EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (*2001*), *Number 3*

Streaming of Complex 3D Scenes for Remote Walkthroughs

Eyal Teler Dani Lischinski

School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, Israel

Abstract

We describe a new 3D scene streaming approach for remote walkthroughs. In a remote walkthrough, a user on a client machine interactively navigates through a scene that resides on a remote server. Our approach allows a user to walk through a remote 3D scene, without ever having to download the entire scene from the server. Our algorithm achieves this by selectively transmitting only small parts of the scene and lower quality representations of objects, based on the user's viewing parameters and the available connection bandwidth. An online optimization algorithm selects which object representations to send, based on the integral of a benefit measure along the predicted path of movement. The rendering quality at the client depends on the available bandwidth, but practical navigation of the scene is possible even when bandwidth is low.

1. Introduction

During the past few years we have witnessed an explosive growth in the performance and capabilities of inexpensive 3D graphics accelerators. Consequently, state-of-the-art home PCs are now capable of interactively displaying fairly complex virtual 3D worlds. Concurrently, standards (such as VRML[3]) have emerged for describing the geometry and the behavior of 3D virtual worlds on the Internet, and applications and plug-ins capable of displaying online 3D content abound. Despite these developments, however, there are still disappointingly few sites on the WWW on which interesting 3D content, such as compelling and complex 3D scenes, can be found. Undoubtedly, one of the major reasons for that is the *latency problem*. Compelling 3D scenes contain a large number of object models, each with detailed geometry, as well as many textures. Such models take up large amounts of storage space, and take a long time to download from the server to a browser running on a remote client. As a result, the “download-and-play” paradigm used in today's VRML browsers is impractical when it comes to such 3D scenes.

In this paper, we describe a *3D scene streaming* approach that almost entirely eliminates latency in remote walkthroughs of complex static 3D scenes. (In a remote walkthrough the user/client interactively navigates through a 3D scene that resides on a remote server.) Data streaming solutions are commonly used for transmission of audio and video content over the Internet. However, in the case of audio and video the order in which data is played on the receiving end is known in advance. In contrast, when a user interactively navigates through a 3D world, no *a priori* transmission order can be determined. Instead, our approach utilizes the fact that the user at the client side typically sees only a very limited view of the world at any given time, a view that changes in a continuous manner. The server is notified of any changes in the velocity of the user's virtual camera, and decides what to transmit based on that information.

The streaming server regards the scene as a collection of 3D objects, each of which may have one or more representations. For example, an object can be represented as a progressive mesh, a precomputed collection of models at different levels of detail, or a view-dependent image-based impostor that is generated on demand. The goal of the server is to select a transmission sequence of object representations that will provide the highest rendering quality throughout the walkthrough, subject to the limitations imposed by the available bandwidth. This transmission schedule is determined by an online optimization algorithm that decides which representation to send at any given moment, based on the image quality improvement predicted for the rest of the walkthrough.

Our approach utilizes several techniques that were originally used for acceleration of interactive *local* walkthroughs of complex 3D scenes. However, it significantly departs from local rendering frameworks in the nature of the optimization that is taking place. In the local walkthrough context the performance bottleneck is the rendering engine, and existing approaches attempt to maximize the quality of each frame, while maintaining a certain frame rate. Each frame is considered separately and almost independently of the others. In our context, the bottleneck is the connection bandwidth between the server and the client; we assume that the client is able to interactively render whatever the server has managed to transmit so far. Any object that has already been transmitted to the client can be rendered from that point on without consuming further bandwidth. Thus, our optimization algorithm ignores frame rates completely, and is concerned with maximizing quality over time. Our results show that in a variety of bandwidth conditions, a user can start traversing a scene almost immediately. The quality of the rendering depends on the available bandwidth, of course, but practical navigation of the scene is possible even when bandwidth is low.

It should be noted that this paper describes work in progress. The main contribution of this paper is a new general formulation for the problem of streaming a scene across a limited bandwidth connection, which uses the novel concept of cumulative benefit integral. In particular, our formulation supports both geometric and image-based impostors in a single online optimization framework. The actual streaming system described in this paper served as a testbed for our ideas; it is not yet ready for real world applications, since our current implementation makes several simplifying assumptions.

2. Background

2.1. Acceleration of local rendering

Interactive navigation through complex 3D worlds requires the ability to render the scene at an acceptable frame rate, while keeping the image quality as high as possible. Over the years, quite a few effective techniques for accelerated rendering of complex objects and scenes have been developed, most of which can be easily incorporated into our framework. One class of acceleration techniques are *visibility culling* algorithms, which attempt to avoid rendering objects that cannot be visible in the image[1, 11, 26, 29]. Another approach is to use *level-of-detail* (LOD) models of objects in the scene[9] and/or image-based *impostors* (e.g., texture mapping the image of a complex object onto some simple geometry)[2, 17, 20, 23].

