Bandwidth Adaptation for 3D Mesh Preview Streaming

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

Bandwidth Adaptation for 3D Mesh Preview Streaming

SHANGHONG ZHAO and WEI TSANG OOI, National University of Singapore
AXEL CARLIER, GERALDINE MORIN, and VINCENT CHARVILLAT, University of Toulouse

Online galleries of 3D models 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 objects: by streaming and showing parts of the mesh surfaces visible along the virtual camera path. This article focuses on the preview streaming architecture and framework and presents our investigation into how such a system would best handle network congestion effectively. We present 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 two advanced methods: (d) KEYVIEW-AWARE, which trades off mesh quality and camera speed appropriately depending on how close the current view is to the keyviews, and (e) ADAPTIVE-ZOOM, which improves visual quality by moving the virtual camera away from the original path. A user study reveals that our KEYVIEW-AWARE method is preferred over the basic methods. Moreover, the ADAPTIVE-ZOOM scheme compares favorably to the KEYVIEW-AWARE method, showing that path adaptation is a viable approach to handling bandwidth variation.

Categories and Subject Descriptors: I.3.2 [**Computer Graphics**]: Graphics Systems—*Distributed/network graphics*

General Terms: Algorithms, Human Factors, Performance

Additional Key Words and Phrases: Progressive mesh streaming, 3D preview, camera path

1. INTRODUCTION

The ease in 3D acquisition, editing, viewing, and printing have led to websites that support online publishing of 3D models. Individual users can now share 3D models on websites, such as p3d.in[1] or Sketchfab.[2] Organizations are also publishing 3D scans of cultural artifacts through websites, such as 3D-COFORM.[3]

[1]`http://p3d.in.`
[2]`http://shw.gl.`
[3]`http://www.3d-coform.eu.`

Authors' addresses: S. Zhao (corresponding author) and W. T. Ooi, National University of Singapore; email: {shzhao17, ooiwt}@comp.nus.edu.sg; A. Carlier, G. Morin, and V. Charvillat, University of Toulouse; email: axel.carlier@enseeiht.fr; {morin, charvi}@n7.fr.

These websites typically allow users to preview a 3D model so that a user can decide whether to actually download the 3D content. Such a preview is normally achieved through either (i) displaying thumbnail images of the 3D model, taken from representative views (or *keyviews*), or (ii) providing a video depicting the 3D model from a camera that moves along a *camera path*, which connects through the keyviews. Image-based preview requires lower transmission bandwidth but conveys a limited amount of information about the model. On the other hand, video-based preview requires a higher transmission bandwidth, but it allows a smooth transition between the keyviews, conveying the overall 3D shape.

We previously proposed a third approach to preview a 3D model, called *3D preview streaming* [Zhao et al. 2013] that is designed for previewing large 3D models encoded as progressive meshes. 3D preview streaming is similar to video-based preview in that it displays the 3D model at the client as viewed from a moving camera that navigates along a predefined camera path, connecting the keyviews. Unlike video-based previewing, however, 3D preview streaming uses view-dependent progressive mesh streaming to transmit 3D content and renders, locally for the user, parts of the 3D model that are visible from the current viewpoint.

The advantages of 3D preview streaming over video-based preview are as follows. First, if the user decides to interact with and view the 3D model after watching the preview, the 3D model information has already been (partially) downloaded and can be reused. Whereas for video-based preview, the video data downloaded is useless for 3D rendering and interaction. Second, since the user visualizes 3D data, he can choose to preview the model under different rendering parameters (e.g., lighting condition). The video-based preview approach would require a new video every time a rendering parameter is changed. Third, depending on the 3D model's geometric complexity, 3D preview streaming may require lower bandwidth than a high-quality video-based preview. Finally, progressive representation of 3D models allows for quality adaptation to available network bandwidth and 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, leading to lower perceptual quality of the 3D model.

3D mesh preview streaming has many similarities with video streaming, which allows many techniques of video streaming to be reused: (i) users watch a streamed media content (mostly) passively; (ii) the media content changes with time, and a playback deadline is associated with each element of the media; (iii) a start-up delay for buffering can ensure a smooth playback; and (iv) the quality of the media can be lowered to reduce the data rate.

There are, however, several intrinsic and interesting differences between 3D mesh preview streaming and video streaming. First, the data rate of a video is coupled with its display rate (i.e., frame rate). While reducing a video stream's frame rate can lower its transmission rate, it also lowers the perceptual quality if the frame rate reduces beyond a threshold [Chen and Thropp 2007]. As for 3D mesh preview streaming, the data rate is decoupled from its display rate—the display rate is controlled by the rendering rate, whereas the data rate is controlled by the speed of moving camera. Thus, we can reduce the camera speed to reduce the data rate, while keeping the rendering rate constant.

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

Third, in video streaming, the rate control decision can be done independently for a frame or a group of frames. For 3D mesh preview streaming, one triangle can be visible from an arbitarily long sequence of viewpoints. Thus, if not sent, this triangle will still need to be transmitted for an arbitarily long time, leading to nontrivial rate control algorithms.

Fourth, by adjusting the camera path, the system can dynamically trade off between the quality of rendered views and the amount of transmitted data, adding another dimension to bandwidth adaptation.

These characteristics of 3D mesh preview streaming lead to new techniques and challenges for bandwidth adaptation and buffer control. In our previous work [Zhao et al. 2013], we have presented three basic techniques that mirrored the common techniques used in video streaming. 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. The second technique, REDUCE-SPEED, slows down the camera movement when the sending rate reduces and resumes normal speed when enough data is buffered. This technique is analogous to reducing the video frame rate, but as already explained, is able to maintain a smooth display rate due to decoupling of rendering rate and data rate. The third technique, REDUCE-QUALITY, lowers the quality of the mesh (by sending fewer refinements of progressive meshes) when the sending rate reduces. This technique is analogous to sending fewer layers in a scalable video but with a reduction of quality in the 3D domain.

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

We have also proposed a hybrid strategy that simultaneously reduces the camera speed and reduces the mesh quality. This 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. We call this scheme KEYVIEW-AWARE. User experiments have shown that KEYVIEW-AWARE is preferred by most users, compared to the naive strategies.

