Distributed Rendering of Virtual Worlds

Siddhartha Chaudhuri Daniel Horn Pat Hanrahan Vladlen Koltun

Virtual Worlds Group
Stanford University

Figure 1: *A 2,500 square kilometer region of San Francisco Bay Area terrain with 10 billion polygons of geometric data, streamed and rendered in real time over a broadband Internet connection. This panoramic image was stitched from screen shots of our real-time rendering environment, obtained by rotating the camera around a static viewpoint.*

Abstract

We present a system for distributed rendering of large and detailed virtual worlds. A cluster of servers create and maintain a hierarchical representation of the world that can be propagated to an unlimited number of clients through a content-distribution network. The preprocessing is easy to parallelize and storage requirements are minimal. Using a small subset of the representation, a client can explore a large and detailed world in real time on consumer hardware over a commodity Internet connection. The required bandwidth is independent of the size of the world. We report extensive performance measurements with a 2,500 square kilometer world, densely populated with objects comprising 10 billion polygons.

1 Introduction

Detailed three-dimensional models of vast geographic environments are becoming more common. Services such as Google Earth and Microsoft® Virtual Earth provide large geo-referenced databases of imagery and terrain data, and three-dimensional models of buildings and vegetation are rapidly being integrated [Google Inc. 2008]. Semi-autonomous modeling of extended urban areas is now practical [Bosse et al. 2004; Thrun and Montemerlo 2005]. It is expected to yield an abundance of three-dimensional geometry that can be incorporated into these services to create detailed virtual cities, countries, and continents.

Networked virtual worlds that do not correlate with real spaces already host millions of participants daily [Miller 2007]. Their scale approaches real-world cities in terms of land mass, population, and economic vitality. These three-dimensional online environments are being used for social interaction, commerce, education, and scientific research [Bainbridge 2007]. Some allow users to contribute their own three-dimensional content. Such worlds now contain hundreds of thousands of continuous virtual acres covered with detailed user-generated architecture and flora.

A critical bottleneck in engineering such systems is the display of complex three-dimensional environments to many simultaneous users in real time over a bandwidth-limited network. Some of the most popular virtual worlds are too large to fit on any single machine and cannot be preloaded. A common solution in practice is to transmit and display only nearby geometry: objects that are immediately adjacent to the viewpoint. The typical cutoff radius is a few hundred meters or less. This hinders the participants' experience by precluding long or panoramic views and limits the visual nature of the worlds.

This paper describes a system for the representation and display of distant geometry in virtual worlds. It is tailored to environments that extend across vast territories and feature local detail throughout. It has the following distinctive characteristics:

- It is appropriate for worlds with fine detail at planetary scales.
- It can accommodate any geometric model, not just special classes such as terrains.
- Its storage cost is a small fraction of the original world data.
- The representation can be computed in linear (or parallel logarithmic) time.
- It can be created on a commodity cluster using simple parallel algorithms.
- It can be served to an unlimited number of users using existing content-distribution networks.
- Changes to the model are quickly and efficiently processed and the update can be transmitted to clients within a short (though not real-time) period of time.
- Finally, the bandwidth needed is bounded as a function of maximal user velocity and is independent of the size of the world. In practice, the consumed bandwidth is a small fraction of the bandwidth required to maintain nearby geometry.

The system hierarchically partitions the world, storing in each cell a collection of orthographic depth images of the cell contents, taken from a small set of canonical directions. Once generated,

the depth images can be delivered by web servers or a content-distribution network. A local change can be propagated up the hierarchy in a logarithmic number of steps.

A client uses standard techniques to render the region that lies within a fixed small distance around the viewpoint. A logarithmic set of depth images are used to display the rest of the world. If a participant's velocity is bounded, the bandwidth required to maintain these depth images during real-time exploration is independent of the size of the world. If the viewpoint is far from complex geometry, for example at high altitudes, velocity can increase exponentially within the same bandwidth limit. The client can optimize the display of distant depth images by compositing them into cubical environment maps, thus bringing the rendering load from logarithmic down to constant.

We evaluate the system using scenes that capture the characteristics of our target worlds. Our primary test scene stretches over a 50 x 50 km^2 expanse covered with buildings and other objects at density and level of detail that are typical of modern virtual world environments. It comprises 10 billion polygons. This scene can be preprocessed on a 16-node server cluster in about two hours and the size of the resulting representation is only 2% of the storage consumed by the world model itself. We demonstrate that a client running on a consumer workstation can explore this scene in real time over a broadband Internet connection. The bulk of the bandwidth is consumed by nearby geometry. Our representation provides the experience of a visually complete world at a bandwidth increase of roughly 10% over the transmission of nearby geometry alone.

2 Design Goals

The system was designed to address the following goals. These are derived from the needs of existing virtual world platforms and their expected growth.

Large worlds: The system must scale to worlds that spread across virtual continents and are densely covered with individual objects and local detail. Previous work has often focused on smaller regions. For the emerging worlds, models of whole countries are being assembled.
To develop a valuable sense of scale, consider planet Earth, which has 150 million square kilometers of land area. At an average density of one polygon per square meter (our test scene has 4 p/m^2), this yields 150 tera-polygons. We view this as a useful guidepost for research on scalable virtual world systems.

General content: The approach must handle an unstructured and unpredictable variety of geometric content. Many techniques were developed for terrains, indoor environments, cities, and individual objects. We seek a general approach that is compatible with any type or representation of input geometry. Because of the user-modifiable nature of some of the most intriguing virtual worlds, content is largely unpredictable and varies through time.

Many users: The system must serve society-scale numbers of concurrent participants. Although it is unavoidable that the computational costs of networking, physics simulation, and world logic rise with the number of participants, the computational load imposed on the server by rendering and geometric representation tasks should ideally be independent of the number of users. Furthermore, artificial limits must not be placed on the number of participants simultaneously occupying the same local area of the world.

Bounded storage: Storage cost must be a linear function of the input model size. Ideally, a proportionately small overhead will be added to the storage already consumed by world data such as meshes and textures.