Accelerated rendering of complex objects and scenes can also be achieved by pure image-based rendering[4, 18] (IBR) and light-field rendering[10, 15], where an object/scene is represented entirely as a collection of images, without any kind of explicit geometric model. An IBR-based remote rendering system has recently been described by Yoon and Neumann[28]. However, it appears that pure image-based rendering and light-field rendering are not readily applicable in the context of remote walkthroughs of complex scenes, since the size of the representation can still be quite large, and thus the latency problem remains. Also, it is not at all obvious how to extend these representations in order to allow dynamic scenes.

Most of the other local rendering acceleration techniques, however, can be easily incorporated into remote walkthroughs. For example, if the server knows (or can estimate) the viewing parameters of the virtual camera at each point in time, various visibility culling approaches can be utilized by letting the server perform the culling. Culled objects need not be transmitted to the client, and the resulting available bandwidth can be spent on transmitting more information about those objects that are visible. Similarly, when a complex object is far away from the virtual camera, the server need not transmit the full model of the object. Instead, a coarser geometric model, or any other kind of impostor can be transmitted. Again, the saved bandwidth can be better spent on objects that are nearer to the virtual camera. In our system, so far we have implemented hierarchical frustum culling[5], simple LODs, and image-based impostors *a la* Shade *et al.*[23].

2.2. Benefit/cost optimization

Funkhouser and Sequin[9] (F&S) describe a *predictive approach* to local rendering. Based on measured performance parameters of the rendering engine, they predict how much geometry can be rendered in a frame's time. Heuristics are used to define a benefit for each LOD of each object, and constrained optimization is used to select the most beneficial LODs (including "object not rendered") for the estimated rendering budget. Maciel and Shirley[17] (M&S) extended this predictive approach to consider entire clusters of objects, and introduced ways to simplify objects, other than geometric LOD models.

The predictive approach is not readily applicable to 3D scene streaming. In the remote rendering paradigm, the bottleneck is not the client's rendering rate, but rather the rate at which object representations arrive at the client. Therefore a notion of optimizing for a "frame" or another client-related period of time is not useful. The client can always use techniques such as the predictive approach to accelerate local rendering, but rendering quality will still only be as high as the data that has arrived from the server so far. In remote rendering data is transmitted continuously and is not synchronized with the rendering of the frames. Network latency, and the amount of time it takes to transmit an object representation, mean that the server cannot optimize for the current frame the client is rendering. The server does not know the exact viewing parameters at the client side, and must estimate them. As a result, it must optimize for the future, instead of for a particular moment in time. A further difference is that once a representation has been transmitted, it can be reused by the client for the remainder of the walkthrough. For example, if the full geometry of the object has been sent, it need never be sent again. This is in contrast to local rendering, where the object needs to be considered as part of the "rendering budget" of every frame.

2.3. Graphics over network

Distributed virtual environments have attracted a considerable amount of research attention during the past decade[8, 16, 24]. However, most of the research in this area is concerned with efficient message passing and management of multiple interacting dynamic users, and does not address the problems of streaming large numbers of geometrically complex object models.

The F&S approach has been extended to the context of remote rendering by Hesina and Schmalstieg[12, 21]. The latter approach is based on continuous LODs with the server attempting to transmit to the client all of the objects within a circular area of interest around the current viewing position. Our approach is somewhat more general in that it supports both geometric and image-based object representations. We also present a new online optimization framework for remote rendering, which uses path prediction and a cumulative benefit function in order to more efficiently exploit the available bandwidth.

Schneider and Martin[22] describe a network graphics framework, where an appropriate representaion for a transmitted object is selected based on the available bandwidth and the rendering capabilities of the client machine. However, they focus on transmission of individual 3D models, rather than on interactive walkthroughs of complex virtual worlds containing many different objects.

2.4. Compression of 3D models

Transferring 3D objects over the Internet has been a subject of considerable academic as well as commercial interest for a while. To reduce bandwidth requirements a variety of *geometry compression* schemes have been devised[7, 25, 27]. These methods are capable of lowering the network bandwidth requirements down to 10 bits per vertex, on average (including coordinates and connectivity). However, even when geometry compression is being used, it is still wasteful to transmit the entire scene across the network, since it contains objects that might never be seen in the walkthrough, or at least never be seen in detail. From our standpoint, compression is equivalent to an increase in bandwidth. It is easy to add compression to any algorithm that optimizes network transfers, such as ours, thus effectively increasing the available bandwidth.

Another representation geared at transmission of 3D objects is *progressive meshes*, introduced by Hoppe[13]. Progressive meshes provide a semi-continuous refinement of an object, that provides a very rough shape of the object with a small amount of data, and can refine it by transmitting further data. Refinement can be view dependent, i.e., take into account what the user is viewing[14].