In this article, we further propose an alternative technique for bandwidth adaptation. Exploiting the fact that changing the camera path changes the perceptual quality of the mesh and the amount of data that needs to be transmitted, the client modifies the camera path depending on the amount of buffers available, zooming out when the buffer fill level is low and keeping to the original path otherwise. This last technique is called ADAPTIVE-ZOOM.

The rest of the article is organized as follows. We first present previous work related to ours in Section 2. Section 3 introduces the background and the basic algorithms of our proposed framework. The bandwidth adaptation schemes are presented in Section 4. We present the evaluation of our schemes in Section 5, and finally we conclude in 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 allowed. Meeting playback deadline is not a central issue in these works but is a core consideration of our approach. The concerns in previous work 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 [2005], Li et al. [2006], and Chen et al. [2005] investigated how to choose between different transport protocols for transmissions, trading reliability and delay. Harris III and

Kravets [2002] designed a new transport protocol that exploits partially-ordered property of 3D objects organized into trees of bounding volumes.

Several existing works also consider error control techniques at the application layer. Park et al. [2006] and Yan et al. [2001] segmented a progressive mesh into smaller partitions that are loss resilient. Al-Regib et al. [2005] and Chen et al. [2005] studied error correction for 3D transmission, while Park et al. [2003] and Tang et al. [2011] investigated error concealment methods. Cheng et al. [2007] and Tian and Al-Regib [2006] used retransmissions for error control. We do not consider error control in this article. 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 a situation is similar to video streaming and well-known solutions exist.

Cheng et al. [2009; Cheng and Ooi 2008] 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 [Cheng and Ooi 2008]. Second, a peer-assisted approach is used to reduce the bandwidth overhead at the server [Cheng et al. 2009]. We consider a client-server architecture in this work. The computational overhead of the server, however, is less, since the camera path is predetermined. 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. [2009, 2010] who studied 3D progressive mesh streaming from the user perspective. In particular, they measured how well users would tolerate slow mesh refinements and high response time [De Silva et al. 2010]. Their findings 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 it 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 [1988] chose nondegenerated views. Some recent work focused on maximizing a criterion based on the 3D geometry, like maximizing the number of polygons seen or the area covered [Plemenos and Benayada 1996], the total curvature [Sokolov and Plemenos 2007], the viewpoint entropy [Vázquez et al. 2001], or the view stability [Vázquez 2009]. Dutagaci et al. [2010] gave a nice comparison of state-of-the-art methods. Perceptual criteria have also been proposed: gathering user experience [Yannakakis et al. 2010] or tracking eye motion [Burelli and Yannakakis 2011; Picardi et al. 2011]. Some of these techniques generalize naturally to more than one view, such as the work of Feixas et al. [2009] which used the entropy of the same object.

Creating a virtual camera path is a common problem in robotics, gaming, or virtual walkthrough. The goals, however, are different. In the context of gaming and virtual walkthrough, the environment is usually a complex scene, and criteria like collision detection are main concerns, whereas visual quality may not be as important. Navigation of viewpoint in such complex worlds uses cinematographic rules [Ranon et al. 2010], dedicated language [Lino et al. 2010], or interactions for the camera control [Khan et al. 2005]. 3D preview streaming has a simplified framework: we start from a set of keyviews, selected by an expert for describing the 3D object, and smoothly link the keyviews with an automatic approach. Han et al. [2010] chose among the visited keyviews using a shortest-path algorithm. In our work, we require an ordering of the keyviews and consider a different cost function between the keyviews. While their cost function depends on the similarity of the keyviews (e.g., viewing angles), our cost function considers the streaming cost. Moreover, their camera path lacks smoothness, which leads to a shaky effect in the resulting videos. Both Burtnyk et al. [2006] and Andújar et al. [2004] used smooth paths,

respectively, modeled by Bézier or Hermite polynomial curves. Similarly, we use Catmull-Rom splines 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 [Hoppe 1998]. 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 meshes is done by transmitting over the base mesh, followed by a sequence of vertex splits. As the client receives the vertex splits, it updates and renders the mesh with increased levels of detail. As a result, the user sees the mesh refining itself.

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

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 virtual camera parameters) is predetermined in the form of a parametric camera path $P(\cdot)$, which is precomputed 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 virtual camera parametrized by $C(t)$. The set of parameters $C(t)$ specifies 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 such 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

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). Given these keyviews, we first determine the order in which the keyviews should be visited. We model the problem as a traveling salesman problem and find the order which maximizes the amount of shared data visible from two consecutive keyviews. We then find a Catmull-Rom spline curve interpolating the keyviews in the chosen order. The details can be found in our previous work [Zhao et al. 2013].

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 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, that is, the rate at 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 playback rate and 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 could happen for several reasons. First, there could be a congestion in the network, causing the server to clamp down on the congestion window, effectively reducing the value of r. Second, there could be new clients in the system, causing the server to reduce the value of r for one client to reallocate 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, 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 by answering the following question: what should the client (and the server) do when there is insufficient data to display the mesh at full resolution?

In the rest of this section and the next section, we review three basic techniques and a more sophisticated technique that is a hybrid of the three, first proposed in Zhao et al. [2013]. We then introduce a new method in Section 4.5.

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 thus 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.

These 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 of handling bandwidth variation. The method works as follows: let the sequence of keyviews along the camera path be $P(R_1), P(R_2), \ldots, P(R_k)$, 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 starting to 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 these three methods STOP-AND-WAIT and call the variations "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 meshes are 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.

This technique is similar to streaming fewer layers in a scalable video but having a reduced quality in the 3D domain. Note that, at the end of the preview, the model will not be available at full resolution.

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 rendering viewpoint $P(t_{disp})$ and the latest complete viewpoint received is $P(t_{recv})$.Consider what happens when the sending rate reduces. 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.

Note that 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. On the other hand, REDUCE-QUALITY could end up showing only the base mesh, as little or no data is being received. Neither of which is desirable.

The basic schemes either optimize for speed (REDUCE-QUALITY) or mesh quality (REDUCE-SPEED). We observe that 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 more appropriate away from the keyviews, as reducing the quality of the mesh around non-interesting regions is more acceptable. This observation has led us to design a new keyview-aware scheme that combines these basic schemes.

4.4 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 $t = 0, 1, \ldots, N-2$, the server computes its camera speed $S_s(t)$ as the following.

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

With this 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 $S_c(t)$ similar to 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 small variations in the camera speed.

