data being transmitted (Figure 8(d)).

The graph for the KEYVIEW-AWARE scheme (in green) is the most interesting. The camera speed (Figure 8(a)) fluctuates between S_{mid} and S_{max}, during a period of high sending rate. After 30 seconds, the camera speed is forced to slow down due to low sending rate. At the same time, the gap (Figure 8(c)) is kept relatively low. The interdependency between the camera speed and gap causes both values to fluctuate during this period. The number of rendered polygons (Figure 8(b)) falls between the other two schemes, but peaks at keyviews. This observation is more obvious at length 3700. There is an increase in the number of rendered polygons compared to REDUCE-QUALITY scheme.

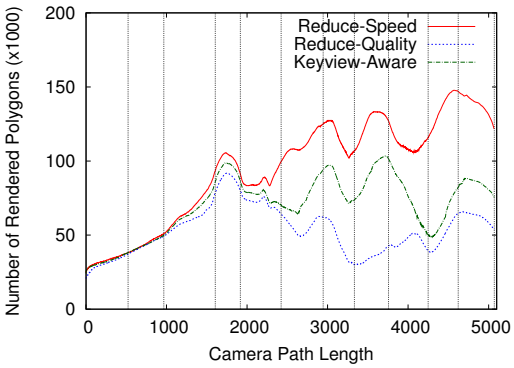
6.3 User Study

We now present the results from a user study, which quantitatively evaluates the proposed strategies.

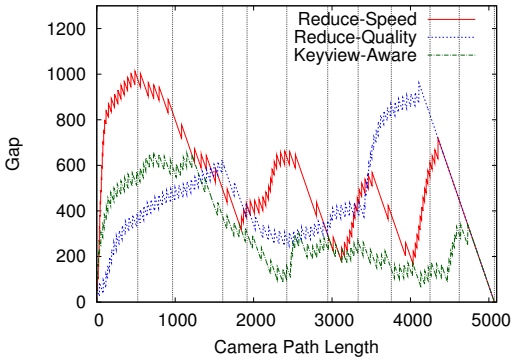
Set-up. We streamed and rendered the 3D meshes using different preview strategies. To allow the participants in our user study to experience these different strategies without installing our client, we captured the rendered frames and encoded them as high quality video clips, with each video clip corresponding to one mesh and one preview strategy. We then set up a Web site for the participants to access, view, and rate these video clips. This Web site has two pages. On the first page, we present the GROUND-TRUTH



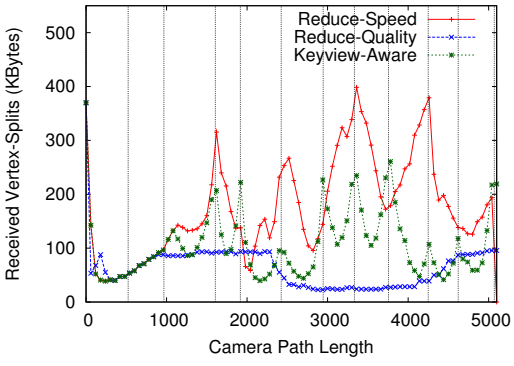
(a) Client's Camera Speed



(b) Number of Rendered Polygons



(c) Gap between Receiving and Rendering Viewpoints



(d) Received Data Size

Figure 8: Evolution of preview streaming schemes for Lucy, recorded on the client side. Both the camera path length and the gap are measured with OpenGL coordinate units.

Preview	Accept. Speed	Accept. Quality	Average Ranking
STOP-AND-WAIT naive	10%	100%	2.5
STOP-AND-WAIT at-keyview	10%	90%	2.3
KEYVIEW-AWARE	90%	80%	1.2

Table 3: Statistics from the Experiment 1. Percentage of participants finding the speed or the quality is acceptable (accept.) for a given video, and the average ranking.

Experiment	NOEXP	NOV	AVG	KNOW	EXP
1	1	3	5	0	1
2	10	2	5	7	1
3	9	5	2	5	0

Table 4: User experience levels in the experiments. NOEXP: no experience; NOV: novice; AVG: average user; KNOW: knowledgeable; EXP: expert.

along with some instructions and explanations about the overall context of the study. After reading the instructions and viewing the GROUND-TRUTH, the participant moves to a second Web page, where we display three different previews that can be viewed independently, and a form to collect participants' answers to questions. The three videos are displayed in a random order generated using a script. The answers are sent to and stored in a remote database. It took between 10 to 20 minutes for a participant to complete the study, depending on the level of attention.