Progressive meshes are very effective for transmission of individual complex objects. Note however, that they do not constitute a complete solution for remote walkthroughs, since they are applied to each object separately. Although they are not currently implemented in our system, nothing in our approach prevents their incorporation. On the contrary, they fit nicely into our cumulative benefit concept, and we plan to add them to our system in the near future.

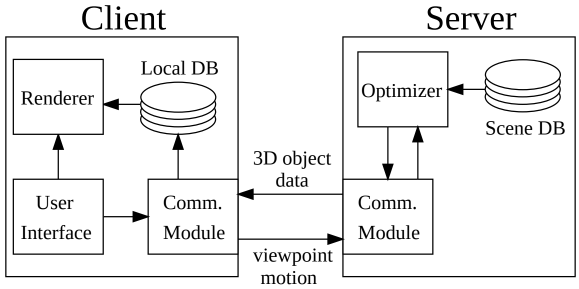


Figure 1: *A client-server architecture for remote walkthroughs*

Finally, Cohen-Or *et al.*[6] describe a compression technique that is well-suited for streaming of non-interactive walkthroughs of 3D scenes, where the user follows a predefined path through the 3D scene. In contrast, our work is geared towards interactive walkthroughs.

3. Overview

The architecture of our remote walkthrough system is shown in Figure 1. The full scene description is initially stored in a remote scene database on the server. The client periodically transmits to the server the user's current viewing parameters, including viewpoint velocity and acceleration. The server performs motion prediction and decides which representations of which objects to transmit to the client. The client stores all of the object representations it receives in its local database. For each viewpoint, the representations that provide the best rendering quality are selected, among those available in the local database, and rendered.

Assuming that the client has a rendering engine powerful enough to render received object representations interactively (applying local rendering acceleration schemes as necessary), the main bottleneck of our system is the connection bandwidth. Since transferring an entire complex scene takes too long, the user starts traversing the scene as soon as information begins to arrive from the server. The server's goal is to select which parts of the scene to send, such that the frames rendered at the client side will look as similar as possible to frames that would have been rendered had the entire model been available to the client. The selection is done using an online optimization algorithm.

The scene description consists of a collection of objects. Each of these objects has a set of representations associated with it. In principle, these representations can include the full geometric model of the object, several LODs or a progressive mesh representation, dynamically generated or precomputed image-based impostors, and any other conceivable representation. The representations may be a static part of the object database, such as a progressive mesh or LODs, or may be created during the walkthrough, as in the case of dynamically generated image-based impostors. Our current implementation uses dynamically generated image-based impostors as well as predefined LODs.

Each object representation has an associated cost, which is its expected transmission duration. It also has an associated view-dependent benefit measure that can be calculated for any viewpoint, which corresponds to the contribution of the object to the total visible quality of the scene. This measure reflects the accuracy of the representation with respect to a given virtual camera, the visibility of the object, and its importance — similarly to the benefit measure used by Funkhouser and Sequin. Our particular cost and benefit measures are discussed in more detail in the next section.

The online optimization algorithm is based on a simple greedy optimization strategy. The algorithm computes an added benefit integral for all relevant representations (of objects that are predicted to become visible), and transmits the representation with the best benefit to cost ratio. When the server finishes sending the representation, the selection process starts over. The computation of the added benefit integral is discussed in Section 6.

4. Representation cost and benefit

In the remote walkthrough context, the cost associated with a particular object representation can be estimated as the size of the representation divided by the available bandwidth, plus some constant overhead cost. In practice, the effective connection bandwidth varies according to network load. Thus, the average bandwidth over a recent time window should be used in the cost estimation.

The size of the representation is easy to compute for predefined object representations, such as pre-generated LODs. However, in the case of dynamically generated representations, such as view-dependent image-based impostors, the exact size of the representation is not known in advance, especially if they are to be compressed before transmission. In such a case, the representation size can be estimated using a table specifying a typical compressed size for image-based impostors of different sizes.

The definition of representation benefit is similar to the one used by F&S and M&S. It is defined as a product of several terms:

Accuracy A measure of how well the representation approximates the appearance of the full object rendered from the same point of view. Unlike M&S, we don't assume that accuracy is static. For example, an image-based impostor is considered 100 percent accurate when viewed from the point of creation, but its accuracy diminishes away from that point.

Visibility A measure of how clearly the object is seen, and how much of it is seen. The size of the object is an important factor for visibility. Other factors include occlusion, how much of the object is inside the view frustum, the speed of the object (fast moving objects are less clearly seen), and effects such as fog.

Importance A measure of how much attention the user is giving, or should be giving, the object. In a game, for example, game objects (monsters, guns, keys) are more important than scenery (plants). Distance also affects importance: a nearby ring on a table is more important than a distant mountain, even if the mountain takes much more screen space. One can also assume that the user is giving more attention to objects at the center of the screen, and that moving objects grab the attention of the user more than static ones. Since importance is strongly dependent on the particular application, we do not currently use this term in our testbed.

