Simplifying Complex Environments using Incremental Textured Depth Meshes

Andy Wilson Dinesh Manocha

http://gamma.cs.unc.edu/ITDM

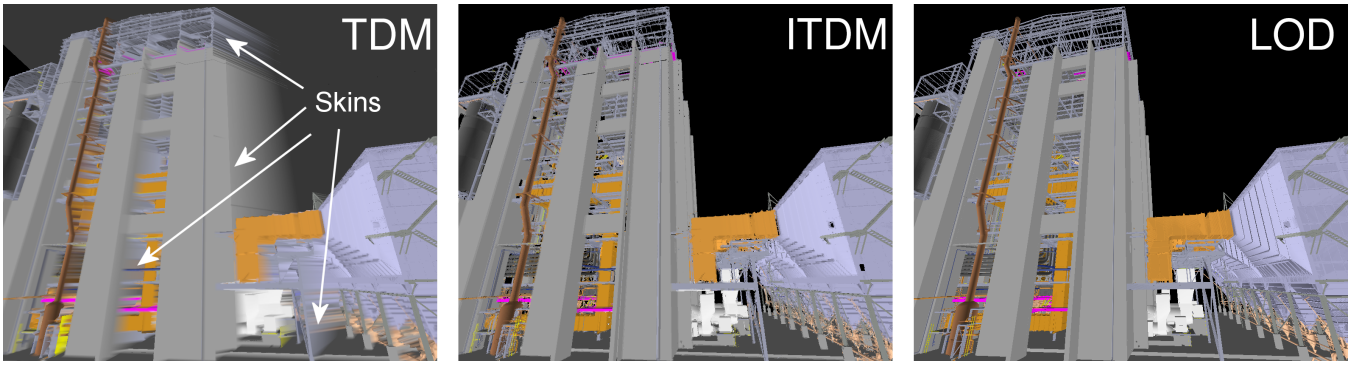


Figure 1: *Comparison of our method (ITDMs) with regular TDMs and geometric LODs on a 12.5M polygon power plant model. ITDMs generate images almost as good as static LODs at a frame rate 9 times faster. Moreover, ITDMs do not show the skin artifacts common in TDMs.*

Abstract: *We present an incremental algorithm to compute image-based simplifications of a large environment. We use an optimization-based approach to generate samples based on scene visibility, and from each viewpoint create textured depth meshes (TDMs) using sampled range panoramas of the environment. The optimization function minimizes artifacts such as skins and cracks in the reconstruction. We also present an encoding scheme for multiple TDMs that exploits spatial coherence among different viewpoints. The resulting simplifications, incremental textured depth meshes (ITDMs), reduce preprocessing, storage, rendering costs and visible artifacts. Our algorithm has been applied to large, complex synthetic environments comprising millions of primitives. It is able to render them at* $20-40$ *frames a second on a PC with little loss in visual fidelity.*

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]:Picture/Image Generation-Display algorithms; I.3.7 [Computer Graphics]: Three-dimensional Graphics and Realism

Keywords: interactive display, simplification, textured-depth meshes, spatial encoding, walkthrough

1 Introduction

Image-based impostors are frequently used to accelerate the rendering of complex models. They are widely used in computer gaming, flight simulators, and interactive display of architectural models, urban environments, or real-world datasets. Different image-based representations such as texture maps, textured depth meshes, and depth images are widely used to drastically simplify portions of the scene that are far from the current viewpoint. The time required to render these representations is mostly a function of image-space resolution and is independent of scene complexity. Many algorithms and systems have been proposed that approximate portions of environments with image-based impostors for faster display [Aliaga et al. 1999; Aliaga 1996; Aliaga and Lastra 1999; Chang et al. 2001; Darsa et al. 1998; Decoret et al. 1999; Jeschke and Wimmer 2002; Maciel and Shirley 1995; Schaufler and Sturzlinger 1996; Shade et al. 1998; Shade et al. 1996; Sillion et al. 1997; Wilson et al. 2001].

Image-based simplifications can have a number of visual artifacts, including dis-occlusions, cracks, tears, poor parallax effects, and skins. Parallax effects can be added to flat images by triangulating the depth values associated with the environment map from each viewpoint. The resulting triangles can be warped using texture mapping hardware [Sillion et al. 1997; Darsa et al. 1998; Decoret et al. 1999; Aliaga et al. 1999]. Moreover, Z-buffer hardware can be used for resolving occlusions and rendering interactively. However, the resulting textured depth mesh (TDM) representations can still show *skins* and *cracks* around depth discontinuities where information is missing. *Skins* are surfaces in a TDM that are not present in the original environment. The left image in Fig. 1 shows an example of the smearing artifacts caused by skins. Removing skins leaves *cracks* in an image where nothing is known about visible surfaces. Fig. 11 shows an example. Skins and cracks arise because image-based samples contain information about only the nearest surface visible from the sample viewpoint. Since no information is available about surfaces occluded from that viewpoint, they cannot be properly rendered; instead, artifacts appear whenever those surfaces should be visible.

Many of these problems can be alleviated by generating more samples from different viewpoints. However, this increases the costs of pre-processing, storage, and runtime rendering, becoming intractable for large environments. Ideally, we want to capture most of the visible surfaces in an environment using as few samples as possible. This problem is closely related to the *art gallery* problem in computational geometry [O'Rourke 1997], though our definition for a "good" sample is different. The classical art-gallery problem is NP-complete. As a result, there is little hope of computing an optimal solution to the sampling problem. Given large environments with uneven distributions of primitives, no good approximations or practical algorithms are known for computing viewpoints for image-based samples of the environment.

Overview: Our goal is to construct image-based simplifications of parts of a large, complex environment. We partition the scene into regions and compute simplifications for each region separately. The simplifications consist of a number of image-based samples acquired from viewpoints within the region. We build impostors from these samples, then use those impostors at runtime to replace primitives outside the navigable region. This approximation trades the widely varying cost of rendering the original geometric environment for the constant, bounded cost of rendering from image-based samples. We use a Voronoi-based sampling algorithm to acquire potentially visible surfaces using a small number of samples, given criteria for measuring error in the impostors.

We build a set of *incremental textured depth meshes* (ITDMs) by spatially encoding the samples in an image-based simplification to replace many points with few triangles. This encoding exploits the spatial relationships among samples to remove points that belong to surfaces that have already been captured. By reducing the number of points that must be processed, we reduce preprocessing and storage costs as well as improve rendering speed. We create polygonal meshes from the reduced samples and apply view-dependent simplification to the results. The output of this algorithm is a set of ITDMs that can be rendered in place of the original prim-

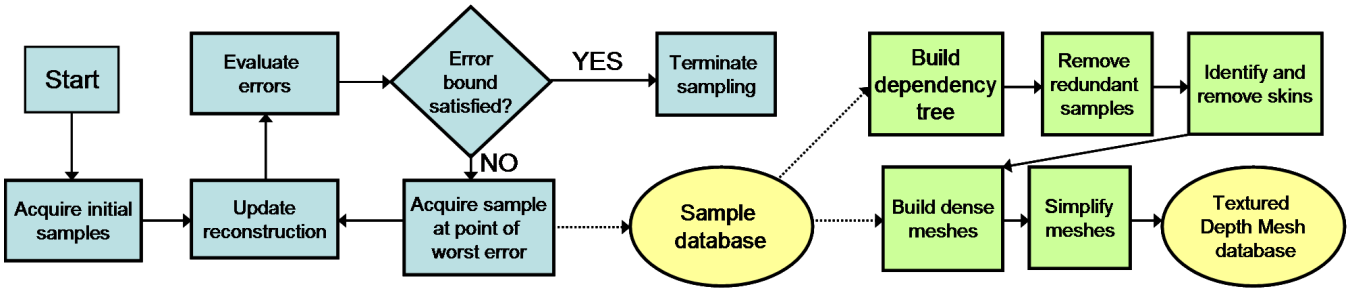


Figure 2: *Overview of our two-phase simplification approach. Stages of Voronoi-based sampling are shown in blue. Incremental TDM construction is shown in green.*

itives captured by the simplification.