Form. For each preview of the second Web page, we ask each participant whether each of the quality and the speed is acceptable. We do not provide any clue regarding what "acceptable" means, and let the participants judge according to GROUND-TRUTH and their own idea of what those parameters should be. Then the participants are required to sort the previews from 1 to 3, 1 being the best video and 3 being the least appealing to them. The participants can explain their choices and leave comments in a specific dialog box. Finally, we ask the participants to state their age, gender, and experience with 3D modeling.

Experiment 1: STOP-AND-WAIT versus KEYVIEW-AWARE. We first evaluated two STOP-AND-WAIT schemes against KEYVIEW-AWARE using the Lucy model. The results of this comparison appear in Table 3. The results of this study were very consistent, and we thus tested only 10 participants (all males, see Table 4). Only one of the 10 participants found the varying speed of the STOP-AND-WAIT strategy acceptable. All others did not find stopping the camera during preview acceptable. Whether the camera stopped at a keyview or not did not make a significant difference. So, clearly, STOP-AND-WAIT is not a reasonable strategy for 3D previews.

Experiment 2: REDUCE-QUALITY versus REDUCE-SPEED versus KEYVIEW-AWARE using Lucy. A total of 25 participants, 5 females and 20 males with age ranging from 23 to 45 and averag-

Preview	Acceptable Speed	Acceptable Quality	Average Ranking
REDUCE-QUALITY	100%	60%	1.92
REDUCE-SPEED	4%	96%	2.68
KEYVIEW-AWARE	80%	96%	1.4

Table 5: Statistics from the Experiment 2. Percentage of users finding the speed or the quality is acceptable for a given video, and the average ranking.

Preview	t Stat	t Critical	P(T<=t)
STOP-AND-WAIT naive	4.333	2.262	0.002
STOP-AND-WAIT at-keyview	2.905	2.262	0.017
REDUCE-QUALITY	2.316	2.064	0.029
REDUCE-SPEED	6.532	2.064	0.000

Table 6: Results of a two-tail t-Test on the Experiment 1 and 2.

ing 27, participated in the user study (see Table 4). Table 5 presents the average ratings of each video. 19 of the participants left a comment, most of them insisting on the fact that they were looking for a better quality but that the length of REDUCE-SPEED was making it really boring to watch. As explained in their comments, participants almost always ranked REDUCE-SPEED as the worst because the camera was too slow. The only participant that ranked this video as 1st was an expert user, who explained his[5] choice in his comment that he considered quality of the model as the most important parameter. REDUCE-QUALITY was ranked 2nd, usually because participants were able to spot mesh refinements and because in some regions of the statue (the eyes, the feet), the quality was too visibly degraded. The KEYVIEW-AWARE scheme was therefore ranked 1st, and was designated by participants as the most convincing compromise between the quality of the model and the length of the video (determined by the camera speed).

A two-tail Student's t-Test paired for means was performed to determine if KEYVIEW-AWARE has a superior user ranking. The null hypothesis is that the average ranking of any other scheme is equal to that of KEYVIEW-AWARE. If a test of significance gives a p-value lower than the significance level α, the null hypothesis is rejected. In Table 6, since the t-statistic > t-critical and p-value < α (α = 0.05) , we can reject the null hypothesis. Therefore, the average ranking of KEYVIEW-AWARE is the best (see Table 3 and Table 5) and has statistically significant difference.

Overall, the user study rules out both the STOP-AND-WAIT and the REDUCE-SPEED mostly because of the duration of the preview. Whereas users are ready to compromise on the quality, they still prefer the hybrid solution, offering a compromise between (a smooth) speed and quality.

Experiment 3: REDUCE-QUALITY versus REDUCE-SPEED versus KEYVIEW-AWARE using Thai Statue.