Bounded preprocessing: The time needed to compute the representation must be linear in the size of the world.

Bounded bandwidth: The bandwidth required per individual client must be independent of the size of the world. Ideally, at most a small linear overhead will be added to the bandwidth required to maintain objects that are within short distance of the viewer. Such objects need to be transmitted in any case for latency reasons.

Dynamic updates: The time to update the representation following incremental changes to the world must be at most logarithmic in the size of the world. In the new user-modifiable virtual worlds, the content of the scene can change significantly during the world's lifetime. Approaches that potentially require a substantial fraction of the preprocessing to repeat in response to a local change in the world are not viable.

Easy parallelization: Virtual worlds are already too large to be stored or processed on any single server. Existing commercial systems use commodity clusters connected by local area networks. Such clusters work best when algorithms are easily parallelizable, such as with Google's map-reduce framework [Dean and Ghemawat 2004]. Algorithms that require optimization across the extent of the scene would be hard to parallelize on existing data centers and are less appropriate.

Visual fidelity: The visual experience provided by the system on every client should be maximized and should strive to approach the quality of a local real-time environment.

3 Assumptions

We believe the following assumptions are acceptable to virtual world participants and justifiable in light of current platforms. They allow us to develop a practical system that addresses all of the stated design goals.

Static subset: We assume that a significant fraction of spatially large objects in the world are static. This holds for most of terrain, buildings, and vegetation. Note that many semantically significant dynamic objects, such as humans, can be succinctly represented in parametric form for extremely efficient transmission [Allen et al. 2003]. Our system is compatible with direct communication and rendering of such objects.

Bounded density: We assume that the world can grow to arbitrary size, but that the spatial density of geometry (polygons per square meter) is bounded by a constant. This bounds the amount of work performed by any client or server.

Bounded velocity: We assume that a participant's speed of movement is bounded. This is used to analyze the network bandwidth consumed by new regions coming into view. This does not preclude discontinuous forms of transport such as teleportation. The velocity limit grows exponentially if the participant is away from geometric detail, for example in flight.

Precomputed shading: We assume that shading can be precomputed for distant static objects and reused despite changes in the view direction. This is a very common assumption for distant geometry in real-time environments.

4 Overview

Our system is a distributed client-server implementation. When used, there are many clients and multiple servers. See Figure 2 for an overview.

To create the hierarchical representation, static geometry in the world, such as terrain and buildings, is subdivided into an octree [Samet 2006]. The leaf level is a regular grid of cells that covers the world. Each server is assigned a separate subtree and these subtrees partition the world. Any number of servers can be accommodated. For simplicity, assume that the world is partitioned among servers at a particular octree level, so that this partition forms a coarse grid.

For each leaf cell, the corresponding server renders its content from a number of canonical directions to form a set of orthographic

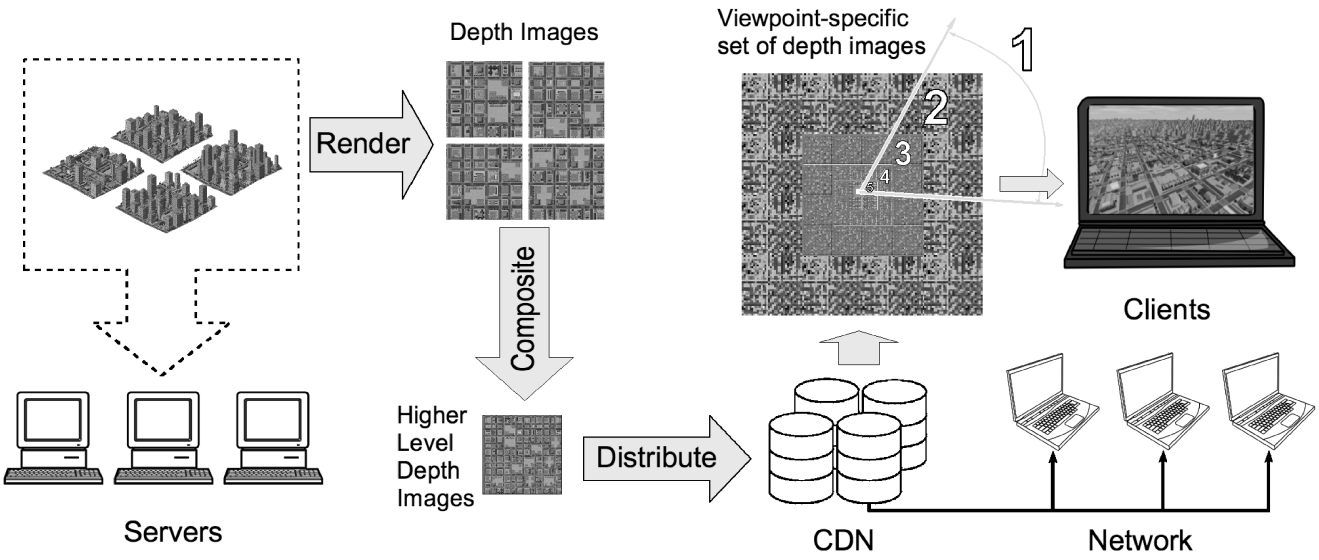


Figure 2: *An overview of the system.*

depth images. The set of directions we use is a subset of the principal directions of a cube, as shown in Figure 3(a). The resolution of the depth images is a parameter, by default set to match the expected screen resolution on the client rendering platforms. The views may be rendered using any rendering system, including graphics hardware and/or high-quality off-line ray tracers.

Depth images for higher-level cells are obtained by compositing the corresponding images from the children, as shown in Figure 3(b). The same canonical directions are used throughout. To form an orthographic depth image at a non-leaf cell, the eight corresponding images from the children cells are combined using RGBZ compositing. The result is a single depth image at the same resolution as the original eight that covers twice as much of the world along each axis.

The bottom-up construction of the hierarchy is handled by a single server within its subtree. These subtrees are then combined to populate the highest-level cells. This is handled analogously to the lower levels, except data is now transferred over a local area network. This stage concludes rapidly since the number of high-level cells is proportionately very small.