New Results: Our approach includes the following three novel algorithms.

1. An optimization-based incremental algorithm that generates panoramic range samples within a region. This algorithm reduces the severity of visibility artifacts.
2. An algorithm for detecting skins and other visibility artifacts that appear in TDMs and a measure of their severity.
3. A new spatial encoding technique that reduces redundancy among different range samples and represents ITDMs incrementally using a dependency tree.

Our current sample generation and rendering algorithm is targeted for walkthrough applications where the user's motion generally lies in a plane and allows for translation and rotation. Different components of our approach are shown in Figure 2.

We have applied our algorithm to two complex architectural environments. We take portions of the scene outside several different navigable regions with sizes between 2 and 30 meters square in the powerplant model. The simplification for each region replaces $1,185$ objects comprising 12.5 million polygons with $5-18$ ITDMs created with a maximum error threshold of 0.2 steradians. The 405 ITDMs take about 1.4GB of space as compared to 660 MB for the original environment. We are able to render them at $20-40$ frames per second on a PC with an NVIDIA GeForce 4 graphics card in immediate mode. We have also performed preliminary comparisons with two other algorithms for rendering large datasets. The use of ITDMs improves the frame-rate by an average factor of 9 over an algorithm that uses geometric levels-of-detail, with little loss in image quality. Our approach results in fewer visual artifacts as compared to previous algorithms based on TDMs. Overall, ITDMs reduce pre-processing time, storage overhead, and rendering cost and result in better simplifications.

Organization: We briefly survey related work in section 2. In section 3 we present the Voronoi-based incremental algorithm for capturing a set of samples in an environment. In section 4 we use these samples to construct incremental textured depth meshes. In section 5 we highlight the performance of our approaches together and compare them with earlier methods. We analyze our approach and highlight some of its limitations in Section 6. We conclude in Section 7.

2 Prior Work

2.1 Interactive Display of Large Models

Many rendering acceleration techniques based on geometric levels of detail (LODs) or visibility culling have been proposed for large, complex environments. Several different LOD computation algorithms for polygonal models are surveyed in [Luebke et al. 2002]. Many of the algorithms reduce the problem to an optimization algorithm and track simplification error using geometric measures such as surface deviation, texture deviation, Hausdorff distance, and appearance attributes [Hoppe 1997; Garland and Heckbert 1997; Luebke and Erikson 1997]. In addition to these geometric measures, LOD algorithms have been proposed that use image-space error measures such as RMS error [Lindstrom and Turk 2000] and perceptually based contrast sensitivity functions [Luebke et al. 2002].

LOD algorithms work well in environments composed of fine tessellations of smooth surfaces. Environments with high depth complexity are more difficult: because LODs do not actually remove surfaces from an environment, they will not help in situations where an application is fill-bound. Similarly, environments with many coarsely tessellated objects are a difficult case for LODs since a drastic simplification of the environment can result in large screen-space errors.

By contrast, visibility culling algorithms accelerate rendering by culling away a subset of primitives that are not visible from the current viewpoint. A recent survey is given in [Cohen-Or et al. 2001]. While good algorithms have been proposed for architectural and urban settings, current visibility culling methods for general environments either need special hardware [Greene et al. 1993], perform approximate culling [El-Sana et al. 2001], or require multiple graphics cards and introduce additional latency in the system [Baxter et al. 2002].

2.2 Image-Based Representations

Image-based representations and impostors accelerate rendering by providing a drastic simplification of distant geometry. For example, many flight simulation systems use images to represent terrains and other specialized models. Common image-based representations include point primitives, flat images, textured depth meshes, and depth images. Point primitives [Pfister et al. 2000; Rusinkiewicz and Levoy 2000] work well for over-sampled datasets. Flat images [Aliaga 1996; Debevec et al. 1998; Maciel and Shirley 1995; Shade et al. 1996; Schaufler and Sturzlinger 1996] are mapped onto planar projections, but only display correct perspective when viewed from the location where the image was created. Textured depth meshes (TDMs) [Aliaga et al. 1999; Darsa et al. 1998; Jeschke and Wimmer 2002; Sillion et al. 1997; Wilson et al. 2001] replace the planar projections used by flat images with simplified meshes created from sampled depth values. TDMs are extended to handle multiple layers in [Decoret et al. 1999]. TDMs are height fields, and many algorithms have been proposed for simplification and view-dependent level-of-detail control of such datasets [Luebke et al. 2002]. However, no good algorithms are known for automatic sampling and TDM generation in large environments.

Depth images [Aliaga and Lastra 1999; Max and Ohsaki 1995; McMillan and Bishop 1995] are rendered at runtime using 3D image warping. Layered depth images [Shade et al. 1998] have been proposed as an extension to reduce disocclusion artifacts at the cost of increased storage overhead. The geometric portion of a TDM can be thought of as a polygonal encoding of a depth image. Depth images may require special-purpose hardware or multiprocessor systems for fast display. Many algorithms generate triangulations from depth images, simplify the meshes and render the resulting data [Nyland et al. 2001]. Overall, image-based representations can either be precomputed from a set of viewpoints or dynamically updated at runtime [Decoret et al. 1999; Schaufler and Sturzlinger 1996; Shade et al. 1996].

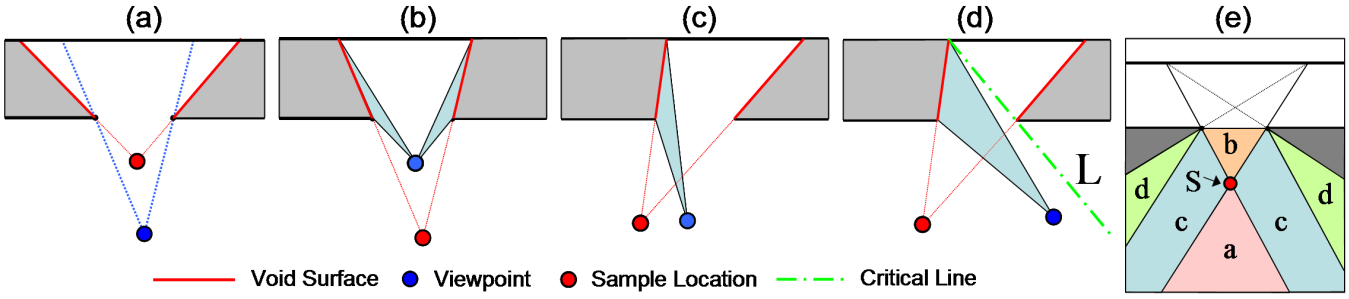


Figure 3: *The visibility of the void surface in this simple environment falls into one of four categories depending on the relative placement of a sample location S and a viewpoint V. Gray regions show the void volume. Shaded blue regions show the projected extent of the void surface with respect to the viewpoint. Part (e) summarizes the characteristics according to the placement of a viewpoint with respect to a sample location. Labeled, colored regions correspond to the cases in parts (a) through (d). Unlabeled gray regions are those parts of case (d) where the viewpoint passes beyond the critical line L and the rear wall is not visible at all.*

2.3 Sampling Large Environments

Most earlier algorithms for generating image-based samples of large environments choose viewpoints uniformly using regular grids [Shade et al. 1996; Wilson et al. 2001] or model-dependent characteristics [Aliaga et al. 1999]. Another approach is to generate samples adaptively to guarantee a constant frame rate [Aliaga and Lastra 1999]. Artifacts arising from the use of image-based impostors are characterized in [Decoret et al. 1999]. The authors also describe an algorithm to improve the quality of TDMs by measuring the size of *visibility events*. This improvement is computed in $2\frac{1}{2}D$ urban environments in which viewing regions are restricted to 1D line segments.