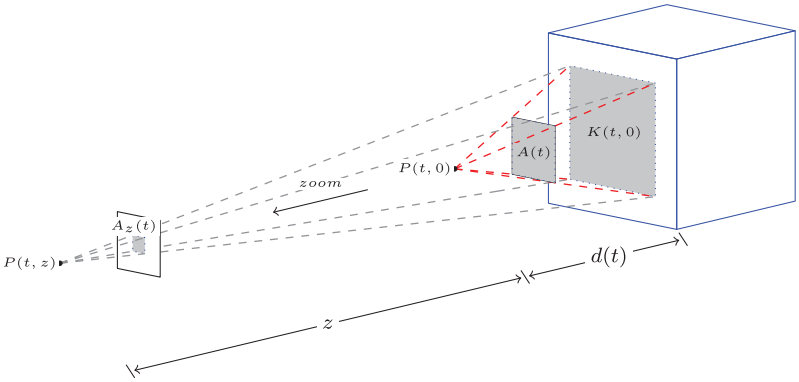


Fig. 1. Screen areas of a given region with different perspective projections: with the virtual camera on the original path $P(t,0)$ (in red), and the virtual camera moved to $P(t,z)$ (in grey).

This control, 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. 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

$$S_c(t) = \min\{k \times l_{gap}, S_{mid} I(t+1) + S_{max}(1 - I(t+1))\}. \tag{2}$$

Note that the 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 the STOP-AND-WAIT method.

4.5 Adapting Camera Zoom

Another interesting difference between 3D preview streaming and video streaming can be exploited: in 3D preview streaming, different camera paths lead to different bandwidth requirements and different view quality. Thus, modifying the camera path provides another flexible dimension in which we can perform bandwidth adaptation. Modifying the camera path, generally, however, is rather challenging—we would like to keep to the given keyviews and computed camera path as much as possible, since they are computed offline. In this article, we therefore consider adapting the camera path by simply zooming away from the 3D object.

Before we explain how we adapt to the available bandwidth, we first explain what we mean by "zooming away" and its effect on the view quality. In our setting, zooming away means moving the camera backwards under the perspective projection, keeping the other camera parameters constant (see Figure 1). Although a zoom-in viewpoint in the original camera path shows the mesh in more detail, a drop in bandwidth ($r(t) \ll B(t)$) can affect the quality of the mesh in the KEYVIEW-AWARE scheme, leading to visible flat faces in the rendered surface. In this case, details are missing, so the mesh is actually coarser than the necessary resolution. The original view reveals the piecewise linear structure of the mesh, reducing the perceptual quality for the user. When the camera zooms out, thanks to the perspective projection, the screen area of the mesh decreases, the unit screen area (pixels per polygon) also decreases, refining the view resolution of the mesh surface and retrieving the smoothness of the object.

These observations led us to design an extension of KEYVIEW-AWARE that zooms out the camera to maintain the view resolution, especially under the network condition $r(t) \ll B(t)$. We call this extension ADAPTIVE-ZOOM.

4.5.1 *Camera Zoom.* Before we present the technical details of ADAPTIVE-ZOOM, we first illustrate how to adapt the camera trajectory with respect to the original path. Let $P(t)$ be a viewpoint in the original path. Figure 1 shows zooming the camera away at a chosen distance z, which moves the camera from its position on the original path $P(t)$ to $P(t, z)$, where $z \geq 0$. We set $P(t, 0) = P(t)$ and $P(t, z) = P(t, 0) + z\vec{v}$, where $\vec{v}$ is a unit vector of direction opposite to the view direction of $P(t)$.

Let $K(t, z)$ be the set of polygons visible from the position $P(t, z)$. We assume that $K(t, 0)$ is not occluded by $K(t, z)$. $A(t)$ is the screen area of the polygons in $K(t, 0)$ rendered with the virtual camera center at $P(t, 0)$, and $d(t)$ is their average depth. Similarly, $A_z(t)$ is the screen area of the polygons in $K(t, 0)$ rendered with the virtual camera at $P(t, z)$. Ideally, we have the following equation based on similar triangles.

$$\frac{A_z(t)}{A(t)} = \left(\frac{d(t)}{d(t) + z}\right)^2. \tag{3}$$

Fixing $A(t)$ and $d(t)$, the screen area $A_z(t)$ decreases as the zoom distance z increases. Thus, the unit screen area $A_z(t)/\#K(t, 0)$ also decreases, improving the view resolution of the polygons in $K(t, 0)$. However, $A_z(t)$ may decrease too much if z becomes large: it is undesirable to get a rendered object too small on the screen.

4.5.2 *Adaptive Zoom Path.* We now describe how to position the camera according to the screen area $A_z(t)$ and the unit screen area $A_z(t)/\#K(t, 0)$.

We first define a range of possible zoom distances, constrained by the reduction in the screen area. In other words, we limit the zoom distance to avoid reducing $A_z(t)$ by too much. Given a user-defined minimum ratio $(A_z/A)_{min}$ (we use 0.3 in our experiments), we can compute the maximum zoom distance $z_{max}(t)$ at time t based on Equation (3).

$$z_{max}(t) = \left(\frac{1}{\sqrt{(A_z/A)_{min}}} - 1\right) d(t). \tag{4}$$

Thus, at each position $P(t)$ in the original camera path, we can zoom out, up to $z_{max}(t)$ units away from the object.

To accommodate zooming out, we extend the notation for $M(t)$ to $M(t, z)$, denoting the set of vertices visible from any point along the curve between $P(t, z)$ and $P(t + 1, z)$ (inclusive) that have not been transmitted before. Similarly, we extend the notation for $B(t)$ (the size of $M(t)$ in bits) to $B(t, z)$.

With the range of zoom distance defined, the *server* algorithm for ADAPTIVE-ZOOM is as follows: for each $t = 0, 1, \ldots, N - 2$, the server sends the set of vertices $M(t, z)$, for $z = z_{max}(t), \ldots, 0$ in decreasing order of z, until either all vertices are sent or the time is up. Similar to KEYVIEW-AWARE, the server's camera adapts its speed based on Equation (1). Let $z(t)$ be the minimum zoom distance such that the server has transmitted $M(t, z(t))$. Before the server moves its camera to the next point in the camera path, the server notifies the client of the viewpoint $P(t, z(t))$. Therefore, for each viewpoint $P(t)$, the minimum zoom distance $z(t)$ changes according to the bandwidth $r(t)$.