Once the depth images are created, they may be transferred to web servers or a content-distribution network. This makes it possible to scale the system to many more users, as modern CDNs serve hundreds of millions of people per day [Akamai Technologies, Inc. 2008].

The hierarchical arrangement ensures that updating the geometric content of a leaf cell requires only to re-render the associated depth images and then propagate the update through a logarithmic number of cells by using fast image compositing operations. The revision can be distributed through the CDN using standard techniques.

The clients are remote machines connected to the network. We assume each client has full rendering capabilities and a GPU. A client is associated with a single viewer. It maintains valid geometry within a fixed distance around the viewer by communicating with a server. The client also maintains a set of depth images that allow it to display the remainder of the world. This set is kept current by a background thread that predictively fetches new images from the network as the viewer moves through space.

We show that a logarithmic number of depth images are generally sufficient for a perceptually satisfactory display of the world. More distant regions are covered by higher-level images. These images are sparser, in the sense that a given area is represented by fewer depth pixels. However, such regions are also more coarsely represented on the viewer's screen, where a single pixel covers greater area as well. The system quantitatively balances these considerations.

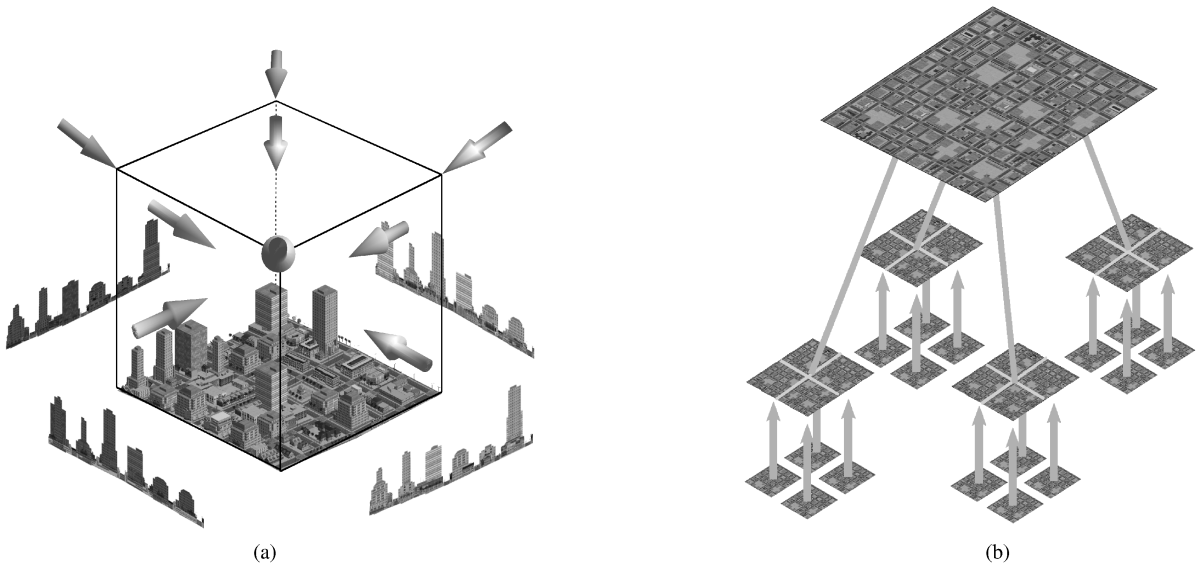
To reduce the number of depth pixels drawn by the client, the more distant depth images are combined into a cubical environment map whose faces are depth images. These depth images are reused within a cell. This brings the number of displayed depth images from logarithmic down to constant. This step is optional and results in improved performance if the available hardware cannot render the original number of three-dimensional points in real time. (About two hundred million for our largest test scene.)

This system satisfies the stated design goals. The preprocessing runs in linear time and uses linear storage that is not much greater than that required to store the world model itself. Parallelization is simple and the number of servers is easily varied. Finally, assuming the viewer's velocity is below an appropriate threshold, the required bandwidth is bounded independently from the size of the world.

Several design issues were given careful consideration in the architecture of the system:

Geometric level-of-detail vs. Image-based techniques: Our system has to accept a wide and unpredictable variety of geometric content. It must represent huge and unstructured collections of objects succinctly. Even in light of significant advances in geometric simplification [Luebke et al. 2002], image-based approaches are inherently less model-dependent.

Light fields vs. Depth images: A densely sampled light field [Gortler et al. 1996; Levoy and Hanrahan 1996] represents the outgoing lighting distribution from a complex object. New views can be generated from the light field using resampling. RGBZ images approximate a light field well, except at occluding contours and where the object is shiny. Theoretically, adding depth to a light field makes it possible to decrease the angular resolution. This allows us to use only a few directions when creating the depth images. After initial trials, we chose against uniformly sampled light fields in favor of depth



(a) (b)

Figure 3: *(a) The nine canonical directions for orthographic depth images. The figure shows the content of a leaf-level octree cell and four of the resulting depth images, shown without the depth information for clarity. (b) Higher levels of the hierarchy are assembled from the lower levels using rapid image compositing operations.*

images in order to minimize preprocessing, storage, and bandwidth costs.

Hierachical points vs. Depth images: One of the best general methods for handling level of detail is hierarchical point representations as used in QSplat [Rusinkiewicz and Levoy 2000]. Point representations uniformly sample geometry and can thus be splatted without introducing holes. However, a disadvantage of the point representation is that the number of points depends on the depth complexity, since interior points are represented in the upper levels of the hierarchy. When such cells are drawn, they will have high depth complexity. An advantage of depth images is that the depth complexity will always be constant, an important issue if we want to guarantee that a cell can be drawn in constant time.

Layered depth images vs. Multiple views: In general, no canonical direction will be perfectly aligned with a particular viewing angle. Thus an object that is occluded in a depth image may be visible to a viewer. This leads to the well-known problem of disocclusion, caused by missing data in the reprojected image. Two approaches have been proposed for minimizing such artifacts: utilizing depth images from multiple viewpoints [Max et al. 1999; Lischinski and Rappoport 1998] and layered depth images (LDIs) [Shade et al. 1998]. Having images from widely varying directions allows us to more uniformly sample objects. After experimenting with both methods, we opted for increasing the number of depth images per cell instead of adding layers of depth. However, a long and narrow inset or passage that is not aligned with any of the projection directions may still appear to be missing. We recognize that this is a fundamental problem with any system using depth images, and is a limitation of our system.