The problem of sample acquisition in a navigable region is addressed in [Fleishman et al. 2000]. The authors subdivide polygons in a scene into very small patches, place many cameras around the border of the navigable region, and choose a subset of those cameras that collectively see most of the visible polygons. An alternate approach based on visibility regions is given in [Sturzlinger 1999].

Issues of sampling large environments also arise in the "art gallery" and "next best view" problems in computational geometry, computer vision, reverse engineering, and robot motion planning. In the art gallery problem the task is to position optimally a set of guards that collectively see an entire environment [O'Rourke 1997]. The art gallery problem assumes complete knowledge of scene geometry. The optimal-placement problem is NP-complete. Global visibility algorithms such as the ones based on aspect graphs can compute all visibility events. Based on that information, we can compute sample locations that capture all visible surfaces. However, computation of aspect graphs in 3D can take as long as $O(n^9)$, where n is the number of primitives in the scene.

Most sampling problems in reverse engineering and computer vision assume that object and scene geometry are initially unknown. The goal is to construct a model using as few observations as possible. The *best-next-view* problem addresses the issue of finding a viewpoint where a new observation will add the most information to a partially captured environment. A survey of prior work on this problem is given in [Pito 1999]. Most best-next-view algorithms operate by identifying depth discontinuities in range images or in the model under construction and reasoning about the visibility and occlusion of such discontinuities [Banta et al. 1995; Maver and Bajcsy 1993; Reed and Allen 1999]. Such methods use local techniques to generate additional samples, sometimes resulting in large databases of many samples. Some algorithms for automatic scene acquisition have combined best-next-view computation with robot motion planning [Gonzalez-Banos and Latombe 1998; Gonzalez-Banos and Latombe 2001; Reed and Allen 1999]. These algorithms also consider the cost of computing a collision-free motion needed to reach the next viewpoint. Such constraints do not arise in our application that deals with synthetic environments.

Our criteria for generating samples are based on minimizing the reconstruction error visible from the navigable volume. Unlike the art gallery and global visibility computation algorithms, we do not attempt to guarantee capture of all visible surfaces or events.

2.4 Incremental Encodings

The notion of incremental representation of a data set is common in video and geometry compression. Schemes such as MPEG store only one out of every N frames in its entirety. The intervening frames are represented as differences from previous frames. A similar technique has been described for polygon-assisted JPEG and MPEG compression of synthetic images [Levoy 1995]. A spatial video encoding scheme that represents a 3D space of images instead of a 1D temporal sequence is presented in [Wilson et al. 2001]. An approach to compressing a space of images using spanning trees is presented in [Aliaga et al. 2002]. Incremental encodings have been proposed for progressive simplifications of polygonal models [Hoppe 1997; Luebke et al. 2002]. Instead of image-to-image differences, these approaches store trees of decimation operations like edge-collapses to transform one mesh to another. A hierarchical image compression scheme for multi-layer image-based rendering system is presented in [Chang et al. 2001]. [Jeschke and Wimmer 2002; Jeschke et al. 2002] have proposed incremental construction schemes for layered image-based impostors.

3 Voronoi-Based Sampling

In this section we describe our Voronoi-based sampling algorithm that is used to generate panoramic samples of an environment. We partition an environment into regions and compute samples for each region separately. We reduce sample generation to an optimization problem and use properties of the void surface in formulating the objective function.

3.1 Problem Definition

Our goal is to capture potentially visible surfaces using a minimum number of samples given some criterion for error in the impostors. We introduce some of the terms that are used in the rest of the paper:

The potentially visible set (PVS) contains all surfaces visible from some navigable region $\mathcal{N}$.

The navigable region, $\mathcal{N}$, is a region of free space containing all possible viewpoints for a particular simplification. We simplify the part of the scene outside $\mathcal{N}$.

A sample is a panoramic environment map plus per-pixel depth and camera information. We represent a sample as six range images arranged as a cube environment map.

A sample location is the viewpoint from which a sample is acquired.

In the rest of this section, we will assume that we are dealing with rectangular 2D regions, as is appropriate for a person moving about at constant eye height, though the algorithm works for 3D. We characterize the *error* introduced by our lossy simplification scheme according to the severity of the artifacts – skins and cracks, described in Section 1 – introduced during reconstruction.

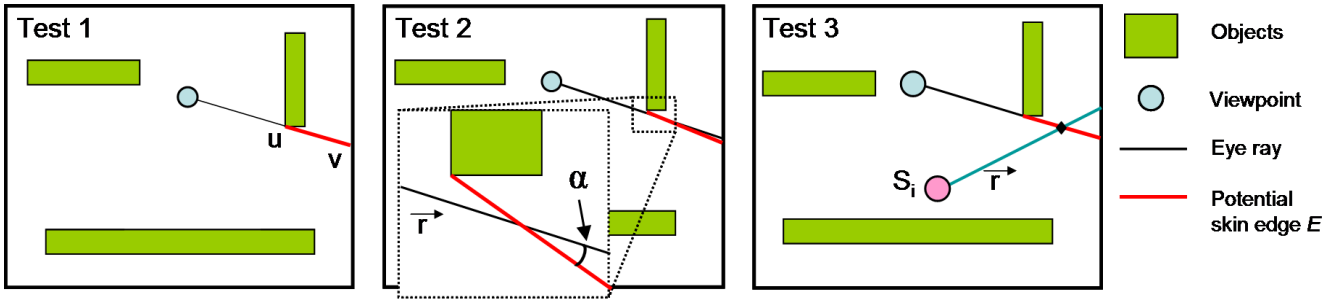


Figure 4: *Skin identification criteria. We classify an edge E as a skin if either endpoint is at maximum depth (Test 1), the angle between E and a ray $\vec{r}$ from the viewpoint through its midpoint falls below some threshold α (Test 2), or if some other sample location S_i can see past the midpoint of E to some more distant surface (Test 3).*

3.1.1 Decomposition into Regions

The separable problem of decomposing a large environment into smaller viewing regions is discussed in [Reference suppressed]. We use the simple approach of partitioning the environment with an octree until cells contain no more than a specified number of polygons. Each cell is treated as a separate region. Each region's simplification contains all objects that are not completely inside that region.

3.2 Simplification as Greedy Optimization

Given a region, our goal is to generate samples that minimize the visibility artifacts. We compute viewpoints for the samples in an image-based simplification using a greedy, incremental optimization strategy. This strategy has four major components:

1. An *objective function* measuring the reconstruction error to be minimized
2. A set of *initial conditions* specifying a few initial samples
3. A set of *termination criteria* to judge when the simplification is "good enough"
4. An *update phase* to choose the location for the next sample

3.3 Objective Function

The objective function measures the severity of reconstruction errors with respect to a set of samples and viewpoints. Rather than identifying these errors by comparing the reconstructed image with an image of the original environment, we identify regions of space where we have little or no information and quantify the visibility of those regions. We first describe a subdivision of space that allows us to construct the boundary of such regions. Next we present the operations necessary to compute this subdivision (and thus the objective function) from a set of samples.

3.3.1 The Void Surface

The void surface, described in [Pito 1999], forms the boundary between space visible from one or more sample locations (the *visible volume*) and space about which nothing is known (the *void volume*) Fig. 3(a) shows an example. The red point is a sample location. Red line segments show parts of the void surface, while shaded gray area belongs to the void volume. Since artifacts appear when we attempt to render surfaces within the void volume, we can approximate their severity by quantifying the visibility of the void surface. We are interested in the *global void surface* that obscures space not seen by any one of a group of samples. We will build up an approximation of the global void surface using the individual void surfaces defined by each single sample.