As in KEYVIEW-AWARE, we set a maximum screen area per polygon to a_{max} pixels to avoid transmitting unnoticeable vertices. Specifically, the server computes and transmits the data in $M(t, z)$ until the screen area of each polygon in the $K(t, z)$ is smaller than a_{max}. If $K(t, 0)$ is not occluded by $K(t, z)$, that is, $K(t, 0) \subseteq K(t, z)$, a_{max} is the upper bound of the average screen area $A_z(t)/\#K(t, 0)$. In our experiments, a_{max} is empirically set to be one pixel.

Table I. Measurements of 3D Mesh Preview Streaming along a Camera Path (Figure 2)

Measure	Thai Statue
Number of Vertices	5 million
Number of Triangles	10 million
Size of Transmitted Mesh	81.2 MB
Size of Base Mesh	10.3 KB
Number of Vertices in Base Mesh	16
Size per Vertex-Split	17 bytes
Precision of Vertex Coordinates	17 bits
Render Frame rate	10–18 fps
Network Overhead	244 bytes

At the *client*'s side, the camera moves along the path defined by $P(t, z(t))$ as informed by the server, guaranteeing that the unit screen area of each view is smaller than a_{max}. Similar to KEYVIEW-AWARE, the client updates its camera speed based on Equation (2).

There are a few interesting properties of the ADAPTIVE-ZOOM scheme. First, $A_z(t)$ tends to decrease as the zoom distance z increases. In particular, when the whole mesh is in the frustum, increasing z decreases $A_z(t)$ (Equation (3)). Second, $B(t, z)$ tends to decrease, that is, fewer triangles need to be transmitted as the screen area $A_z(t)$ decreases. This is because (i) we set the maximum pixel threshold a_{max} so that the upper bound of the size of $M(t, z)$ would be $A_z(t)/a_{max}$; (ii) when the camera zooms out, it tends to view more vertices that have been transmitted before, reducing the size of $M(t, z)$ effectively. Third, if the user-defined ratio $(A_z/A)_{min}$ is set to 1, then the maximum zoom distance $z_{max}(t) \equiv 0$, and the ADAPTIVE-ZOOM scheme reduces to the KEYVIEW-AWARE scheme.

The proposed strategies, in particular the two advanced ones, are evaluated in the next section.

5. EVALUATION

In this section, we first summarize the evaluation results of the basic 3D preview streaming schemes proposed in Zhao et al. [2013], followed by an evaluation of ADAPTIVE-ZOOM and KEYVIEW-AWARE.

5.1 Experimental Setup

We used the Thai Statue model from the Stanford 3D Scanning Repository for evaluation in this article. We converted the model into progressive meshes for streaming. The vertex coordinates are encoded with a lossy compression algorithm [Cheng and Ooi 2008] and are of reduced size. We rendered each model on a 500×500 pixels canvas with OpenGL. Table I summarizes the mesh we used and some basic performance metrics of preview streaming. All measurements are done with a Linux PC equipped with Intel Core 2 Quad 2.83GHz CPU, 4GB memory, and NVIDIA GeForce G210 graphics card.

We picked ten keyviews for the Thai Statue (three of the keyviews are shown in Figure 3), and constructed the camera path as described in Zhao et al. [2013]. The final camera path is shown in Figure 2.

For evaluation, we also implemented a GROUND-TRUTH scheme which 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`, the Linux traffic controller. We tested the Thai Statue with the data rate $r(t)$ that starts from 50KBps and switches between 10KBps and 50KBps every 20 seconds. We use this range of values as it roughly matches the variation in bandwidth required to transmit the Thai Statue model.

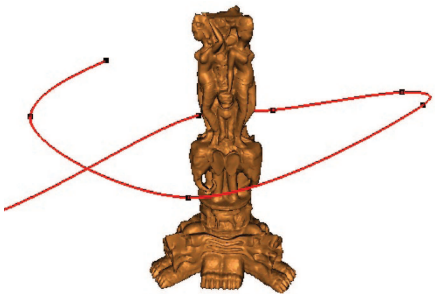


Fig. 2. The original camera path for the Thai Statue. The black points are the camera positions corresponding to keyviews.

Fig. 3. Three of the chosen keyviews for the Thai Statue.

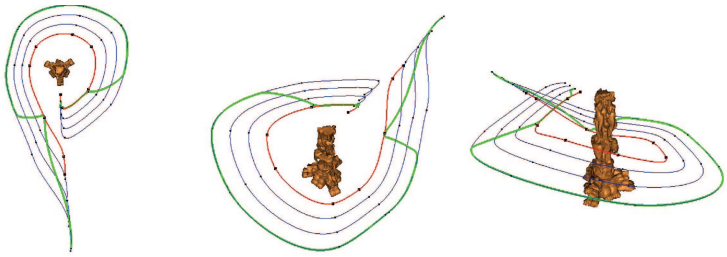
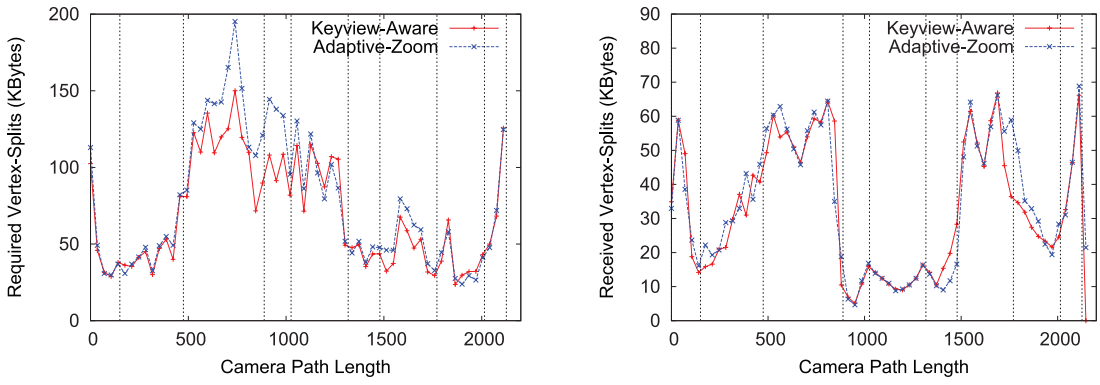


Fig. 4. Zoom paths for the Thai Statue. The original path (in red), three zoom paths (in blue) for different z, and the adaptive zoom path (in green).