5 Server Architecture

The geometry in the world is partitioned into a regular grid of cells, each 200 x 200 x 200 m^3. These cells form the leaves of a pyramid, or octree. The octree subdivision can be asymmetric, for example if the world has significantly greater extent in some of the axes. Our test scenes are set on terrains and required no subdivision in the vertical axis.

The world is distributed among the available servers for preprocessing. Each server is responsible for a subtree of the hierarchy. These subtrees are equal in size. If there are n servers, each is responsible for $1/n$-th of the world.

For each cell (leaf or otherwise) in the octree, orthographic depth images of the cell's content from a number of directions are created. The set of directions is the same for all cells. Our implementation uses a subset of the principal directions of a cube, namely $\{-1, 0, +1\}^3 \setminus \{0, 0, 0\}$. The specific nine directions we used are illustrated in Figure 3(a). Note that upward-facing directions were not included because our test scenes have few overhanging faces. Any set of directions could have been used, with a proportional increase or decrease in storage and preprocessing time.

For the leaf cells, the depth images are created on the respective servers by rendering the contents of each cell. Each server node reads in the model for that cell, renders all orthographic views for the different directions, and then writes the depth images to disk. Reading the model once saves disk bandwidth, which is often the rate limiting resource. Rendering may be performed using a graphics card or software renderer. The rendering setup on the server should match the rendering setup on the clients to eliminate seams and other differences in appearance. Finally, when rendering we turn on full antialiasing to remove as many aliasing artifacts from the leaf images as possible. If the polygons and texels are bounded by a minimum size, we can render at resolution that ensures that they are sampled at the Nyquist frequency.

Depth images for higher-level cells are created from lower-level images using two-dimensional compositing. This is illustrated in Figure 3(b). Consider a cell that is part of a subtree handled by a single server. To create an orthographic depth image from a particular projection direction, the corresponding images from the eight children cells are composited. The resulting image has the same resolution as the original eight but covers twice as much of the world along each axis. We composite at the resolution of the children images and then average down by a factor of two. The depth of an output pixel is the closest of the depths of the candidate pixels in the composited images.

Depth images for the highest-level cells that transcend server boundaries are assembled analogously, the only distinction being that images are exchanged using a fast local area network linking the servers. Each one of these highest-level cells is assigned in a bottom-up process to a server that handles one of the children cells. Since the highest levels of a hierarchy contain exponentially fewer nodes, this final stage is considerably faster for our test scenes than the intra-server assembly of the lower levels.

To summarize, this algorithm is linear in space and time because the number of octree cells is proportional to the number of leaf cells, and each step takes constant time if we assume the scene characteristics that were proposed in Section 3. It is also easy to parallelize.

6 Client Architecture

The client uses the scene graph (including mesh and texture data) for objects that fall within a fixed radius around the viewer's location. This requires 4 (for 2.5D scenes) or 8 (for full 3D octrees) leaf cells. The scene graph is transferred to the client using the CDN. Rendering the nearest parts of the world as geometry reduces latency and guarantees that nearby objects are drawn faithfully.

The more distant parts of the world are displayed by rendering depth images. These depth images are found by traversing the octree from the top, marking each cell for display if its associated depth images have sufficient resolution. They have sufficient resolution if a hypothetical pair of adjacent pixels would be separated by at most one pixel after reprojection. We can conservatively bound the distance between pixels because we know the depth range associated with the octree node. When a qualified cell is reached, its subtree is not traversed further. The set of displayable octree cells is maintained by a background thread that downloads the associated depth images from the CDN. We have also developed several simple algorithms for prefetching cells before they are needed.

The above algorithm guarantees that the resolution of distant octree cells is always the same. Standard counting arguments can be used to show that at any viewpoint the number of relevant cells from each level of the hierarchy is constant [Samet 2006]. Thus the entire background can be represented using a logarithmic number of cells. For each of these cells, we identify and download a small number of depth images (three by default) whose projection directions make the smallest angle with the view direction.

If a viewer is moving at constant velocity, we can determine the rate that is needed to fetch new octree cells. Regan and Pose [1994] showed that a cell in a hierarchy (in their case, hierarchical environment maps) need only be updated when the user crosses a boundary of a cell of the same size as the octree cell; this hypothetical update cell is centered on their viewpoint. Thus, nearby octree cells change at twice the frequency of the next level of the hierarchy, which changes at twice the frequency of the next, etc. The bandwidth required to maintain all relevant depth images is thus less than twice the bandwidth consumed by updating the leaf cells. This is a constant factor of the viewer's movement speed and is independent of the scale and complexity of the world.

If the lowest-level octree cells occupied by the viewer are empty, for example if the viewer is in flight and is not near any geometric objects, depth images that come from cells at these levels need not be rendered or updated. This allows the maximal movement speed achievable within a certain bandwidth cap to increase exponentially as the viewpoint moves away from detailed information.

Rendering load reduction. The total number of points in the depth images that the client is required to maintain for our largest test scene is over two hundred million (assuming an image resolution of 640 by 480). This number grows logarithmically in the size of the scene. For example, squaring each dimension (e.g., from 50 x 50 km^2 to 2,500 x 2,500 km^2) would double the rendering load to about half a billion points. Doubling the image resolution would also double the number of points. This is just beyond the real-time capabilities of current graphics hardware.

The rendering load can be reduced by compositing the depth images into a small set of cubical environment maps centered near the current viewpoint. The faces of the environment maps are themselves depth images. We found that a single environment map suffers from disocclusion artifacts, so we create 8 cubical environments each centered at the vertices of the cubical cell surrounding the viewpoint. As the viewpoint approaches a cell boundary, we create new cubical environment maps at the new vertices of the adjacent cell. Normally only 4 new environment maps need to be rendered when the face is crossed.