To further validate our findings, we repeated Experiment 2 using a different model, the Thai Statue. We had 21 participants in this experiment (5 females and 16 males, with age ranging from 21 to 63 and an average age of 28). The results are summarized in Table 7 and Table 8. As in the Experiment 2, most users preferred the KEYVIEW-AWARE preview for the Thai Statue: 19 users out of 21 ranked KEYVIEW-AWARE as 1st. Further, REDUCE-QUALITY was more appreciated than REDUCE-SPEED: 12 users commented that low speed was less acceptable than low quality. Namely, they complained that the duration of REDUCE-SPEED really impaired the user experience despite the ideal quality. This result was coherent with Experiment 2. Table 8 shows the statistical significance of the user preference to KEYVIEW-AWARE.

7. CONCLUSION

3D mesh preview streaming is a new form of continuous media transmissions that contains characteristics of both fully interactive 3D progressive mesh streaming and video streaming. We introduce a basic framework for 3D mesh preview streaming and identify several basic ways the system can adapt to varying bandwidth. We also introduce an approach that controls both the camera speed and the mesh quality, while being aware of the keyviews of the model. User study reveals that the keyview-aware approach is indeed preferred over the basic approaches.

[5]we use he as a gender-neutral pronoun to refer to a participant

Preview	Acceptable Speed	Acceptable Quality	Average Ranking
REDUCE-QUALITY	90%	29%	2.29
REDUCE-SPEED	10%	100%	2.52
KEYVIEW-AWARE	90%	86%	1.19

Table 7: Statistics from the Experiment 3. Percentage of users finding the speed or the quality is acceptable for a given video, and the average ranking.

Preview	t Stat	t Critical	$P(T \leq t)$
REDUCE-QUALITY	4.418	2.086	0.000
REDUCE-SPEED	6.693	2.086	0.000

Table 8: Results of a two-tail t-Test on the Experiment 3.

There are many interesting new problems related to 3D mesh preview streaming. We give two examples here. One issue that we are looking at is the construction of streaming-friendly camera path. Ideally, the data rate as the view moves along the camera path should be as smooth as possible. It is unclear how one can find such a camera path, while balancing the smoothness of data rate and the camera path. Further, in complex objects, the chance that the camera path intersects with the model increases.

Another area of interest is the automatic determination of keyviews. A set of chosen keyviews is a natural and efficient way to indicate a continuous camera path. Currently, finding keyviews and finding camera path are two independent processes. It is, however, possible to refine both iteratively. A keyview can, for instance, be nudged if doing so would lead to a better camera path without sacrificing the importance of the area being viewed. Doing so would require analysis of the 3D model and some semantic/context information.

We note that, the work is a first approach, and can be extended to consider other cases (e.g., using artistic or dynamic camera paths). But a predefined camera path nails down the many degrees of 3D navigation freedom to one: the speed. Exploiting the camera speed is a first step and has already led to various proposed schemes. We'll take more degrees of freedom into consideration in future.

Acknowledgments

We thank the participants of our user studies for their participation, Jia Han Chiam for his implementation of the TSP solution, and the Stanford Computer Graphics Laboratory for making available the meshes we use in this research.

8. REFERENCES

[1] 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.

[2] G. Al-Regib, Y. Altunbasak, and J. Rossignac. Error-resilient transmission of 3D models. *ACM Transactions on Graphics*, 24(2):182–208, 2005.

[3] C. Andújar, P. Vázquez, and M. Fairén. Way-Finder : guided tours through complex walkthrough models. *Computer Graphics Forum*, 23(3):499–508, 2004.

[4] P. Burelli and G. N. Yannakakis. Towards adaptive virtual camera control in computer games. In *Proceedings of the*

11th International Symposium on Smart Graphics, pages 25–36, Bremen, Germany, 2011.

[5] N. Burtnyk, A. Khan, G. Fitzmaurice, and G. Kurtenbach. ShowMotion - camera motion based 3D design review. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 167–174, 2006.

[6] J. K. C. Yuksel, S. Schaefer. Parameterization and applications of Catmull-Rom curves. *Computer Aided Design*, 43:747–755, 2011.

[7] 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.

[8] W. Cheng, D. Liu, and W. T. Ooi. Peer-assisted view-dependent progressive mesh streaming. In *Proceedings of the 17th ACM International Conference on Multimedia*, MM '09, pages 441–450, Beijing, China, 2009.

[9] W. Cheng and W. T. Ooi. Receiver-driven view-dependent streaming of progressive mesh. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '08, Brauschweig, Germany, May 2008.