5.2 Quantitative Results

Before we present the user study, we first illustrate the proposed schemes in greater detail. We present, in this section, the quantitative results from the ADAPTIVE-ZOOM scheme and compare it against KEYVIEW-AWARE. The results of the basic schemes and a more detailed analysis of the KEYVIEW-AWARE scheme can be found in Zhao et al. [2013].

5.2.1 *Zoom Paths.* Figure 4 illustrates the camera paths of ADAPTIVE-ZOOM at different zoom distances for previewing the Thai Statue. The three figures depict the same set of camera paths viewed from different angles. Each path is a sequence of the viewpoints $P(t, z(t))$ that are characterized by the zoom distance function $z(t)$ with the maximum value $z_{max}(t)$, where $t = 0, 1, \ldots, N-2$. The red curve is the original camera path with $z(t) \equiv 0$, while the blue curves are the intermediate zoom paths corresponding to $z(t) \equiv \alpha \cdot z_{max}(t)$, where $\alpha = \frac{1}{3}, \frac{2}{3}, 1$. The green curve is the adaptive zoom path recorded on the client side, with the distribution of the $z(t)$ shown in Figure 6(c).



(a) Required data size vs. t_{recv}.

(b) Received data size vs. t_{recv}.

(c) Estimated bandwidth vs. t_{disp}.

(d) Client's camera speed vs. t_{disp}.

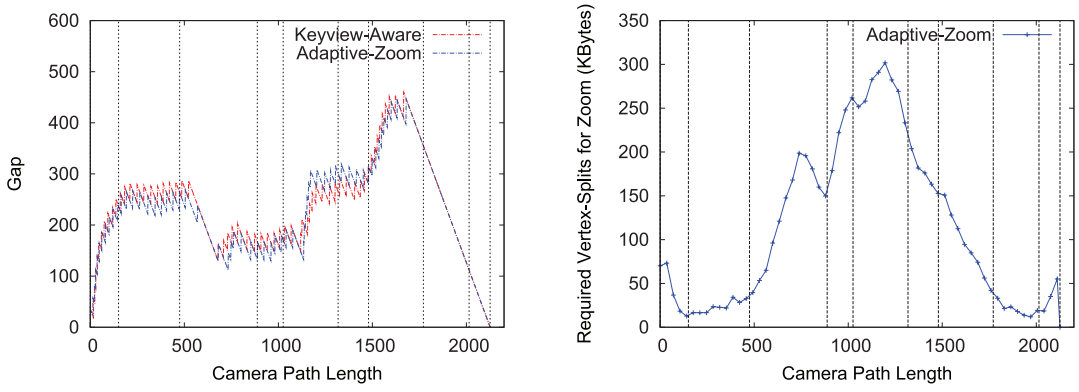
Fig. 5. Evolution of preview streaming schemes for the Thai Statue, recorded on the client side.

As we can see in Figure 4, the set of viewpoints $P(t, z(t))$ of the zoom path has a one-to-one correspondence to the set of viewpoints $P(t, 0)$ of the original path. KEYVIEW-AWARE uses the original path. To compare with KEYVIEW-AWARE, the parameters of ADAPTIVE-ZOOM at the $P(t, z(t))$ are shown against the $P(t, 0)$. Therefore, the x-axes of the graphs (Figures 5 and Figures 6), the gap (Figure 6(a)), and the client's camera speed (Figure 5(d)) are shown in OpenGL coordinate units of the length of the original camera path. The vertical lines indicate the positions of the keyviews.

5.2.2 *Evolution over Time.* Figures 5(a)–(d) and Figures 6(a)–(f) illustrate how KEYVIEW-AWARE and ADAPTIVE-ZOOM behave over time as bandwidth changes.

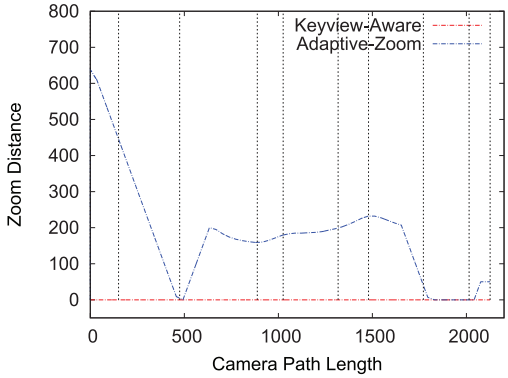
Figure 5(a) shows how much data the two schemes require in total, if the bandwidth is sustainable. ADAPTIVE-ZOOM has a higher demand for transmission bandwidth. As this strategy sends all vertices visible from a camera positioned at different zoom distances, more data may need to be sent, especially in scenarios where zooming away causes more of the 3D object to fall into the camera's view. In total, for the Thai Statue model, an extra amount of 600 KBytes is needed, which is 14.3% more than 4,200 KBytes of KEYVIEW-AWARE.

Figure 5(b) shows how much data the client receives at each t_{recv} for both schemes. We see three dips in the figure. The first (around $t_{recv} = 200$) and third (around $t_{recv} = 2000$) dips are due to the lack of data to be sent at the corresponding viewpoints. The second (t_{recv} between 1000 and 1500) dip is due to

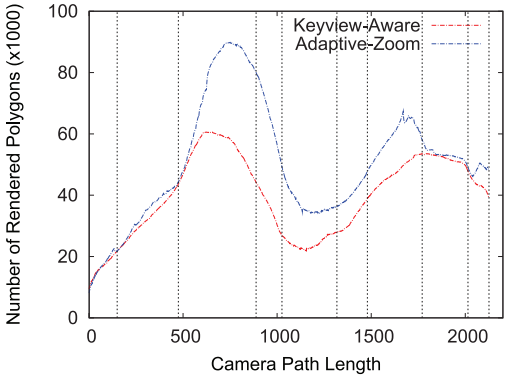


(a) Gap between receiving-rendering viewpoints vs. t_{disp}.