With an image resolution of 640 by 480, we found that good results are obtained if the resolution of the environment map faces is 1024 by 1024. Thus, the 8 cubical environments contain 48 million points. This is much less than the 200+ million points in the octree cells, and hence significantly speeds up the rendering of individual frames.

The cost of this reduction in rendering load is offset by the cost of creating the environment maps. This cost is amortized over many frames since the environment maps are reused while the viewpoint remains in the same cell. However, care is required to prevent this compositing from interfering with the primary rendering task. Our initial implementation conducted the compositing process on the GPU, but we found the system was faster if the compositing was performed on a number of background CPU threads using fast SSE algorithms, taking advantage of multiple cores when available.

Finally, we have opted to use the simplest possible point rendering algorithm, drawing each point as a square equal in size to a pixel. Points are rendered using antialiasing, and we oversample the environment maps by a factor of two to further reduce aliasing and decrease the chance of a hole appearing.

7 Implementation and Results

The following table describes two of the models used to test our system. The first is a flat 12.8 x 12.8 km^2 plane populated with pre-modeled geometry. To create the second scene we used a 51.2 x 51.2 km^2 region of San Francisco Bay Area terrain at 10m resolution, obtained from the San Francisco Bay Area Regional Database (`http://bard.wr.usgs.gov`). We populated this terrain with a mix of hand-modeled buildings and vegetation, and procedural skyscrapers. The skyscrapers were placed in accordance with a population density map from the Spokane, WA, region. (The Bay Area population density map did not yield visually pleasing results because of the bay that cuts through the area. This also accounts for the relative flatness of our model.)

	Scene 1	*Scene 2*
Area	12.8km × 12.8km	51.2km × 51.2km
Max altitude	150m	350m
Total polygons	614.4 million	9.8 billion
Model storage	294 GB	1.0 TB
Octree levels	7(X) × 1(Y) × 7(Z)	8(X) × 1(Y) × 8(Z)
Leaf cell area	200m × 200m	400m × 400m
Polygons/meter^2	3.75	3.85

Note that although some objects repeat many times in the test scenes, we utilized no instancing. Every occurrence of an object is treated as a distinct model. It is reloaded afresh from the disk and stored separately in memory, thus we do not take advantage of the repetitions.

The system was implemented on a number of Unix-based platforms, including 32- and 64-bit Linux and Mac OS X, and tested with a variety of NVIDIA and ATI GPUs. Our primary cluster was a 16-node Linux rack with fairly old ATI 9800 graphics chipsets.

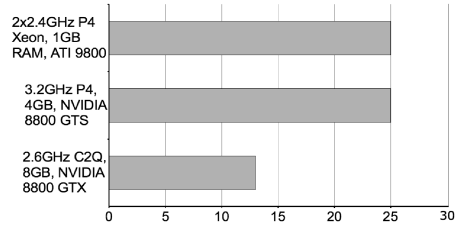


Figure 4: *Preprocessing times, in minutes, for 1/16-th of Scene 1 on a single server node. These equal the preprocessing times for Scene 1 on respective 16-node server clusters.*

The test models were partitioned into 16 regions and distributed among the servers. Depth images were generated at 1024×1024 resolution. Figure 4 shows the preprocessing times for Scene 1 on three server configurations.

For compression, each depth image was divided into color (RGB) and depth (Z) components. The color component was heavily compressed using a JPEG codec at the 50 quality setting. The depth component was treated with lossless ZIP compression. These two compressed buffers were serialized into a single depth image. Typical file sizes were in the 100-300KB range, with occasional complex views approaching 1MB. The size of the complete database is as follows:

	Model size	Size of all depth images	Storage overhead
Scene 1	294 GB	1.5 GB	0.5%
Scene 2	1.0 TB	22 GB	2%

Thus the storage overhead of the hierarchical depth image representation is negligible compared to the cost of storing the geometry itself.

A basic HTTP server (MiniHttpd) was set up on each of the 16 nodes. The client requested depth images by filename. These were typically transmitted in a fraction of a second over our fast network connection (intra-departmental Ethernet). For a more representative scenario, we hosted the content distribution server on a remote machine that we connected to over a broadband Internet connection. We provide empirical evidence from both scenarios. We found that a fast broadband connection can support rates of movement similar to those on a Local Area Network, because the computational load of compositing depth images into environment maps remains a bottleneck when network bandwidth is no longer an issue.

Our primary test machine for the client was a quad-core workstation with 8GB of memory and an NVIDIA 8800 GTX graphics card. While this is on the high end of current workstations, it was bought off-the-shelf in a consumer electronics store, pre-assembled as a Hewlett-Packard Pavilion m9180f. We upgraded it only by doubling the default 4GB of RAM and replacing the graphics card. To reduce memory requirements on the client end, depth images were kept in memory in compressed form and decompressed on-the-fly as required for compositing.

The active set of depth images at any given time numbered about 200 for the 12.8km scene and 250 for the 51.2km scene. This gives a total of about 200 million points to splat, which was too slow on our test system. By compositing the depth images into environment maps, as described in Section 6, we were able to achieve interactive rendering rates: 30+ fps for depth images alone and about 10-20fps for depth images and geometry. A plot of the frame rate during an automated test run is presented in Figure 5. The variability in the frame rate is caused by environment map compositing. Better scheduling algorithms that involve prefetching should alleviate this behavior.