Visibility of change Switching representations can cause hysteresis, alerting the user to the change. It is therefore preferable to make fewer, and less drastic, changes in representation. If possible, it is best to switch representations when the object is out of view. It should be noted that in the case of a powerful client, the client can take steps to reduce hysteresis, such as blending or morphing between representations.

4.1. Accuracy

Each representation has its own accuracy function, but the values should be comparable. It is important to define the accuracy functions such that representations with the same accuracy provide a visible error that is "perceptually similar". Different representations provide different visual errors. An image-based impostor can appear pixellated (when close), or at a wrong angle, while an LOD (or progressive mesh) can appear too coarse. In order to create accuracy functions that are on the same scale we have performed an experiment in which users were asked to grade the quality of image-based impostors at various scales and angles. The experiment was not perfect, as it showed objects separately (against a single color backdrop) and lacked the context of a scene. It did show some interesting points, for example that when the impostor is rotated by a small angle (up to 20–30 degrees), this hardly affects perceived quality. Our accuracy measure, empirically derived to match the results of that experiment, is:

$$\min\left(1, \frac{h_0}{h}\right) \cdot \left(\frac{1}{\sin 60} \min\left(\sin\alpha, \sin 60\right)\right)^2,$$

where h_0 is the height of the impostor's texture, h is the height (in pixels) of the object's image from the current viewpoint, and α is the angle between the impostor's base line, and the line from the eye to the impostor's center. Qualitatively, the larger the visible height, compared to the texture's height, the lower the quality, due to pixellation, and the smaller the angle, the lower the quality, with angles between 60 and 120 degrees considered to be full quality.

For geometric LOD representations the following accuracy function was used:

$$\sqrt{\frac{\max(a/F,1)}{\max(a/f,1)}}$$

Here, a is the area, in pixels, of the bounding box of the object on the screen, f be the number of faces in the LOD, and F the number of faces in the full model of the object. Accuracy is determined by the ratio between the average face size in the full object, and the average face size in the LOD, where a/F and a/f are estimates for these. A square root is taken to make this measure comparable to the impostor accuracy, which uses height rather than area.

Face sizes of under one pixel are ignored, being considered instead as one pixel, as such higher accuracy doesn't have a visual effect. As the object becomes smaller on the screen, the face size of the full object becomes smaller than a pixel, and the accuracy of the LOD increases, as the ratio between 1 and a/f is calculated. Once the LOD's face area drops below 1 pixel, it is considered to have full quality.

4.2. Visibility and hysteresis

Our implementation only considers the object's size and relation to the viewport. Objects outside the view frustum get a visibility of zero. For objects inside the view frustum, visibility is defined as the number of pixels that the object's bounding rectangle takes on the screen (in our current implementation, it is the bounding rectangle of the projection of the object's bounding box). The visibility of objects that are partly outside the view frustum is reduced accordingly.

We do not explicitly consider the effects of hysteresis in our quality measure, but our system can optionally fade out an existing representation, and fade in the new one, over a short period of time, so that hysteresis is reduced.

5. The added benefit integral

Our goal is to determine which object representations the server should transmit to the client, and in what order, such that visual accuracy is maximized over the walkthrough. More formally, let r_{ij} denote representation j of object i, with transmission duration d_{ij} and a *representation benefit* b_{ij}. Since the benefit of a representation depends on the viewing parameters, which change continuously during the walkthrough, we write it as a time-dependent function $b_{ij}(t)$.

It can be assumed that at any time t the client has already received several different representations for object i, so it is free to choose among them the best one for the current view. Thus, the actual *object benefit* of the displayed representation at time t can be expressed as:

$$b_i(t) = \max_{j\in R_i(t)} b_{ij}(t),$$

where $R_i(t)$ is the set of different representations of object i that are available to the client by time t. Due to the limited bandwidth, the sets $R_i(t)$ must satisfy:

$$\sum_i \sum_{j\in R_i(t)} d_{ij} \le t$$

Now consider for a moment an offline version of the problem, when the walkthrough path is known in advance; in this case, we are faced with the following scheduling problem: find a transmission schedule of r_{ij} that maximizes the cumulative benefit of all objects over the entire length of the walkthrough:

$$\sum_i \int_{t=0}^{t_{\text{end}}} b_i(t)dt$$

where t_{end} is the end time of the walkthrough.

In the online case, when the server considers r_{ij} as a candidate for transmission it must estimate how much r_{ij} will add to the object benefit b_i, given the representations already available at the client side at this moment in time. The "added benefit" at a particular time t is then defined as $\max(b_{ij}(t) - b_i(t), 0)$. This expression should be integrated over the remainder of the walkthrough in order to estimate the cumulative benefit of transmitting r_{ij}.