[10] W. Cheng, W. T. Ooi, S. Mondet, R. Grigoras, and G. Morin. An analytical model for progressive mesh streaming. In *Proceedings of the 15th ACM International Conference on Multimedia*, MM '07, pages 737–746, Augsberg, Germany, September 2007.

[11] R. N. De Silva, W. Cheng, D. Liu, W. T. Ooi, and S. Zhao. Towards characterizing user interaction with progressively transmitted 3D meshes. In *Proceedings of the 17th ACM International Conference on Multimedia*, MM '09, pages 881–884, Beijing, China, 2009.

[12] R. N. De Silva, W. Cheng, W. T. Ooi, and S. Zhao. Towards understanding user tolerance to network latency and data rate in remote viewing of progressive meshes. In *Proceedings of the 20th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '10, pages 123–128, Amsterdam, The Netherlands, 2010.

[13] H. Dutagaci, C. P. Cheung, and A. Godil. A benchmark for best view selection of 3D objects. In *Proceedings of the ACM Workshop on 3D Object Retrieval*, 3DOR '10, 2010.

[14] M. Feixas, M. Sbert, and F. Gonzalez. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception (TAP)*, 6(1), Feb. 2009.

[15] S.-R. Han, T. Yamasaki, and K. Aizawa. Automatic preview video generation for mesh sequences. In *Proceedings of the 17th IEEE International Conference on Image Processing*, ICIP '10, pages 2945–2948, Sept. 2010.

[16] 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 Systems Support for Digital Audio and Video*, NOSSDAV '02, pages 43–49, Miami, FL, May 2002.

[17] H. Hoppe. Efficient implementation of progressive meshes. *Computers Graphics*, 22(1):27–36, January-February 1998.

[18] T. Kamada and S. Kawai. A simple method for computing general position in displaying three-dimensional objects. *Computer Vision, Graphics and Image Processing*, 41(1), Jan. 1988.

[19] A. Khan, B. Komalo, J. Stam, G. Fitzmaurice, and G. Kurtenbach. HoverCam: Interactive 3D navigation for proximal object inspection. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, volume 1, pages 73–80, Washington, DC, 2005.

[20] H. Li, M. Li, and B. Prabhakaran. Middleware for streaming 3D progressive meshes over lossy networks. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2(4):282–317, 2006.

[21] C. Lino, M. Christie, F. Lamarche, G. Schofield, and P. Olivier. A real-time cinematography system for interactive 3D environments. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 139–148, 2010.

[22] S.-B. Park, C.-S. Kim, and S.-U. Lee. Error resilient coding of 3D meshes. In *Proceedings of IEEE International Conference on Image Processing*, volume 1 of *ICIP '03*, pages I–773–6, 2003.

[23] 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.

[24] A. Picardi, P. Burelli, and G. N. Yannakakis. Modelling virtual camera behaviour through player gaze. In *International Conference On The Foundations Of Digital Games*, Bordeux, France, June 2011.

[25] D. Plemenos and M. Benayada. Intelligent display in scene modeling. new techniques to automatically compute good views. In *Proceedings of the International Conference on Computer Graphics & Vision*, 1996.

[26] R. Ranon, M. Christie, and T. Urli. Accurately measuring the satisfaction of visual properties in virtual camera control. In *Proceedings of the 10th International Symposium on Smart Graphics*, pages 91–102, Banff, Canada, June 2010.

[27] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on Computing*, 6(3):564–581, 1977.

[28] D. Sokolov and D. Plemenos. Virtual world explorations by using topological and semantic knowledge. *The Visual Computer*, 24(3):173–185, Nov. 2007.

[29] Z. Tang, X. Guo, and B. Prabhakaran. Receiver-based loss tolerance method for 3D progressive streaming. *Multimedia Tools and Applications*, 51:779–799, 2011.

[30] D. Tian and G. AlRegib. On-demand transmission of 3D models over lossy networks. *EURASIP Journal on Signal Processing: Image Communication*, 21(5), June 2006.

[31] P.-P. Vázquez. Automatic view selection through depth-based view stability analysis. *The Visual Computer*, 25(5-7):441–449, Mar. 2009.

[32] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Viewpoint selection using viewpoint entropy. In *Proceedings of the Vision, Modeling, and Visualization Conference*, pages 273–280, Stuttgart, Germany, 2001.

[33] I. Viola, M. Feixas, M. Sbert, and M. E. Gröller. Importance-driven focus of attention. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):933–40, 2006.