3.3.2 Characteristics of the Void Surface

Although the exact behavior of the void surface is highly scene-dependent, we claim that it will tend to become more visible as a viewpoint V moves farther away from a sample location S. We support this claim with examples drawn from a simple environment shown in Fig. 3. This environment has exactly two surfaces (one rear wall and one front wall). The front wall is pierced by a gap. The visibility of the void surface (which can only be seen through the gap) can be assigned to one of the following four cases according to the relative placement of V and S:

- The void surface is not visible at all. (Fig. 3(a))
- The void surface is visible on both sides of the gap. The rear wall will always be visible.(Fig. 3(b))
- The void surface is visible on one side of the gap. The rear wall will always be visible. (Fig. 3(c))
- The void surface is visible on one side of the gap. If the viewpoint passes beyond a critical line L, nothing except the void surface is visible beyond the gap. (Fig. 3(d))

This behavior will help us compute candidate sample locations for the next sample in our optimization. A new sample will contribute information wherever it sees surfaces obscured by the void surface. Viewpoints that see large areas that are covered by the void surface (e.g. viewpoints beyond L in Fig. 3(d)) will contribute more information than those in situations such as Fig. 3(a).

3.3.3 The Void Surface in a Sample

We can compute a close approximation to the void surface within a single sample by triangulating the depth buffer as a height field. This is the process that gives rise to skins in textured depth meshes. In fact, those skins make up exactly the surface we want to identify: they form a boundary between visible volume (surfaces and free space sampled by the TDM) and void volume (space occluded by the surfaces visible in a TDM). We will use skins as an approximation of the void surface within a sample.

We identify skins by computing per-pixel adjacency within a captured depth buffer. If two pixels u_{screen} and v_{screen} are not part of the same surface, the edge E between the corresponding world-space points u_{world} and v_{world} does not correspond to any surface in the environment. We detect such edges using the following three tests, illustrated in Fig. 4.

1. If either u_{screen} or v_{screen} has a depth value beyond some threshold, the edge is a skin.
2. Construct the view ray $\vec{r}$ from the sample location through the midpoint of the edge E. If the angle between $\vec{r}$ and E is less than some threshold α, the edge is a skin.
3. For each sample S_i, construct a ray $\vec{r}$ from S_i's viewpoint through the midpoint of E. If the first intersection between $\vec{r}$ and a surface visible from S_i is beyond the midpoint of E, the edge is a skin.

We compute the adjacency information for all eight neighbors of each pixel of each face in each sample. The void surface for a sample is approximated by constructing triangles that cover the discontinuity between any two pixels that are not adjacent.

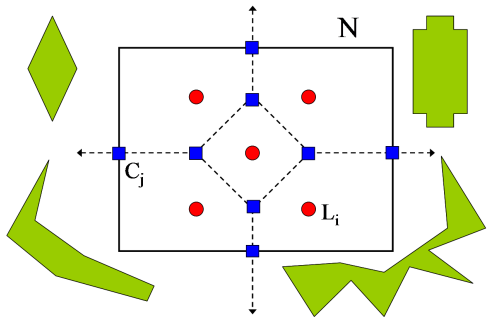


Figure 5: *Candidates for the next sample location (blue squares) are constructed from the Voronoi diagram of existing sample locations (red circles).*

3.3.4 Approximating the Global Void Surface

In the previous section we used skins to identify the void surface within a single sample. However, we are interested in the *global* void surface defined by a group of samples. The global void volume contains all space not observed by any sample in the group. Any point in the global void volume must therefore be occluded by the void surfaces for each sample. This allows us to describe the visibility of the global void surface as follows. Let $Skins(S_i)$ be the void surface for sample S_i, $\mathbb{V}_{global}$ be the global void surface, and $Projection(surf, V)$ be the screen-space projection of the visible portion of some surface $surf$ from viewpoint V. Given a group of samples $S_1 \ldots S_n$ and a viewpoint V, we have:

$$Projection(\mathbb{V}_{global}, V) = \bigcap_{i=1}^{n} Projection(Skins(S_i), V).$$

We compute $Projection(\mathbb{V}_{global}, V)$ using the graphics hardware. First, we render the valid surfaces in each sample S_i. We then compute $Projection(Skins(S_i))$ by rendering the skins for sample S_i into the same buffer as these valid surfaces. We use the stencil buffer to accumulate the screen-space intersection of these individual void surfaces.

3.3.5 Computing Projected Area and Objective Function

The algorithm described in the previous section identifies pixels that belong to the visible extent of the global void surface. Recall that the value of the objective function at some point V is the magnitude of the solid angle subtended by the portion of the global void surface visible from V. We compute this by adding up the solid angles subtended by each pixel in $Projection(\mathbb{V}_{global}, V)$. If we treat a single pixel as a spherical polygon with interior angles $\theta_1 \ldots \theta_4$, the function $Area(p)$ yields the solid angle subtended by a single pixel p:

$$Area(p) = (\sum_{i=1}^{4} \theta_i) - 2\pi.$$

Given $Area(p)$ for each pixel, the overall objective function F for a viewpoint V and a set of samples, $S_1 \ldots S_n$, is defined in terms of the solid angle subtended by all pixels belonging to the global void surface. Let $\mathcal{G} = Projection(\mathbb{V}_{global}, V)$. We then have

$$F(V, S_1 \ldots S_n) = \sum_{p \in \mathcal{G}} Area(p).$$

As we acquire more samples, n increases and $F()$ decreases. The goal of our optimization is to keep acquiring new samples S_i until the objective function $F()$ falls below some threshold.

3.4 Starting and Ending Criteria

In order to capture a large part of the potentially visible set with a few samples and thus minimize the required number of simplification steps, we begin with one sample taken from each of the four corners of the navigable region. The optimization process continues until the value of the objective function falls below some user-specified threshold ϵ.

3.5 Finding the next sample location

At each step of the optimization, we compute the location for the next sample to be acquired. We begin with the initial samples from the corners of the navigable region and add samples in a greedy manner.

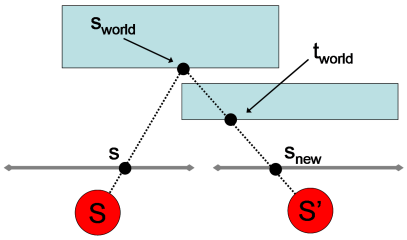


Figure 6: *Identifying redundant points in a sample. Red points are sample locations, gray lines are projection planes for those samples, and blue rectangles are objects in the world.*

3.5.1 Candidate locations for the next sample

Ideally, we would like the next sample viewpoint to be at the point in the navigable region where the objective function attains its maximum. Since the objective function is scene-dependent, with many local maxima and minima, finding this point exactly can be prohibitively expensive in complex environments. We instead search a limited set of candidate viewpoints $C_1, C_2, \ldots, C_n$ to find the next sample location.

We construct these candidate viewpoints using the assumption that the visible extent of the void surface increases as the distance from a viewpoint to the existing sample locations increases. We use the 2D Voronoi diagram of previous sample locations to find points as far from any sample location as possible. Since the Voronoi diagram can extend beyond the navigable region, we choose the actual candidate viewpoints $C_1 \ldots C_j$ by collecting all of the Voronoi vertices that fall within the navigable region and by finding the intersections between the boundary of the navigable region and the edges of the Voronoi diagram. See Fig. 5 for an example.

We evaluate the objective function (the visible extent of the void surface) at each candidate viewpoint C_j. The point at which the void surface subtends the largest solid angle is chosen as the viewpoint location for the next sample of the environment.

The simplification process ends when the maximum detected visibility error falls below a user-specified threshold. The panoramic samples constructed by the simplification process will be used to compute a corresponding set of impostors.

4 Incremental Textured Depth Meshes

The image-based simplification method presented in Section 3 replaces the parts of a complex environment outside some navigable region with a set of panoramic samples. In this section we spatially encode those samples to construct an incrementally represented set of textured depth meshes.