Note that while we write the added benefit as a function of t, it is really a function of the viewing parameters, which in turn change with t. The server doesn't know the true viewing parameters at future times t, and uses motion prediction to estimate them. Therefore $b_{ij}(t)$ and $b_i(t)$ are just estimates of the actual benefit values. Since we know that our motion prediction is not perfect, we limit the integration to a finite window of time into the future:

$$\int_{t=t_0}^{\infty} \max\left(b_{ij}(t) - b_i(t), 0\right) \cdot att(t - t_{cur})dt$$

Here t_0 is the earliest time of the arrival of the representation. This is the current time plus the time d_{ij} it takes the representation to arrive at the client side. *att* is an attenuation function that reduces with time, to take into account the uncertainty about path prediction. In our implementation, *att* is a simple cutoff function, that limits the range of the integral. The calculation of the added benefit integral is described in Section 6.

6. Estimation of added benefit and object selection

A complex scene contains many objects, each of which may have many different representations. In fact, the number of representations may be infinite. For example, for each object there is an infinite number of directions from which a view-dependent image-based impostor might be constructed. Obviously, the server cannot afford to take all possible representations of all objects into account when deciding which representation to transmit next: the choice must be made from a small subset of object representations.

In order to limit the set of object representations that must be considered at each instance, the server performs path prediction based on the viewpoint motion information it receives from the client. Path prediction is also necessary for the server to be able to compute the added benefit integral for each object representation, since the integral contains view-dependent benefit terms.

In order to perform path prediction, we must make some simplifying assumptions regarding possible motions of the viewpoint. Our current implementation assumes that at any given instance in time the user is either (i) standing in place; (ii) turning about; or (iii) moving in a straight line. It is important to note that these assumptions do not force the user to actually follow straight lines — the actual motion can trace a more general curve. In this case, the prediction algorithm could compute, at any instance, the average direction in which the user seems to be advancing. Also note that the optical axis of the viewing frustum need not be aligned with the viewpoint path. Thus, it is also possible to handle the strafing (sideways) movement commonly used in 3D games.

To predict the viewpoint path, we simply assume that once the user has started a particular type of motion, she is likely to continue it in the near future. This assumption allows us to discard many objects at any given instance — all of the objects that will never be seen so long as the current motion continues. Having guessed the path in this manner we can decide which representations to consider for each of the remaining objects, and compute the added benefit integral only for these representations. The following sections describe how objects and representations are selected, and how the integral is calculated for each of the three types of motion.

6.1. Standing in place

In this simple case the viewing parameters are fixed, and we only consider objects that are inside the current viewing frustum. For each object, in addition to its view-independent representations, we consider image-based impostors generated using the current viewing parameters.

In this case, the benefit of each representation does not change with time, so the integral is simply a product of the added benefit and the length of the integration interval: from the time that the representation arrives to the client until the cutoff time. The cutoff time in this case grows with time, based on the assumption that the longer the user stays in place, the more likely she is to continue to stay. This way, short stops will only consider representations that take little bandwidth, and provide an immediate improvement to the scene, but if the user takes a coffee break, the scene will look much improved when she returns.

6.2. Turning about

When turning about, we assume that the objects look the same as they move across the viewport. This is true for a cylindrical projection, and close enough for a planar perspective projection with a reasonably small field-of-view.

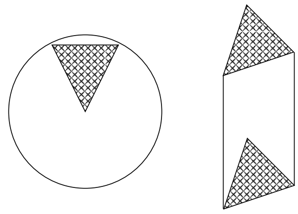


Figure 2: *Object selection areas defined by a moving view frustum. a) turning about traces out a circular selection area; b) when moving in a straight line the selection area has the shape of a prism.*

Therefore, we consider image-based impostors generated with the object rendered at the center of the viewport.

Since representation accuracy remains fixed in this case, we can take it outside the integral, and integrate only the remaining benefit terms — in our case, just visibility. Visibility has two different integration phases. When the object is fully inside the viewport, visibility is constant. When the object is at the edge of the viewport, entering or exiting, visibility grows or diminishes linearly. Both cases are simple to integrate.

When turning at a fixed speed, angle and time are interchangeable. We start integrating from the time (angle) at which the representation will reach the client. The cutoff value in this case is set to twice the field-of-view angle (in the direction of rotation). Eventually, we will consider all objects that are inside the circle that is created by turning the view frustum, as shown in Figure 2a. The far clipping plane determines the radius of this circle.

6.3. Moving in a straight line

When moving, forward or backward, the objects we take into consideration are those inside a corridor defined by sliding view frustum forward or backward along a straight line, as shown in Figure 2b. Our cutoff function determines where this corridor ends. In this case, the set of considered objects changes with time. In fact, this is the only case when we can't pre-select the objects and keep them for the entire length of the specific motion. To generate an image-based impostor for an object, we select the viewpoint half way between the time when the object enters the view frustum and the time it exits.