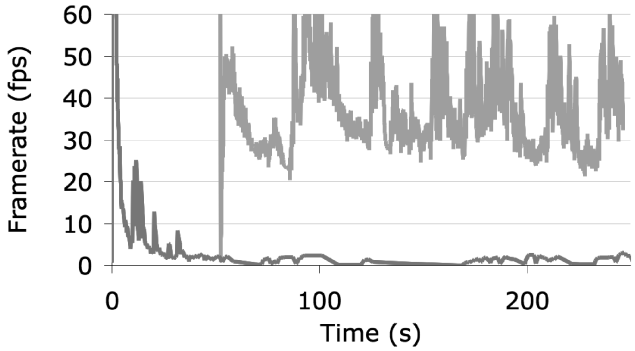


Figure 5: *Client frame rate measured over an automated four-minute test path through Scene 2, traced at 15 meters/second over an Internet connection. The red curve represents direct rendering of all of the depth images on the GPU. The green curve reflects the use of environment map compositing as described in Section 6. The periodic frame rate peaks occur as the viewer crosses cell boundaries, and environment maps that are no longer relevant are discarded.*

Bandwidth utilization during the same test run is shown in Figure 6 for both Internet and LAN connections. Note that the movement speed in these experiments is 15 meters per second, which is over 30 miles per hour—a normal driving speed.

The bandwidth consumed by depth images was between 6% and 12% of the bandwidth consumed by nearby geometry, as shown in Table 1. Thus, for additional 12% bandwidth, our system extends the visible part of the world to the horizon.

We also measured the time taken to update the hierarchy when the content of a leaf cell (400 x 400 m^2 area) is modified. When all depth images that are touched are loaded cold from disk, the update takes 20.5 seconds. (15.2 seconds for the lower 6 levels, 5.3 seconds for the upper 2 levels, averaged over 6 tests.) Thus, local updates such as a newly erected or razed building can be rapidly propagated through the system.

8 Related Work

Image-based systems. The seminal MMR system [Aliaga et al. 1999] utilized an array of advanced geometric and image-based techniques to enable interactive visualization of complex static data sets such as a detailed power plant model. The system partitions the model into cells and for each cell creates a set of textured depth meshes [Darsa et al. 1998; Mark et al. 1997; Sillion et al. 1997] that depict how the rest of the model appears from within the cell. This general approach is not appropriate for our application scenario. The target virtual worlds are so huge that no single server node would be able to render (or even store) the entire world to create the requisite textured depth meshes. Furthermore, the preprocessing time is quadratic in the size of the model, as the entire model has to be rendered from each of the cells. Lastly, a local change in the world would potentially require recreating the entire set of textured depth meshes, violating the dynamic updates requirement. In spite of these issues MMR was a key inspiration for all subsequent systems.

Many other approaches spread image-based representations throughout the environment to assist subsequent rendering. Decoret et al. [Decoret et al. 1999] describe multi-layered impostors and utilize these for rendering acceleration in a similar fashion to MMR. Other methods take the particular scene geometry into account in positioning the image-based representations [Aliaga and Lastra 1999; Jeschke et al. 2005; Rafferty et al. 1998; Shade et al. 1996; Wilson and Manocha 2003]. Our design goals necessitate a system

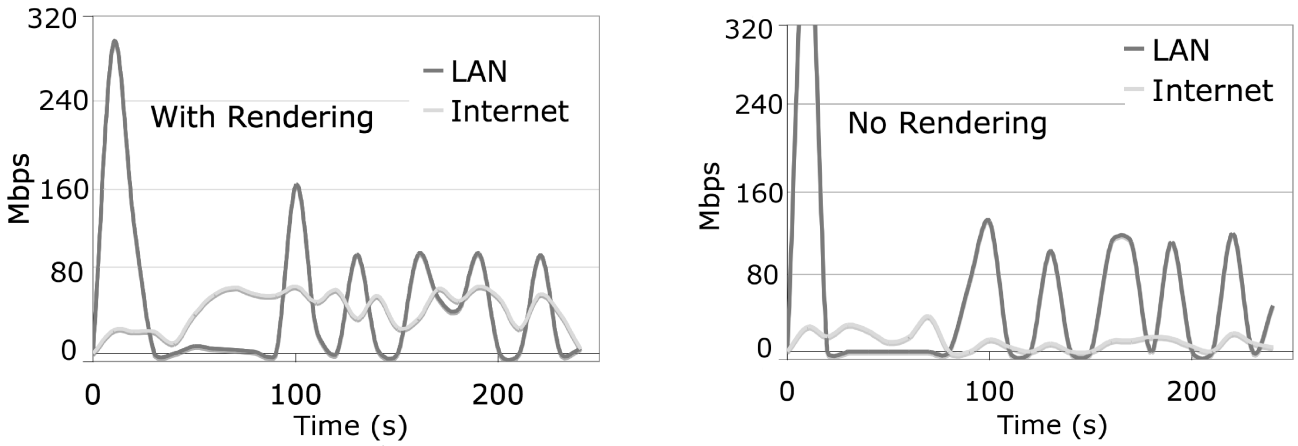


Figure 6: *Network bandwidth utilized in an automated four-minute test path through Scene 2, traced at 15 meters/second over two different network connections: a fast local area network and a consumer Internet connection. The first experiment (left) was conducted with default rendering. The second experiment (right) was conducted with all compositing and rendering turned off, simulating a situation in which computational load is never a bottleneck. Note that the client software utilized additional bandwidth provided by the fast LAN connection for batch downloads. The sharp spike at the beginning of both red graphs represents the rapid download of the initial set of geometry and depth images over the LAN; subsequent spikes correspond to points where cell boundaries are crossed and new depth images are batch downloaded.*

that is largely independent from the particular scene content to ensure fast updates, easy parallelizability, and rapid and numerically stable preprocessing.

Schaufler and Stürzlinger [1996] cover an environment with images in a regular grid that is similar to ours. Their images contain no depth information and have to be dynamically generated as the viewer moves through the scene. Their approach was not designed for and would not be appropriate for distributed applications, but is an important precursor of our system due to its simplicity and agnosticism to the particulars of the model.

Depth images have also been proposed to represent complex scene geometry [Chen and Williams 1993; McMillan 1997]. Shade et al. [1998] introduce the Layered Depth Image as a rendering primitive. These are used by Chang et al. [1999] to create LDI trees that hierarchically represent three-dimensional models. This elaborates upon the approach of Schaufler and Stürzlinger [1996] by adding (layered) depth information. Max proposed hierarchical z-buffers for drawing trees [Max and Ohsaki 1995]. He also proposed to use multiple parallel projections along the cubical face directions, an approach which later extended to use texture mapping hardware [Max et al. 1999] and LDIs [Lischinski and Rappoport 1998]. We were inspired by these papers in developing our system.