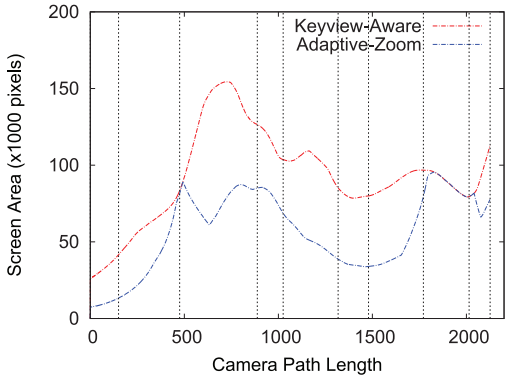
(b) Required data size for camera zoom vs. t_{recv}.



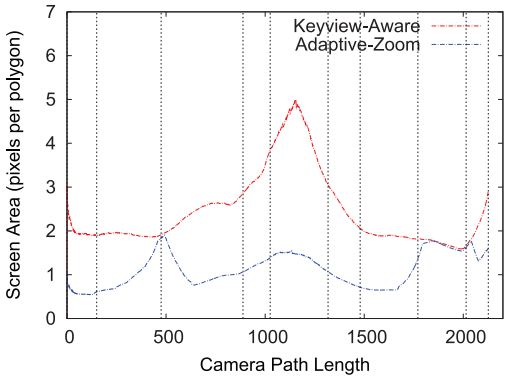
(c) Zoom distance vs. t_{disp}.



(d) Number of rendered polygons vs. t_{disp}.



(e) Screen area vs. t_{disp}.



(f) Average screen area vs. t_{disp}.

Fig. 6. Evolution of preview streaming schemes for the Thai Statue, recorded on the client side.

the decrease in bandwidth. We can observe that both schemes behave similarly in the amount of data received.

The next figure, Figure 5(c), shows the estimated bandwidth at the client, plotted against t_{disp}. The number of bytes received in the last two seconds is used to estimate the network bandwidth.

Figure 5(d) plots the camera speed at each t_{disp}. One can see that the camera speed adapts to the distance between the keyviews, slowing down near the keyviews, and achieving maximum speed in between two keyviews. Figure 6(a) shows the gap between t_{recv} and t_{disp}, plotted against t_{disp}. Note that the gap drops linearly to zero towards the end. Near the end of the session, no more data are being received, and the client is rendering the buffered vertices, allowing t_{disp} to catch up with t_{recv}.

So far, the observed behaviors of KEYVIEW-AWARE and ADAPTIVE-ZOOM are quite close. This result is expected, as ADAPTIVE-ZOOM is based on KEYVIEW-AWARE, sharing similar bandwidth adaptation of camera speed, gap, and received data size. The small divergences are usually caused by different estimated bandwidth (see Figure 5(c)).

The main difference between KEYVIEW-AWARE and ADAPTIVE-ZOOM is the camera position at the client. Even though the two schemes receive almost the same amount of data, ADAPTIVE-ZOOM renders the 3D mesh using a camera at a further distance away such that the average screen area per polygon is kept small, when the client is missing some details from the server. Figure 7 shows the difference between three pairs of simultaneous frames on KEYVIEW-AWARE and ADAPTIVE-ZOOM paths.

Figure 6(b) plots the amount of data that is still missing from the server, while using ADAPTIVE-ZOOM. Should the client received these missing vertex splits, it would have rendered the 3D mesh using the original camera path (i.e., $P(t, 0)$). We can see that as the estimated bandwidth starts to drop around $t_{disp} = 500$, the amount of missing vertex splits starts to increase. Note that after the estimated bandwidth has restored to the original level, it takes some time for the amount of missing vertex splits to drop below 100KB. This lag is due to sharing of vertices among the views along the camera path: having less data sent earlier means that more data needs to be sent later.

Figure 6(c) illustrates how the zoom distance $z(t)$ adapts to the bandwidth. When the bandwidth is sufficient (e.g., $t_{disp} \leq 500$ or $t_{disp} \geq 1500$ in Figure 5(c)), the amount of missing vertex splits starts to drop, allowing the virtual camera to smoothly zoom in, with the zoom distance $z(t) \rightarrow 0$ (Figure 6(c)). The zooming operation enlarges the screen area of the mesh quickly (Figure 6(e)), with a small increase in the average screen area per polygon (Figure 6(f)). As such, the client receives all the missing vertex splits, and the ADAPTIVE-ZOOM scheme reduces to the KEYVIEW-AWARE scheme. This effect is especially obvious around $t_{disp} = 500$ and between t_{disp} of 1750 and 2000 (Figures 6(c)–(f)). Note that, the small fluctuation of camera zoom distance after $t_{disp} = 2000$ is due to a sudden increase of the amount of required data (Figure 6(b)), due to missing vertex splits.

Moreover, corresponding to the amount of missing vertex splits, we can see that the zoom distance increases between t_{disp} of 500 and 1750. Zooming out, however, leads to more polygons being rendered for the ADAPTIVE-ZOOM scheme between this period (compared to KEYVIEW-AWARE), as shown in Figure 6(d). Within this time period, however, placing the camera further away from the 3D mesh leads to a smaller total screen area for the object (Figure 6(e)). As a result, the average screen area per polygon for the mesh (Figure 6(f)) is kept below two pixels. Note that during this time, KEYVIEW-AWARE continues with the original camera path, and with the drop in bandwidth, the lack of vertex splits being received results in a coarse mesh being rendered—the average screen area per polygon increases to as much as 5 pixels. The last figure (Figure 6(f)) clearly illustrates the advantage of ADAPTIVE-ZOOM.

5.3 User Study

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

In Zhao et al. [2013], we presented two sets of user study experiments, comparing KEYVIEW-AWARE with the basic schemes. We summarize the experimental results in the next section and present a user study comparing ADAPTIVE-ZOOM and KEYVIEW-AWARE in the next section.