This case is the most complicated to integrate, since the representation's accuracy changes as we move. Visibility also changes in a non-linear manner. The representation providing the maximal benefit $b_i(t)$ at the client can also change during such movement. For simplicity, we integrate numerically in this case, by sampling the benefit function along the integration interval.

The integration interval is taken from the moment the object enters the view frustum, to the moment it exits it. This interval is further limited by the transmission time of the rep-

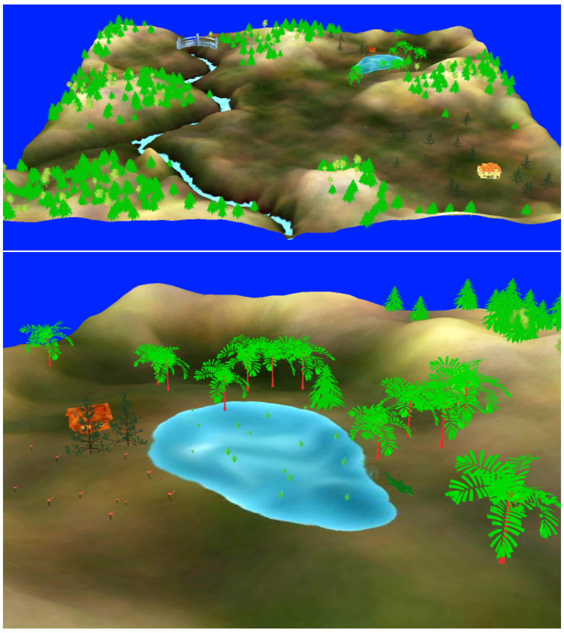


Figure 3: *Two bird's eye views of our test scene*

resentation, and the cutoff time. The cutoff value is defined as twice the time it takes the user to reach the current far clipping plane.

7. Results

In order to experiment with our streaming strategy, we implemented a testbed client-server system in Java, with rendering done through the Magician OpenGL interface library. The system is able to simulate network connections with different bandwidths between the client and the server modules. In our current implementation, object representations are transmitted through this connection without any kind of compression, except surface textures, which are stored and transmitted as JPEG images.

Our system was tested on several outdoor scenes. Because of space limitations we report the results only for the most complex scene we experimented with: a terrain populated with trees, animals, and a few man-made structures. Two bird's eye views of the scene are shown in Figure 3. The scene model consists of a terrain described by a height map, and divided into tiles, each of which has a texture. Polygonal objects, some of which are textured, were placed on this terrain. Although various objects were replicated, different instances of the same object are considered as different objects by the optimization algorithm, and indeed their geometry was replicated several times in memory. In the test described below, there are 839 objects in the scene, 256 of which are terrain tiles. The remaining 583 objects contain over 3.2 million polygons. The total size of the model is roughly 116MB, with the terrain and all textures taking under 300K, and the rest is geometric data (including texture coordinates).

In order to test and evaluate the performance of our streaming approach, we recorded a 230 seconds long walkthrough path through the scene, and then simulated a remote walkthrough along this path at three different connection bandwidths: 2000 bytes/sec, 20000 bytes/sec and 100000 bytes/sec. At these bandwidths, downloading the entire scene model would have taken 16.9 hours, 101 minutes, and 20 minutes, respectively. Note that in our tests the streaming server was not "aware" that the walkthrough path was predetermined in advance, and had no information about future motion of the viewpoint.

Figure 4*(a)–(c) shows how one frame of our walkthrough looks at the three different bandwidths. For comparison, Figure 4*(d) shows the same frame rendered using the full model of the scene.