Another important precursor to our work was the priority rendering approach of Regan and Pose [1994]. They made the observation that distant environment maps can be rendered less frequently than those up close. By counting the update rates under constant velocity, they showed that only a constant number of updates are needed on average. Similar arguments apply to mipmaps [Tanner et al. 1998]. On average the bandwidth needed to prefetch mipmaps is constant when moving at constant velocity. We use this basic insight to prove our bandwidth bounds.

Other rendering acceleration techniques. Since the number of pixels on the display is constant, the ideal real-time rendering algorithm will strive towards performance that is largely independent of scene complexity [Clark 1976; Sudarsky and Gotsman 1997]. One avenue of research towards this goal is geometric level of detail techniques that produce coarse representations for distant objects, see [Luebke et al. 2002] for a recent survey. Level of detail has been successfully applied to terrains. The mipmap representations of textures [Williams 1983] coupled with the clipmap technique for prefetching and culling [Tanner et al. 1998; Losasso and Hoppe 2004] make it possible to roam over the Earth in real-time. These techniques do not work well for our application because we need to simplify collections of millions of objects.

Another approach to rendering acceleration is to cull parts of the scene that are outside the field of view or occluded [Cohen-Or et al. 2003]. Airey et al. [1990] and Teller and Séquin [1991] were among the first to develop visibility culling algorithms in the context of architectural models. A long line of research pursues visibility techniques for urban environments, culminating with the approach of Leyvand et al. [2003].

Point-based systems [Rusinkiewicz and Levoy 2000; Wand et al. 2001] are able to handle huge models in real time by sampling points from the surface of the models. Rusinkiewicz and Levoy [2001] develop view-dependant progressive transmission that adapts point-based methods for remote rendering. There are many advantages to a point-based representation, but one important drawback is that the average depth complexity can grow linearly. We have elected to use depth images to avoid this increase in depth complexity.

Remote rendering. Remote rendering [Schmalstieg 1997] has been used extensively to render very large datasets. As model size increases, it becomes more efficient to render images near the data on the server, and transfer images rather than models over a network. Remote rendering also leverages widely available commodity video streaming codecs. A disadvantage is that interactive applications will have additional latency. Mann and Cohen-Or [1997] and Yoon and Neumann [2000] present image-based approaches to remote rendering, in which the server actively assists each client with the rendering task. These approaches do not meet our design goals due to potentially high loads on the server, which in our target setting must be able to support very large numbers of concurrent participants.

Networked virtual environments. Singhal and Zyda [1999] review the state of the art in networked virtual environments circa 1999. Much of the earlier work focused on networking issues in virtual world systems [Funkhouser 1995; Macedonia et al. 1995], with emphasis on how to efficiently transfer updates to a large number of participants using multicast and how to reduce latency using

Viewer speed (m/s)	Nearby geometry (Mbps)	Depth images (Mbps)	Image vs. geometry costs
2	7.6	0.9	11.4%
5	19.5	1.8	9.4%
10	39.6	3.0	7.7%
15	59.5	4.3	7.2%
20	79.5	5.5	6.9%
40	159.5	10.7	6.7%

Table 1: *Amount of bandwidth required to maintain nearby geometry vs. depth images. The bandwidth was averaged over three 400-second test runs at each of the listed movement speeds. Depth images require an order of magnitude less bandwidth.*

dead reckoning. Schmalstieg and Gervautz [1996] and Teler and Lischinski [2001] consider bandwidth-limited remote walkthroughs and develop heuristics to guide the transmission of geometry.

9 Discussion and Future Work

In this paper we have applied image-based rendering to the display of large virtual worlds. Our approach is to use a hierarchial RGBZ representation to approximate the distant parts of the world. The key result is that any one client need only access a number of depth samples that is proportional to its image resolution times a logarithmic factor in the size of the world, and that this set of samples can be kept current under a constant bandwidth cap independent of the size of the world. Applying this approach directly (with no environment map compositing to reduce the rendering load) requires over two hundred million points for our test scene. Since this number grows only logarithmically in the size of the world, the entire planet Earth would require about one billion points. Although this number of points is beyond what current graphics hardware can render in real-time, and what can be delivered over consumer Internet connections, we are not far. Once GPUs and networks cross this performance threshold these techniques will become forever practical.

There are, however, two algorithmic ways the system could be improved. First, we would like to absolutely guarantee that the environment is optimally sampled so that no holes will ever appear in the reprojected depth images. We believe a combination of depth images and points might provide this guarantee at little additional cost. Second, the overall depth complexity is still quite high. We could reduce the depth complexity significantly by knowing more precisely when we could reduce the number of views, and by using some form of occlusion culling.

Finally, we have built a prototype of our system that runs on a small cluster with a few clients to demonstrate the feasibility of the methods we have developed. We are actively working towards scaling our system to much larger configurations. We believe that participating with others in a visually rich virtual world will significantly affect one's sense of presence in the space.

References

AIREY, J. M., ROHLF, J. H., AND FREDERICK P. BROOKS, J. 1990. Towards image realism with interactive update rates in complex virtual building environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 41–50.

AKAMAI TECHNOLOGIES, INC. 2008. `http://www.akamai.com`.

ALIAGA, D. G., AND LASTRA, A. 1999. Automatic image placement to provide a guaranteed frame rate. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 307–316.

ALIAGA, D., COHEN, J., WILSON, A., BAKER, E., ZHANG, H., ERIKSON, C., HOFF, K., HUDSON, T., STUERZLINGER, W., BASTOS, R., WHITTON, M., BROOKS, F., AND MANOCHA, D. 1999. Mmr: an interactive massive model rendering system using geometric and image-based acceleration. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 199–206.

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 587–594.

BAINBRIDGE, W. S. 2007. The scientific research potential of virtual worlds. *Science 317*, 472–476.

BOSSE, M., NEWMAN, P., LEONARD, J., AND TELLER, S. 2004. Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *The International Journal of Robotics Research 23*, 12, 1113–1139.