The process of constructing ITDMs can be seen as a lossy encoding of range data. Although a simplified polygonal mesh approximates most or all of the points in an input sample, it is not possible to recover those points exactly. We only address mesh encoding in this paper. The image portion of a TDM can be separately encoded using JPEG or image-based spatial encoding algorithms [Aliaga et al. 2002; Wilson et al. 2001].

4.1 Reducing Redundancy in a Group of Samples

The first stage of ITDM construction is to identify and remove redundant information within a group of samples. This removal takes place by identifying and deleting points in a sample S_0 that belong to surfaces visible in some other set of samples $S_1, S_2, \ldots, S_k$. This introduces a dependency between samples: if surfaces present in a sample S_1 are removed from a sample S_0, then the meshes created from S_0 depend on the meshes created from S_1 to fill in gaps. We refer to the meshes created from a single sample as a *scan cube*.

We first identify a single sample within a group to serve as the *intra sample*. By definition, the intra sample does not depend on any other sample: all of its surfaces are considered non-redundant. All other samples in the group are designated *delta samples*, as they will store only the difference (delta) between their potentially visible sets. To minimize the amount of information present in the delta samples, we want to choose an intra sample I that maximizes the intersection between its PVS and those of the delta samples. We

choose the sample nearest the centroid of all sample locations in a group to be the intra sample.

4.1.1 Building a Dependency Tree

The dependency tree D for a group of samples is rooted at the intra sample and incorporates all of the delta samples as vertices. We build it by first constructing a complete graph with one node for each sample in the group. The edge between vertices i and j in the graph is assigned a weight proportional to the cubed distance between the sample locations for samples S_i and S_j. D is then computed as a minimum spanning tree rooted at the intra sample S_{intra}. By choosing the cubed distance as the edge weight, we favor spanning trees where a particular node's parent or child nodes are close to it in space.

4.1.2 Identifying Redundant Points

The dependency tree yields a partial order on the set of samples that allows us to compute the differences in delta samples. We remove from each delta sample all points that are present in surfaces in any of that delta sample's ancestors, up to and including the intra sample. Our redundancy test only considers whether or not a given point is present in a surface captured in a given sample. The test for a sample element s within a sample S (illustrated in Fig. 6) is as follows:

1. Transform the screen-space point s to its world-space location s_{world}.
2. Project s_{world} into the screen-space point s_{new} in sample S'.
3. Find the element t at screen-space coordinates s_{new} in sample S'.
4. Transform t into the world-space point t_{world}.
5. If s_{world} and t_{world} are within some distance ϵ, sample element s is redundant.

The equality tolerance ϵ increases with the distance from the viewpoint to the sample point to account for the decreasing precision of the depth buffer with increased depth. The complete algorithm for redundant sample removal iterates over every depth sample s in every scan S of the environment and tests it in turn against all of S's ancestor scans.

4.2 Mesh Creation

We construct polygonal meshes for those portions of a sample that remain after redundant points have been removed. We begin by creating a dense, regular mesh over the surviving points in each sample. Although this mesh has many more polygons than we want to render, its regular structure makes it well suited to geometric simplification. We apply view-dependent simplification to reduce the storage and rendering cost for our impostors by taking advantage of the limited viewpoints and viewing directions for a set of TDMs.

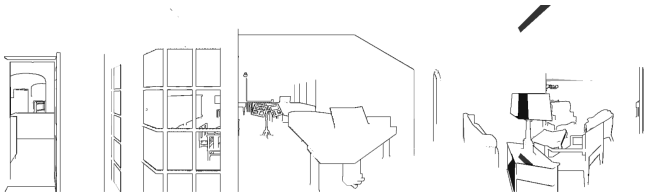
4.2.1 Generating a Dense Regular Mesh

In order to avoid introducing cracks or skins in the reconstruction as can happen with some point-based impostor schemes, we create the mesh for each face of each scan cube as a continuous surface wherever we can be reasonably sure of surface connectivity. This connectivity information is exactly the same as the per-pixel adjacency computed during Voronoi-based sampling and uses the tests described in Section 3.

In order to cover seamlessly the entire space surrounding the viewpoint, we cover the entire solid angle subtended by each surviving pixel in a sample with two triangles. Since OpenGL samples the scene at pixel centers [Woo et al. 1997], we compute depth values at the corners of each pixel by interpolating among the four pixels surrounding each corner. Once again, we use per-pixel adjacency to identify depth discontinuities. We include a pixel's depth value in the interpolation only if there is no discontinuity between it and the pixel being triangulated.

We construct triangles for every non-redundant pixel of every face of every sample. The result is a set of six regular meshes for each scan cube. The next stage of ITDM construction is geometric simplification to reduce the number of polygons in these meshes.

(a) Four sides of a panoramic view from the house environment.



(b) Per-pixel adjacency. Darkened pixels are endpoints of skin edges.

(c) Visibility error. Lighter pixels are observed by fewer samples than darker ones. Red pixels belong to the global void surface.

Figure 7: *Skin detection and visualization of the void surface in the house environment*

4.2.2 Simplifying Dense Meshes

The viewpoints for a group of samples are all defined to fall within some navigable region. The incremental TDMs created from these samples will be used only for viewpoints within that region. As a result, we compute a hierarchical representation for view-dependent simplification as part of ITDM generation.

We must take particular care to avoid errors that cause screen-space deviation in silhouette and boundary edges in the dense meshes. Since textures will be applied to ITDMs using projective texturing instead of per-vertex texture coordinates, the texture will not deform with the geometry. Instead, screen-space errors in silhouettes and mesh boundaries will cause the projected texture to appear partly on surfaces neighboring the locus of error. Moreover, by preserving screen-space boundaries we can assemble the six faces of a scan cube seamlessly into a panorama surrounding the viewer.

The simplification process consists of two phases. First, we generate a view-independent sequence of edge collapses by simplifying the original mesh using a quadric error metric formulation [Garland and Heckbert 1997] until it has only a few hundred triangles. These edge collapses form a merge tree similar to those used in view-dependent simplification algorithms such as [Hoppe 1997; Luebke and Erikson 1997; Xia et al. 1997].

The second phase consists of a static, view-dependent simplification. The merge tree constructed in the first phase of simplification defines a hierarchical grouping of the vertices in the input mesh. This grouping is used as an input to a hierarchical view-dependent simplification scheme [Luebke and Erikson 1997]. This view-dependent phase preserves mesh boundaries and silhouette edges to within half a pixel of error. We identify a set of silhouette edges by taking several viewpoints at the extremes of the navigable region and constructing the union of the set of silhouettes from each viewpoint. No further modifications are made to the meshes after this phase terminates.

Model	# of regions	# of samples per region (avg.)	Time per sample (avg.) (min)	Sample size (avg.) (KB)	Time per scan cube (avg.) (min)	Avg. % of points removed	Avg. scan cube size (KB)	Avg. scan cube polygon count
House	11	6.3	4.75	1,284	28.9	91.4%	114	5,039
Powerplant	31	13	5.9	1,879	10.2	89.0%	3,340	169,342

Table 1: *Preprocessing statistics for the house and power plant environments. The higher polygon count and storage requirements for the power plant are a result of its high complexity. Scan cubes in the house took a long time to create because some samples contained objects spanning hundreds of thousands of pixels that were simplified as one unit. Removing redundant points reduced storage requirements by a factor of two and processing times by a factor of six.*

4.3 Rendering Distant Primitives Using ITDMs

The ITDM construction process yields the following information:

- A set of simplified polygonal meshes (one mesh for each of the six faces of a scan cube corresponding to one of the original samples of the environment)
- The viewpoints of the cameras used to acquire the original samples
- The spanning tree D that was used for redundant sample removal
- Color information from the original samples.