5.3.1 *Basic Schemes vs.* KEYVIEW-AWARE. In the first set of user study, ten participants compare STOP-AND-WAIT schemes against the KEYVIEW-AWARE scheme. The Lucy model, obtained from the

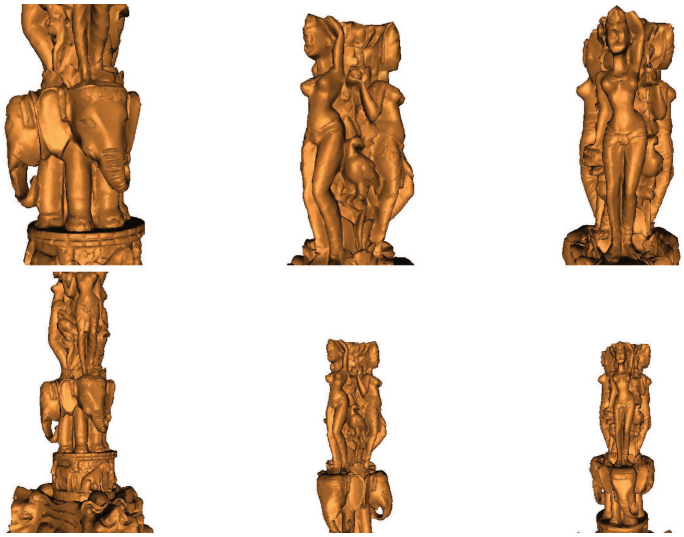


Fig. 7. View quality of selected views of the Thai Statue. Top: KEYVIEW-AWARE. Bottom: ADAPTIVE-ZOOM.

Stanford 3D Scanning Repository, is used. Only one participant finds the STOP-AND-WAIT strategy acceptable, regardless of whether the camera stops at a keyview or not. This study shows that, not surprisingly, STOP-AND-WAIT is not a good strategy.

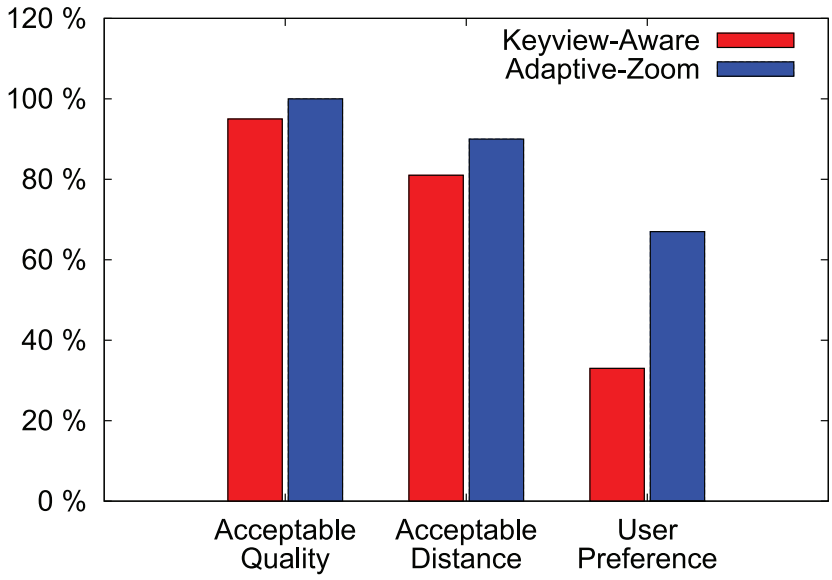
The second set of user studies compares REDUCE-QUALITY, REDUCE-SPEED, and KEYVIEW-AWARE, using both the Thai Statue model (21 participants) and the Lucy model (25 participants). The users are asked to rank the three strategies. The average ranking yields KEYVIEW-AWARE as the best, followed by REDUCE-QUALITY, and then REDUCE-SPEED. Qualitative comments by the participants indicate that, even though they are looking for a better mesh quality, the camera speed of REDUCE-SPEED makes it too boring to watch. For REDUCE-QUALITY, the participants are able to spot coarse mesh quality at certain regions of the models. The KEYVIEW-AWARE scheme is designated by the participants as the most convincing compromise between mesh quality and camera speed.

5.3.2 ADAPTIVE-ZOOM *vs.* KEYVIEW-AWARE. We now describe the new user study to compare between ADAPTIVE-ZOOM and KEYVIEW-AWARE scheme. The setup and methodology are similar to the experiments to compare KEYVIEW-AWARE and the three basic schemes. We present the details here for completeness.

Set-up. To allow the participants in our user study to experience these different strategies without installing our client, we stream the Thai Statue model using ADAPTIVE-ZOOM and KEYVIEW-AWARE strategies and encode the rendered frames as high-quality video clips, with each video clip corresponding to one preview strategy. The snapshots of selected views of each video clip are shown in Figure 7. We then set up a website for the participants to access, view, and rate these video clips. This website has two pages. On the first page, we present the GROUND-TRUTH 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 webpage, where we display two different previews that can be viewed independently and a form to collect participants' answers to questions. The two videos are displayed in a random order generated using a script. The answers are sent to and stored in a remote database. It takes between 10 to 20 minutes for a participant to complete the study, depending on the level of attention.

Table II. User Experience Levels with 3D Modeling

NO EXPERIENCE	NOVICE	AVERAGE	KNOWLEDGEABLE	EXPERT
9	6	3	0	3



(a) User study results.

Fig. 8. Percentages of users finding whether the quality or the distance is acceptable for a given video, and whether this video is preferable.

Form. For each preview of the second webpage, we ask each participant whether the quality and the distance to the mesh are acceptable. We do not provide any clue regarding what "acceptable" means, and let the participants judge according to GROUND-TRUTH. Then the participants are asked which preview they prefer. The participants can justify their choices and leave comments in a specific dialog box. Finally, we ask the participants to state their own age, gender, and experience with 3D modeling.

Participants. A total of 21 participants—4 females and 17 males—with age ranging from 22 to 28 and averaging 25 participated in the user study. Table II presents their experience levels. Figure 8 presents the acceptance rate and user preference of each video recorded with different bandwidth settings; 12 participants left comments.

Results. The user study shows that for a bandwidth setting of 50-10-50KBps, more users (67%) preferred ADAPTIVE-ZOOM than KEYVIEW-AWARE. While the comparison between the ADAPTIVE-ZOOM and KEYVIEW-AWARE schemes is not statistically significant, these advanced schemes both provide a better user experience than naive schemes. Moreover, the qualitative comments from the users have confirmed our understanding. One participant commented that he/she prefers a bigger distance between the camera and the mesh, while another commented that the camera of KEYVIEW-AWARE was too close to the mesh to maintain the quality. About the same number of users have found the quality and distance from the object acceptable for ADAPTIVE-ZOOM.