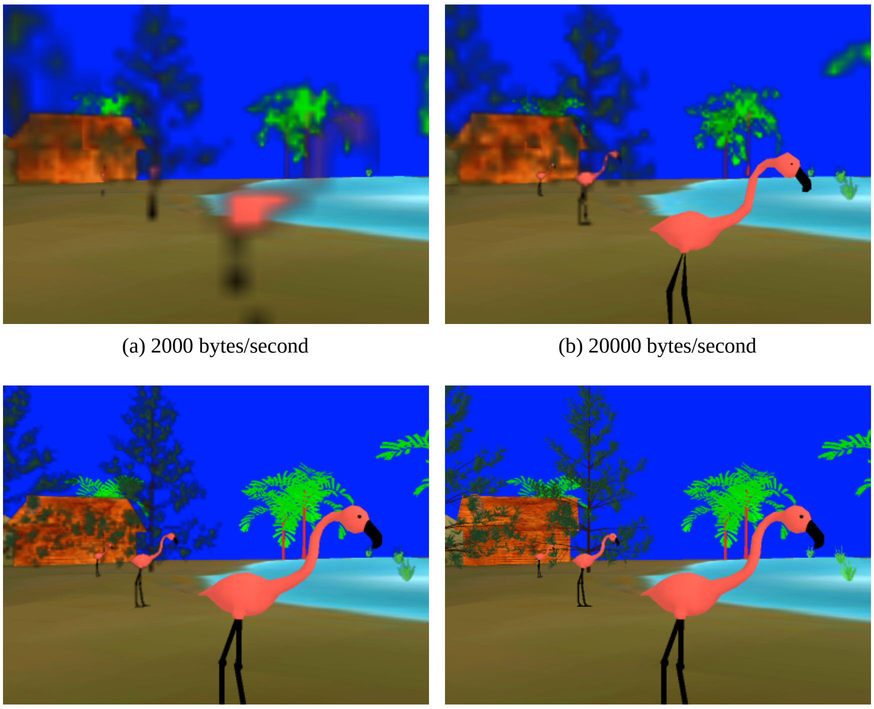
In all three tests the visible terrain tiles were the first objects selected for transmission by the server, and their full geometric description was transmitted to the client. This is not surprising, since the geometric description of each tile is rather compact, while the added benefit is large: each visible tile covers many pixels on the screen.

At 2000 bytes/sec, there is not enough bandwidth to initially send any objects except the terrain tiles, as the user quickly starts to change her direction of view. As the walkthrough progresses, crude image-based impostors start to arrive, since the geometric description of anything but the terrain is too bandwidth intensive. A couple of minutes into the walkthrough, we can see crude representations for most objects on the screen, as shown in Figure 4a. However, some objects are typically missing even after a while, especially when the user turns, and a previously unseen direction is seen for the first time.

At 20000 bytes/sec, there is enough bandwidth for decent quality representations of the objects to arrive in a more timely fashion. The flamingo in the foreground of Figure 4b is rendered using a fairly accurate LOD representation, since it contains fewer polygons than the other objects (trees and cabin), and it is close to the viewer. The other objects in the frame are still image-based impostors, but of better quality than the ones in Figure 4a.

At 100000 bytes/sec, the remote walkthrough looks very similar to the one rendered with full geometry (compare Figures 4c and 4d). Distant trees are still represented using image-based impostors (of better quality than the ones in Figure 4b), but more often than not, the full models can be transmitted in time.

In order to provide a more quantitative comparison between the three walkthroughs, we plotted the *walkthrough quality* over time in Figure 5. For the purpose of these plots, we define walkthrough quality at a particular time t as the sum of the benefit measures of all the object representations rendered at time t, divided by the sum of the benefit measures of their full representations. A walkthrough quality of 1 means that all rendered objects are indistinguishable from



(a) 2000 bytes/second (b) 20000 bytes/second

(c) 100000 bytes/second (d) full model

Figure 4: *A comparison of image qualities at different bandwidths.*

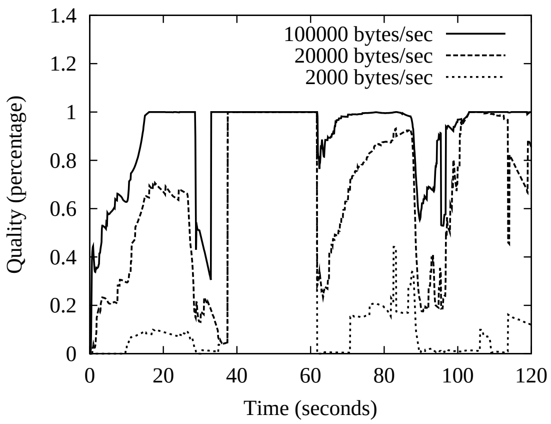


Figure 5: *Walkthrough quality over time*

their full representations. When terrain tiles are included in the walkthrough quality measurement, the resulting quality exceeds 80 percent at all times in all three walkthroughs. Therefore, in order to more clearly see the differences between the three bandwidths, we excluded the terrain tiles from the computation and plotted the total quality of the remaining objects.

Figure 5 plots the walkthrough quality during the first 120 seconds of the walkthrough. At 2000 bytes/sec all of the bandwidth is initially consumed by transmission of the ground tiles, and hence the plotted quality is zero during the first 10 seconds. Later low-quality representations are transmitted from time to time, but for the most part quality stays under 20 percent. At 100000 bytes/sec walkthrough quality rises to 100 percent during the first 16 seconds, and stays high throughout the walkthrough. Brief dips in quality occur when complex objects suddenly enter the view frustum. The 20000 bytes/sec walkthrough quality lies between these two extremes. Between seconds 37 and 62 the walktrough path goes through a sparsely populated region of the scene (at most one non-terrain object is visible), and as a result the quality is at 100 percent for all three bandwidths.