[34] 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.

[35] G. N. Yannakakis, H. P. Martínez, and A. Jhala. Towards affective camera control in games. *User Modeling and User-Adapted Interaction*, 20(4):313–340, 2010.

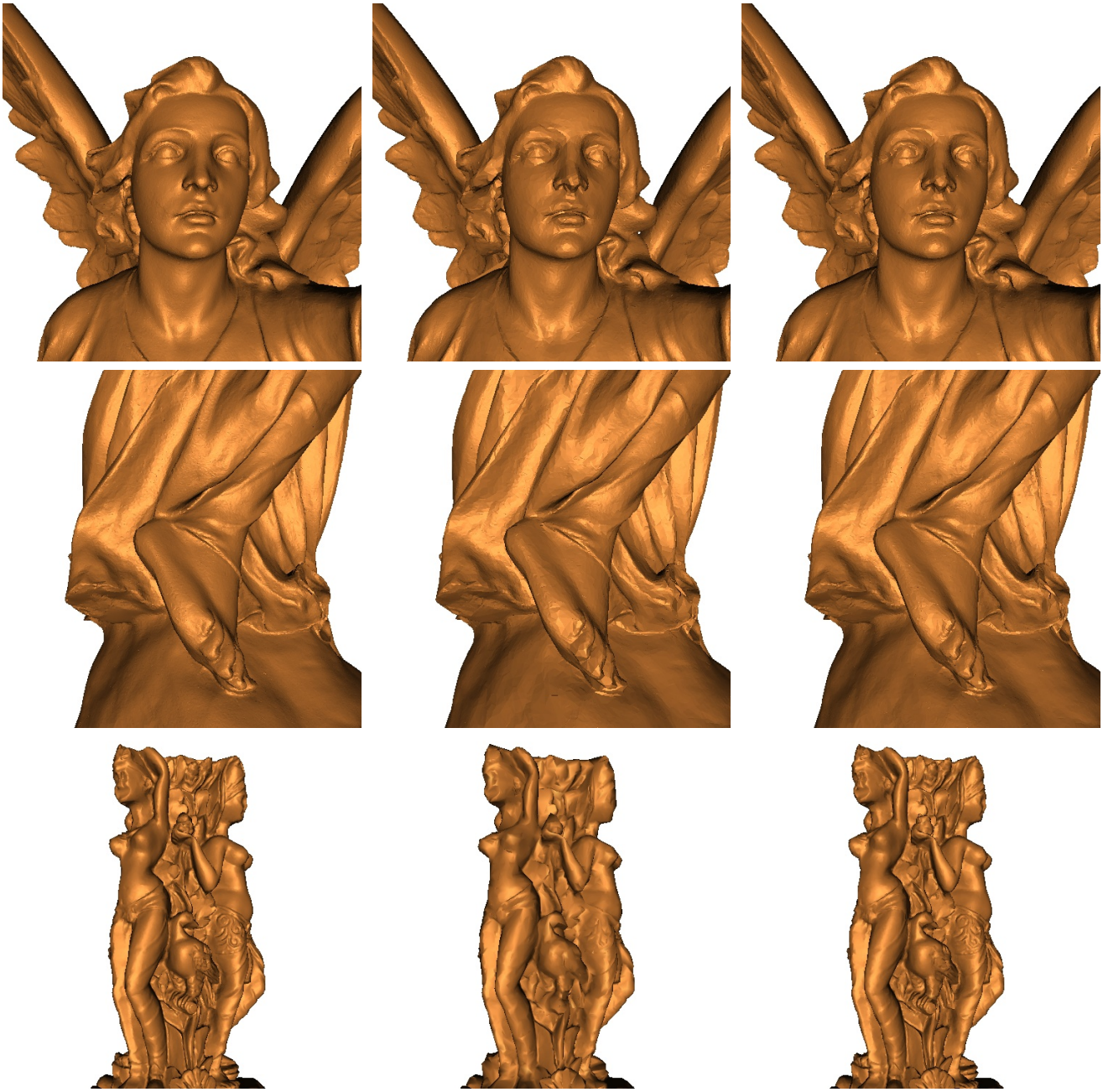


Figure 6: Mesh quality of selected views: left: REDUCE-SPEED, middle: REDUCE-QUALITY, right: KEYVIEW-AWARE.