We use these components to identify a set of scan cubes to construct a view of the environment for rendering with few or no cracks. This identification can incorporate a triangle budget to maintain some minimum frame rate or a scan budget to maximize fidelity at the risk of lower frame rates.

4.3.1 Rendering with a Fidelity Target

In order to maximize fidelity at the (possible) expense of lower frame rates, the user can specify a budget of up to k scan cubes when rendering ITDMs. We select a set of scan cubes $D_1 \ldots D_k$ to meet this budget as follows.

The first scan cube selected, denoted D_1, is the one whose sample location is nearest the user's viewpoint. This will probably be a delta cube. Since a delta cube contains only those portions of surfaces not visible in any of its ancestors, we also select the ancestors $D_2, D_3, \ldots, D_j$ of D_1 in the dependency tree, up to and including the intra scan (D_j). Scans are added to the renderable set beginning with the intra scan D_j and proceeding down the dependency tree toward D_1. This reversal ensures that a delta cube will be added to the renderable set only after all of its parents are already present. If the set of scan cubes to be added is larger than the scan budget, we truncate the set.

Once D_1 has been added, more scan cubes can be selected for rendering if the scan budget has not been exhausted. We select the scan D_{j+1} nearest to the user that has not yet been selected for rendering, find the path from D_{j+1} to the root of the dependency tree, then add it and its ancestors to the renderable set in reverse order as above. This process of selection and addition continues until the scan budget is exhausted.

4.3.2 Rendering with a Triangle Budget

The user may wish to guarantee some minimum frame rate instead of image fidelity. The algorithm for selecting scan cubes for rendering is exactly the same as above except for the termination conditions. Rather than track the number of scan cubes selected so far, we track the number of primitives in selected scan cubes that fall within the view frustum.

After computing a set of scan cubes, we draw the meshes that fall within the view frustum. We use projective texture mapping to place the color information from the original samples back onto the simplified geometry.

5 Implementation and Results

We have tested Voronoi-based sampling and ITDMs in two complex environments. The first is a one-story model of a house containing 262,879 triangles with per-vertex colors, surface normals, and texture coordinates. Moreover, the house model incorporates 19 megabytes of high-resolution textures to provide surface detail. Although the house model is small enough to be rendered at close-to-interactive rates on current graphics hardware without image-based simplification, its visibility patterns are complex enough to provide good tests for our methods. Our second environment is a model of an 80-meter-tall coal-fired power plant containing 12.5 million triangles. Surfaces in the power plant are augmented with per-vertex color and normals.

5.1 Navigable Region Generation

We tested our algorithm in several navigable regions in both environments. Regions in the power plant were automatically created to be as large as possible while enclosing no more than a user-specified number of polygons. One set of samples and one set of impostors are created for each navigable region in order to replace primitives outside the region. Regions in the power plant vary from 2 to 30 meters on a side to accommodate an irregular distribution of primitives. Regions in the house are chosen to be uniformly 2 meters on a side due to the model's smaller size. In our current implementation, we do not simplify the primitives inside the navigable region. At runtime, they are rendered using the original geometry.

5.2 Voronoi-Based Sampling

We selected 31 regions in the power plant and 11 in the house to test the performance of Voronoi-based sampling algorithm. These regions include a mix of easy and difficult cases as discussed in Section 6. In each region, we computed an image-based simplification of all primitives outside that region. Table 1 gives aggregate preprocessing statistics for each model.

5.2.1 Per-pixel Adjacency and Visibility Error Computation

We chose a difficult region in the power plant for a closer evaluation of incremental Voronoi-based sampling. The difficulty in this region arises from many complex, interacting occluders near the navigable region on all sides. This region is shown in the video in the comparison between ITDMs and static LODs. Sampling began with a set of 4 samples (512x512 pixels on each face) from the corners of the navigable region and continued until the sampling algorithm run had acquired 20 samples total. Fig. 8 shows the decrease of the detected visibility error as more samples were added.

5.3 Incremental Textured Depth Meshes

We generated a set of ITDMs for each of the regions in which we ran the Voronoi-based sampling algorithm. In this section we give preprocessing statistics for the creation of ITDMs and compare their rendering speed and fidelity with both static LODs and standard TDMs.

5.3.1 Preprocessing

We generated ITDMs for 31 cells in the power plant environment using the sample groups created by our approach. Table 1 gives statistics on preprocessing time and storage requirements. Overall, the power plant environment was far more complex and time-consuming to handle than the house. The higher scan cube creation time for the house is an implementation detail: when simplifying meshes, we handle one object at a time, which is inefficient when one or two objects cover the entire screen (as is often the case in the house).

The process of removing redundant samples resulted in a 6-times improvement in the time required for geometric simplification and a 2-times reduction in the size of the resulting meshes. We measured this by running ITDM creation twice on the same

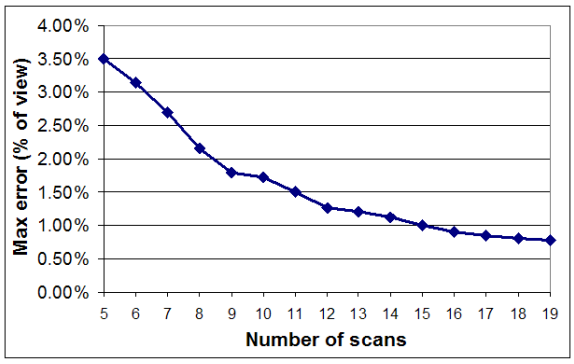


Figure 8: *Decrease in maximum detected visibility error as more samples are generated to simplify a region of the power plant containing 12.5M triangles and 1,185 objects.*

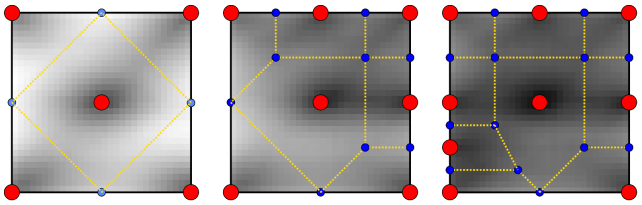


Figure 9: *Visibility error for a region in the power plant as more samples are acquired. Darker colors represent lower errors. Red points are sample locations, blue points are candidate viewpoints, and yellow lines show the Voronoi diagram of existing sample locations. From left to right, we show the error function after acquiring 5, 7, and 9 samples.*

data sets: once with redundant points removed, once with all points kept (equivalent to creating all scan cubes as intra scans). All other settings were left unchanged. The comparatively small decrease in storage space is due to the higher number of boundary edges (which are preserved exactly) in meshes with redundant points removed.

5.4 Comparison with previous approaches

In this section, we compare our algorithm with two earlier approaches for rendering large, complex models. The first one is based on standard TDMs [Aliaga et al. 1999; Darsa et al. 1998; Sillion et al. 1997; Wilson et al. 2001], where the scene was subdivided into uniform-sized cells and a single panoramic TDM was computed for each cell. These cells were kept small in order to minimize the severity of skins. The second approach uses LODs of all the objects in the scene and switches between different LODs based on the viewpoint.

5.4.1 Rendering Speed

The increased fidelity of ITDMs comes at the cost of higher polygon counts in order to accommodate the geometric portion of multiple ITDMs. As a result, the frame rate of an interactive system is lower with ITDMs than with standard TDMs. Standard TDMs are typically created from uniformly spaced samples, only render one scan cube at a time, and are simplified far more aggressively than ITDMs. The frame rates achieved by ITDMs are illustrated in Fig. 10. The frame rates we observe with ITDMs are well within interactive ranges ($20 - 40$ frames per second) for most viewpoints. These frame rates are consistently faster than those achieved by static LODs, which fall to 2 or 3 frames per second when the user looks toward the (very complex) center of the power plant.