Note that 2000 bytes per second is even less than the bandwidth available with today's standard modems (which is about 5000 bytes per second), and recall that our current implementation transmits object representations without any compression, using 4 bytes per pixel for image based impostors, and floating point coordinates for geometry. Thus, our results indicate that with compression, a good quality remote walkthrough of our test scene would have been obtained even through a standard modem connection.

8. Summary

We have described a new and general 3D scene streaming approach that allows users to walk through a large scene residing on a remote server, without having to wait for the entire scene to be downloaded to the client computer. Our approach employs an online optimization algorithm, which schedules object transmissions based on the integral of added benefit along a predicted viewpoint path. We have demonstrated that the approach adjusts well to different connection bandwidths.

This paper describes work in progress. We would like to extend our framework and implementation to incorporate progressive meshes, as another supported type of object representation. Progressive meshes fit very nicely into our concept of added benefit. We are also planning to add occlusion culling, in order to more effectively handle indoor scenes. Another fairly straightforward improvement to our framework would be to add hierarchical image-based impostors[23], thus allowing efficient transmission of distant clusters of objects.

Other interesting topics for further study include investigating better motion prediction heuristics for the viewpoint path, developing a more sophisticated online optimization algorithm, incorporation of other representations (such as relief textures[19]) into our optimization framework, incorporation of state-of-the-art compression techniques for more efficient transmission of object models, and further study of appropriate benefit measures and quality perception by users.

References

1. J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):41–50, Mar. 1990. 2
2. D. G. Aliaga and A. A. Lastra. Architectural walkthroughs using portal textures. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 355–362, Nov. 1997. 2
3. R. Carey and G. Bell. *The Annotated VRML 2.0 Reference Manual*. Addison-Wesley Developers Press, 1997. 1
4. S. E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics* Proceedings, Annual Conference Series, pages 279–288, Aug. 1993. 2
5. J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, Oct. 1976. 2
6. D. Cohen-Or, Y. Mann, and S. Fleishman. Deep compression for streaming texture intensive animations. In *Computer Graphics* Proceedings, Annual Conference Series, pages 261–268, Aug. 1999. 3
7. M. F. Deering. Geometry compression. In *Computer Graphics* Proceedings, Annual Conference Series, pages 13–20, Aug. 1995. 3
8. T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 85–92. ACM SIGGRAPH, Apr. 1995. 3
9. T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics* Proceedings, Annual Conference Series, pages 247–254, Aug. 1993. 2
10. S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *Computer Graphics* Proceedings, Annual Conference Series, pages 43–54, Aug. 1996. 2
11. N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Computer Graphics* Proceedings, Annual Conference Series, pages 231–238, Aug. 1993. 2
12. G. Hesina and D. Schmalstieg. A network architecture for remote rendering. In A. Boukerche and P. Reynolds, editors, *Proceedings of Second International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pages 88–91, July 1998. 3
13. H. Hoppe. Progressive meshes. In *Computer Graphics* Proceedings, Annual Conference Series, pages 99–108, 1996. 3
14. H. Hoppe. View-dependent refinement of progressive meshes. In *Computer Graphics* Proceedings, Annual Conference Series, pages 189–198, 1997. 3
15. M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics* Proceedings, Annual Conference Series, pages 31–42, Aug. 1996. 2
16. M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A network software architecture for large scale virtual environments. *Presence*, 3(4), 1994. 3
17. P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *1995 Symposium on Interactive 3D Graphics*, pages 95–102, Apr. 1995. 2
18. L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics* Proceedings, Annual Conference Series, pages 39–46, Aug. 1995. 2
19. M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Computer Graphics* Proceedings, Annual Conference Series, pages 359–368, July 2000. 9
20. G. Schaufler and W. Stürzlinger. A three dimensional image cache for virtual reality. In *Proceedings of Eurographics '96*, 1996. 2
21. D. Schmalstieg. *The Remote Rendering Pipeline*. PhD thesis, Technical University of Vienna, 1997. 3
22. B.-O. Schneider and I. M. Martin. An adaptive framework for 3D graphics over networks. *Computers and Graphics*, 23(6):867–874, Dec. 1999. 3
23. J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Computer Graphics* Proceedings, Annual Conference Series, pages 75–82, 1996. 2, 9
24. S. Singhal and M. Zyda. *Networked Virtual Environments*. Addison-Wesley, 1999. 3
25. G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, Apr. 1998. 3
26. S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Computer Science Division (EECS), UC Berkeley, Berkeley, California 94720, Oct. 1992. Available as Report No. UCB/CSD-92-708. 2
27. C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34, June 1998. 3
28. I. Yoon and U. Neumann. Web-based remote rendering with ibrac (image-based rendering acceleration and compression). *Computer Graphics Forum*, 19(3):321–330, 2000. 2
29. H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Computer Graphics* Proceedings, Annual Conference Series, pages 77–88, Aug. 1997. 2