CHANG, C.-F., BISHOP, G., AND LASTRA, A. 1999. Ldi tree: a hierarchical representation for image-based rendering. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 291–298.

CHEN, S. E., AND WILLIAMS, L. 1993. View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 279–288.

CLARK, J. H. 1976. Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10, 547–554.

COHEN-OR, D., CHRYSANTHOU, Y., SILVA, C., AND DURAND, F. 2003. A survey of visibility for walkthrough applications. *Transactions on Visualization and Computer Graphics 9*, 3, 412–431.

DARSA, L., SILVA, B. C., AND VARSHNEY, A. 1998. Walkthroughs of complex environments using image-based simplification. *Computers and Graphics 22*, 1, 55–69.

DEAN, J., AND GHEMAWAT, S. 2004. Mapreduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation*, 137–150.

DECORET, X., SILLION, F., SCHAUFLER, G., AND DORSEY, J. 1999. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum 18*, 3, 61–73.

FUNKHOUSER, T. A. 1995. Ring: a client-server system for multi-user virtual environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 85–ff.

GOOGLE INC. 2008. *Google 3D Warehouse*. `http://sketchup.google.com/3dwarehouse`.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 43–54.

JESCHKE, S., WIMMER, M., SCHUMANN, H., AND PURGATHOFER, W. 2005. Automatic impostor placement for guaranteed frame rates and low memory requirements. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 103–110.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 31–42.

LEYVAND, T., SORKINE, O., AND COHEN-OR, D. 2003. Ray space factorization for from-region visibility. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 595–604.

LISCHINSKI, D., AND RAPPOPORT, A. 1998. Image-based rendering for non-diffuse synthetic scenes. In *Proceedings of the Ninth Eurographics Workshop on Rendering*, 301–314.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph. 23*, 3, 769–776.

LUEBKE, D., WATSON, B., COHEN, J. D., REDDY, M., AND VARSHNEY, A. 2002. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New

York, NY, USA.

MACEDONIA, M. R., BRUTZMAN, D. P., ZYDA, M. J., PRATT, D. R., BARHAM, P. T., FALBY, J., AND LOCKE, J. 1995. Npsnet: a multi-player 3d virtual environment over the internet. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 93–ff.

MANN, Y., AND COHEN-OR, D. 1997. Selective pixel transmission for navigating in remote virtual environments. *Computer Graphics Forum 16*, 3, C201–C206.

MARK, W. R., MCMILLAN, L., AND BISHOP, G. 1997. Post-rendering 3d warping. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 7–ff.

MAX, N., AND OHSAKI, K. 1995. Rendering trees from precomputed Z-buffer views. In *Proceedings of the Eurographics Workshop on Rendering*, 74–81.

MAX, N., DEUSSEN, O., AND KEATING, B. 1999. Hierarchical image-based rendering using texture mapping hardware. In *Proceedings of the Eurographics Workshop on Rendering*, 57–62.

MCMILLAN, L. 1997. *An image-based approach to three-dimensional computer graphics*. PhD thesis, Chapel Hill, NC, USA.

MILLER, G. 2007. The promise of parallel universes. *Science 317*, 1341–1343.

RAFFERTY, M. M., ALIAGA, D. G., POPESCU, V., AND LASTRA, A. A. 1998. Images for accelerating architectural walkthroughs. *IEEE Comput. Graph. Appl. 18*, 6, 38–45.

REGAN, M., AND POSE, R. 1994. Priority rendering with a virtual reality address recalculation pipeline. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 155–162.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 343–352.

RUSINKIEWICZ, S., AND LEVOY, M. 2001. Streaming qsplat: a viewer for networked visualization of large, dense models. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 63–68.

SAMET, H. 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.

SCHAUFLER, G., AND STÜRZLINGER, W. 1996. A three dimensional image cache for virtual reality. *Computer Graphics Forum 15*, 3, 227–236.

SCHMALSTIEG, D., AND GERVAUTZ, M. 1996. Demand-driven geometry transmission for distributed virtual environments. *Computer Graphics Forum 15*, 3, 421–431.

SCHMALSTIEG, D. 1997. *The Remote Rendering Pipeline*. PhD thesis.

SHADE, J., LISCHINSKI, D., SALESIN, D. H., DEROSE, T., AND SNYDER, J. 1996. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 75–82.

SHADE, J., GORTLER, S., WEI HE, L., AND SZELISKI, R. 1998. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 231–242.

SILLION, F., DRETTAKIS, G., AND BODELET, B. 1997. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum 16*, 3, C207–C218.

SINGHAL, S., AND ZYDA, M. 1999. *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

SUDARSKY, O., AND GOTSMAN, C. 1997. Output-senstitive rendering and communication in dynamic virtual environments. In *VRST '97: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 217–223.

TANNER, C. C., MIGDAL, C. J., AND JONES, M. T. 1998. The clipmap: a virtual mipmap. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 151–158.

TELER, E., AND LISCHINSKI, D. 2001. Streaming of complex 3D scenes for remote walkthroughs. In *Eurographics*, A. Chalmers and T.-M. Rhyne, Eds., vol. 20(3). Blackwell Publishing, 17–25.

TELLER, S. J., AND SÉQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. *SIGGRAPH Comput. Graph. 25*, 4, 61–70.

THRUN, S., AND MONTEMERLO, M. 2005. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research 25*, 5-6, 403–430.

WAND, M., FISCHER, M., PETER, I., AUF DER HEIDE, F. M., AND STRASSER, W. 2001. The randomized z-buffer algorithm: interactive rendering of highly complex scenes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 361–370.

WILLIAMS, L. 1983. Pyramidal parametrics. *SIGGRAPH Comput. Graph. 17*, 3, 1–11.

WILSON, A., AND MANOCHA, D. 2003. Simplifying complex environments using incremental textured depth meshes. *ACM Trans. Graph. 22*, 3, 678–688.

YOON, I., AND NEUMANN, U. 2000. Web-based remote rendering with IBRAC. *Computer Graphics Forum 19*, 3.