5.4.2 Image Quality

The overall image quality of ITDMs is almost comparable to that of static LODs, though they show small visual artifacts like small cracks from some viewpoints. ITDMs also do not have the popping artifacts that arise when switching between different static LODs.

ITDMs provide a higher-quality reconstruction of the far field with fewer artifacts than standard TDMs, as shown in Fig. 1. This is due largely to the absence of skins in ITDMs. Since depth discontinuities in a single ITDM are exposed (rather than covered by skins), we can fill in the holes and thus reduce or eliminate visibility artifacts by rendering meshes constructed from several different sample locations. Examples of increasing ITDM fidelity by adding samples to the reconstruction are shown in Figure 11. By contrast, rendering more than one standard textured depth mesh in the same view region makes matters worse: not only do the skins from the first TDM obscure any newly visible surfaces contained in subsequent meshes, but the skins in subsequent meshes will probably obscure surfaces that are already rendered properly in the reconstruction! Moreover, navigable regions in systems using standard TDMs are often kept small to minimize the severity of artifacts. Since we have reduced these artifacts, we can create larger cells while maintaining a high-fidelity reconstruction of the far field. This can lead to lower preprocessing and storage costs due to fewer sets of impostors that must be generated.

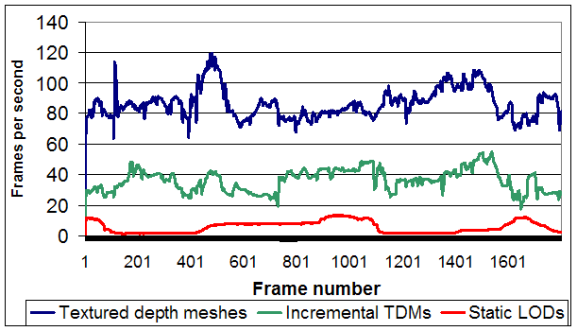


Figure 10: *Comparison of frame rates for standard TDMs, ITDMs, and static LODs in the power plant. ITDMs maintain an interactive frame rate while providing a reconstruction with fewer artifacts than standard TDMs.*

Lack of Popping: We have also reduced the artifacts that appear when the user moves between different regions. In some systems [Aliaga et al. 1999], this transition was accompanied by a distracting "pop" as the artifacts from one TDM were replaced with the (drastically different) artifacts from another. Cell-to-cell transitions using ITDMs are far smoother due to the reduced incidence of skins and cracks.

5.4.3 Storage Overhead

The storage costs of ITDMs are greater than those of static LODs and lower than those of standard TDMs for a particular image quality. Static LODs roughly double the size of the input database, turning the 660MB power plant into 1.3GB of geometry. ITDMs for the 31 navigable regions occupy a total of 1.4GB. TDMs acquired at 1.5-meter intervals (to increase image quality) throughout these regions occupy a total of 9.4GB, almost 7 times larger than ITDMs.

These figures depend on the particular error measures used. Static LODs typically generate several versions of an object, each with half the polygons of its predecessor: this leads to the doubling of database size. Lower error thresholds for TDMs and ITDMs decrease the spacing between adjacent samples. In such a scenario the storage overhead for ITDMs is considerably lower as compared to that of TDMs. This is mainly due to our improved sampling algorithm, which generates fewer viewpoints as well as the removal of redundant points during construction of delta cubes.

6 Analysis and Limitations

In this section we analyze the performance of our algorithms, highlight cases where they are expected to work well, and describe many of their limitations.

6.1 Complexity

We discuss the complexity of Voronoi-based sampling and ITDM construction in terms of the number of samples, n. We assume that the size of a sample is constant. The overall behavior of each algorithm depends on the complexity of its various steps, which are described below.

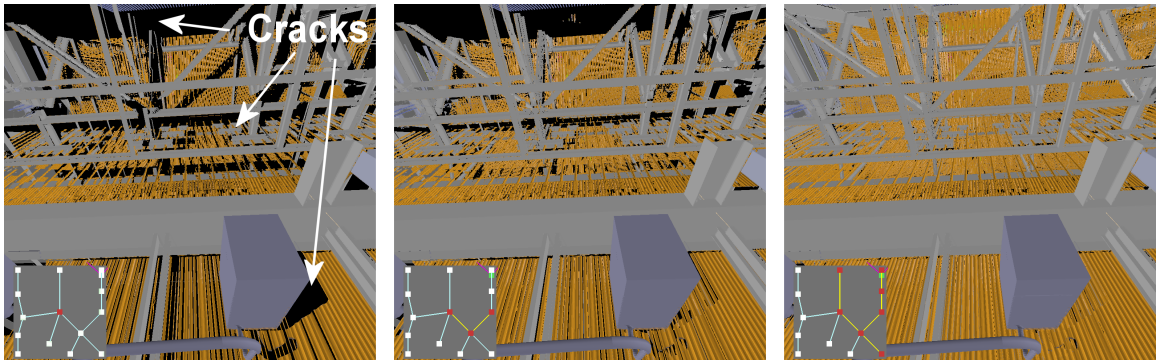


Figure 11: Fidelity improvement of ITDMs with more scans. From left to right, we show a far field in the power plant rendered with the intra cube plus zero, two, and five delta cubes. The ITDM dependency tree is shown at bottom left. Red nodes show which cubes are rendered.

6.1.1 Sampling algorithm

the complexity of the various steps performed as each sample is added. Identifying skins and building the void surface within a sample can take up to $O(n)$ time in the worst case when using the multiple-source-image test. Acquiring a sample and adding it to the reconstruction are independent of the number of samples. The computation of 2D Voronoi diagram can take $O(n \log n)$ time during each iteration. The overall time for sample computation is $O(n^2 \log n)$. This can be easily improved by using dynamic point insertion Voronoi computation algorithms.

6.1.2 ITDM construction algorithm

The two most asymptotically expensive parts of ITDM construction are building the dependency tree and identifying redundant samples. We build the dependency tree on a complete graph over n samples having $O(n^2)$ edges. The minimum spanning tree for such a graph $G = (V, E)$ can be computed in $O(E + V \log V)$ time for a total cost of $O(n^2)$ time in the number of samples. In the worst case, the spanning tree has depth n. In this case, redundant sample identification in a sample may have to examine all $O(n)$ ancestor samples. Performing this step on n samples gives a worst-case complexity of $O(n^2)$. In practice, the redundancy tests usually exit after only 1 or 2 comparisons. The mesh simplification and merge tree computation takes about $O(m \log m)$ steps, where m is the number of primitives in the dense mesh.

6.2 Voronoi-Based Sampling

The Voronoi-based sampling generates image-based simplifications of distant primitives for a navigable region. Its performance is governed by the underlying objective function and incremental sample computation.

The performance of our algorithm on two complex environments suggests that the solid angle subtended by the void surface can be a reasonable, conservative measure of visibility error. This measure does not require comparing two images to detect differences, and thus does not suffer from the limitations of numeric techniques used to comparing the visual similarity of two different images. Moreover, the void surface can be constructed using only information present in a set of samples: it does not require access to the original primitives in the environment.

The angle subtended by the void surface can over-estimate the severity of artifacts in two ways. First, as an aggregate measure it has no notion of the individual errors in a view. One thousand single-pixel errors scattered through an image are counted just as severely as a large 1000-pixel block. Second, there is no notion of minimum perceptible error. Artifacts that interpolate between two surfaces of identical color and depth will be counted as severely as errors that involve radical changes in color or occlusion. In spite of these shortcomings, we find that in practice the visibility of the void surface guides us toward regions of high error.

6.2.1 Good Scenarios for Incremental Sampling

In general, environments where the potentially visible set changes slowly and smoothly when traversing the navigable region will be well captured by Voronoi-based sampling. Such situations occur when most objects in the environment are far from the navigable region or when there are a few visibility events between such objects over the region. Environments with complex geometry in the distance are often handled well: the farther away an object is from the camera, the less horizontal parallax it will display as the camera moves, which in turn reduces the potential visibility of any void volume behind it.