6. CONCLUSION

3D mesh preview streaming is a new form of continuous media transmission 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 in which 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. We also explore the possibility of adapting camera position to yield better (perceived) mesh quality by zooming out further away from the mesh when the quality is low. The user study shows that this adapted camera path improves the user experience.

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 a 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 the 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 have also shown that a simple edition of the path, here zooming away by moving the virtual camera, can help adapt the streaming to bandwidth variation. We could consider more degrees of freedom in the navigation in the 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.

REFERENCES

G. Al-Regib and Y. Altunbasak. 2005. 3TP: An application-layer protocol for streaming 3D models. *IEEE Trans. Multimedia* 7, 6, 1149–1156.

G. Al-Regib, Y. Altunbasak, and J. Rossignac. 2005. Error-resilient transmission of 3D models. *ACM Trans. Graph.* 24, 2, 182–208.

C. Andújar, P. Vázquez, and M. Fairén. 2004. Way-Finder: Guided tours through complex walkthrough models. *Comput. Graph. Forum* 23, 3, 499–508.

P. Burelli and G. N. Yannakakis 2011. Towards adaptive virtual camera control in computer games. In *Proceedings of the 11th International Symposium on Smart Graphics*. Lecture Notes in Computer Science, vol. 6815, Springer, Berlin Heidelberg, 25–36.

N. Burtnyk, A. Khan, G. Fitzmaurice, and G. Kurtenbach. 2006. ShowMotion - camera motion based 3D design review. In *Proceedings of the Symposium on Interactive 3D Graphics and Games (I3D'06)*. ACM, 167–174.

J. Chen and J. Thropp. 2007. Review of low frame rate effects on human performance. *IEEE Trans. Syst., Man Cybernet. Part A: Syst. Hum.* 37, 6, 1063–1076.

Z. Chen, J. F. Barnes, and B. Bodenheimer. 2005. Hybrid and forward error correction transmission techniques for unreliable transport of 3D geometry. *Multimedia Syst.* 10, 3, 230–244.

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

W. Cheng and W. T. Ooi. 2008. 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)*. ACM, 9–14.

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

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

R. N. De Silva, W. Cheng, W. T. Ooi, and S. Zhao. 2010. 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)*. ACM, 123–128.

H. Dutagaci, C. P. Cheung, and A. Godil. 2010. A benchmark for best view selection of 3D objects. In *Proceedings of the ACM Workshop on 3D Object Retrieval (3DOR'10)*. ACM, 45–50.

M. Feixas, M. Sbert, and F. Gonzalez. 2009. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Trans. App. Perception* 6, 1, 1:1–1:23.

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

A. F. Harris III and R. Kravets. 2002. 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)*. ACM, 43–49.

H. Hoppe. 1998. Efficient implementation of progressive meshes. *Comput. Graphics* 22, 1, 27–36.

T. Kamada and S. Kawai. 1988. A simple method for computing general position in displaying three-dimensional objects. *Comput. Vision, Graph. Image Proces.* 41, 1, 43–56.

A. Khan, B. Komalo, J. Stam, G. Fitzmaurice, and G. Kurtenbach. 2005. HoverCam: Interactive 3D navigation for proximal object inspection. In *Proceedings of the Symposium on Interactive 3D Graphics and Games (I3D'05)*. ACM, 73–80.

H. Li, M. Li, and B. Prabhakaran. 2006. Middleware for streaming 3D progressive meshes over lossy networks. *ACM Trans. Multimedia Comput. Commun. Appl.* 2, 4, 282–317.

C. Lino, M. Christie, F. Lamarche, G. Schofield, and P. Olivier. 2010. A real-time cinematography system for interactive 3D environments. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'10)*. Eurographics Association, 139–148.

S.-B. Park, C.-S. Kim, and S.-U. Lee. 2003. Error resilient coding of 3D meshes. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'03)*. 773–6.

S.-B. Park, C.-S. Kim, and S.-U. Lee. 2006. Error resilient 3-D mesh compression. *IEEE Trans. Multimedia* 8, 5, 885–895.

A. Picardi, P. Burelli, and G. N. Yannakakis. 2011. Modelling virtual camera behaviour through player gaze. In *Proceeding of the International Conference on the Foundations of Digital Games (FDG'11)*. ACM, 107–114.

D. Plemenos and M. Benayada. 1996. Intelligent display in scene modeling: New techniques to automatically compute good views. In *Proceedings of the International Conference on Computer Graphics and Vision*.

R. Ranon, M. Christie, and T. Urli. 2010. Accurately measuring the satisfaction of visual properties in virtual camera control. In *Proceedings of the 10th International Symposium on Smart Graphics (SG'10)*. Lecture Notes in Computer Science, vol. 6133, Springer-Verlag, Berlin Heidelberg, 91–102.

D. Sokolov and D. Plemenos. 2007. Virtual world explorations by using topological and semantic knowledge. *Visual Comput.* 24, 3, 173–185.

Z. Tang, X. Guo, and B. Prabhakaran. 2011. Receiver-based loss tolerance method for 3D progressive streaming. *Multimedia Tools Appl.* 51, 2, 779–799.

D. Tian and G. Al-Regib. 2006. On-demand transmission of 3D models over lossy networks. *Signal Proces. Image Commun.* 21, 5, 396–415.

P.-P. Vázquez. 2009. Automatic view selection through depth-based view stability analysis. *Visual Comput.* 25, 5–7, 441–449.

P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. 2001. Viewpoint selection using viewpoint entropy. In *Proceedings of the Vision, Modeling, and Visualization Conference*. Aka GmbH, 273–280.

Z. Yan, S. Kumar, and C.-C. Kuo. 2001. Error-resilient coding of 3-D graphic models via adaptive mesh segmentation. *IEEE Trans. Circuits Syst. Video Technol.* 11, 7, 860–873.

G. N. Yannakakis, H. P. Martínez, and A. Jhala. 2010. Towards affective camera control in games. *User Model. User-Adapt. Interact.* 20, 4, 313–340.

S. Zhao, W. T. Ooi, A. Carlier, G. Morin, and V. Charvillat. 2013. 3D mesh preview streaming. In *Proceedings of the 4th ACM Multimedia Systems Conference (MMSys'13)*. ACM, New York, NY, 178–189.