6.2.2 Bad Scenarios for Incremental Sampling

The presence of complex, interacting occluders close to the view volume can result in rapid and abrupt changes in the potentially visible set. Situations such as a view through a closely spaced set of parallel pipes can be difficult because of the lack of overlap in the the space seen through the pipes even for nearby viewpoints. Environments with surfaces parallel to the camera view axis can also pose problems: in some cases, parts of these surfaces will be erroneously classified as skins. Such configurations are usually easily resolved by the addition of a single sample that sees such surfaces at a different angle of incidence.

6.3 Incremental Textured Depth Meshes

We build a set of incremental textured depth meshes from the panoramic samples. We use the adjacency information computed during sampling to avoid introducing skins into ITDMs and minimize cracks by using additional ITDMs.

6.3.1 Benefits of Incremental TDMs

The identification and removal of redundant data considerably reduces the costs of preprocessing, storing, and rendering multiple TDMs. Since a set of ITDMs is created from multiple samples, it provides a more faithful reconstruction of distant primitives than standard TDMs that can only render one mesh built from one sample at any given time. As a result, the reconstruction of the far field provided by ITDMs remains usable over a much larger region than with traditional, single-source TDMs.

6.3.2 Difficult Situations for Incremental TDMs

Incremental textured depth meshes perform best in situations with fewer visibility events and high spatial coherence. They can perform poorly in the following two situations:

1. Rapidly Changing Visibility Between Viewpoints: Redundant sample removal depends on the assumption that visibility changes smoothly. This will perform poorly in situations where this assumption is incorrect. One example of such a situation is when large occluders are interposed between the viewpoint of a child scan and its ancestor scan. Fortunately, such situations are rare due to the requirement that the the region containing the viewpoints for the original panoramic samples of the environment be free from geometry. If such geometry exists, it is removed during impostor generation and handled separately at runtime.

2. Complex Silhouettes in the Depth Buffer: In order to make the borders of one ITDM line up neatly with another, we preserve mesh boundaries and silhouette edges with sub-pixel accuracy. This decision can cause problems in configurations with complex silhouettes by creating meshes with many more polygons than we would like to render and is the chief cause of high polygon counts in ITDMs. Since we only require that the borders of different meshes line up exactly, we could conceivably modify complex boundaries to be straight lines instead of the stair-step shapes created by removing redundant pixels. We could also fuse boundaries across meshes using a method similar to the one described for range images in [Curless and Levoy 1996].

6.3.3 Errors Introduced by Use of Delta Cubes

The process of redundant sample removal is a lossy simplification scheme that introduces two kinds of error. First, discarded points sometimes contain information about surface texture that is not present elsewhere. By discarding this information, we limit texture quality in a sample to whatever is present in that sample alone. In practice, the user will rarely approach ITDMs closely enough for this to be a problem. Second, removing redundant samples can cause seams to become visible where surfaces in one scan cube meet surfaces in another. If no samples were removed (i.e. all scan cubes are intra scans), these seams could be filled in when a single surface is rendered in two or more scan cubes. This solution would bring with it higher preprocessing, storage, and rendering costs for those polygons belonging to duplicated surfaces.

7 Conclusions and Future Work

We have presented a new approach for computing image-based simplifications of a large environment. Our approach includes a Voronoi-based sampling algorithm as well as an incremental TDM representation. It has been applied to two complex environments. Our preliminary results are encouraging: we are able to render the complex 13 million powerplant model at more than 20 frames a second on a single PC with very few visual artifacts (as shown in the video). The large arrays of pipes in this model are a challenge for any rendering acceleration algorithm and highlight the potential of our approach.

There are many directions for future work. The Voronoi-based sampling algorithm is currently limited to a two-dimensional navigable region. The extension to three dimensions is relatively straightforward: since the objective function is defined at all points, we would need to compute candidate points using the 3D Voronoi diagram of existing sample locations instead of the 2D version. Moreover, we would need to acquire the initial samples from within a 3D space of viewpoints. We would also like to incorporate a quality measure into the objective function that accounts for parameters such as sampling density, standoff distance, and angle of incidence.

At present, incremental TDMs are limited to showing surface texture and diffuse lighting present in the original sample. Real-time programmable shading offers far more flexibility. For example, if we were to capture color and normal information in the sample instead of simply color, we could change the illumination in ITDMs dynamically, including specular highlights. We would also like to optimize ITDM construction so that it can be performed dynamically instead of as a preprocess. Furthermore, we want to investigate automatic region decomposition techniques for large environments. Finally, we would like to apply these ideas to represent and render range images of real world scenes.

References

ALIAGA, D., AND LASTRA, A. 1999. Automatic image placement to provide a guaranteed frame rate. In *Proc. of ACM SIGGRAPH*.

ALIAGA, D. ET AL 1999. Mmr: An integrated massive model rendering system using geometric and image-based acceleration. In *Proc. of ACM Symposium on Interactive 3D Graphics*.

ALIAGA, D., FUNKHOUSER, T., YANOVSKY, D., AND CARLBOM, I. 2002. Sea of images. In *Proc. of IEEE Visualization*.

ALIAGA, D. G. 1996. Visualization of complex models using dynamic texture-based simplification. In *Proc. of Visualization'96*, 101–106.

BANTA, J., ZHIEN, Y., WANG, X., ZHANG, G., SMITH, M., AND ABIDI, M. 1995. A best-next-view algorithm for three-dimensional scene reconstruction using range images. *Proc. of SPIE 2588*, 418–429.

BAXTER, B., SUD, A., GOVINDRAJU, N., AND MANOCHA, D. 2002. Gigawalk: Interactive walkthrough of complex 3d environments. *Proc. of Eurographics Workshop on Rendering*.

CHANG, C., LI, Z., VARSHNEY, A., AND GE, Q. 2001. Hierarchical image-based and polygon-based rendering for large-scale visualization. In *Scientific Visualization*, Springer-Verlag.

COHEN-OR, D., CHRYSANTHOU, Y., DURAND, F., GREENE, N., KOLTUN, V., AND SILVA, C. 2001. Visibility, problems, techniques and applications. *SIGGRAPH Course Notes # 30*.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *SIGGRAPH 96 Conference Proceedings*, 303–312. 1996.

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

DEBEVEC, P., YU, Y., AND BORSHUKOV, G. 1998. Efficient view-dependent image-based rendering with projective textures. *Proc. of Eurographics Workshop on Rendering*, 105–116.

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

EL-SANA, J., SOKOLOVSKY, N., AND SILVA, C. 2001. Integrating occlusion culling with view-dependent rendering. *Proc. of IEEE Visualization*.

FLEISHMAN, S., COHEN-OR, D., AND LISCHINSKI, D. 2000. Automatic Camera Placement for Image-Based Modeling. In *Computer Graphics Forum* 19(2), pp. 100-110, June 2000.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error bounds. *Proc. of ACM SIGGRAPH*, 209–216.

GONZALEZ-BANOS, H., AND LATOMBE, J. 1998. Planning robot motions for range-image acquisition and automatic 3d model construction. In *Proc. AAAI Fall Symp.*, AAAI Press.

GONZALEZ-BANOS, H., AND LATOMBE, J. 2001. A randomized art-gallery algorithm for sensor placement. In *Proc. ACM Symp. on Computational Geometry*.

GREENE, N., KASS, M., AND MILLER, G. 1993. Hierarchical z-buffer visibility. In *Proc. of ACM SIGGRAPH*, 231–238.

HOPPE, H. 1997. View dependent refinement of progressive meshes. In *ACM SIGGRAPH Conference Proceedings*, 189–198.

JESCHKE, S., AND WIMMER, M. 2002. Textured depth mesh for real-time rendering of arbitrary scenes. In *Proc. Eurographics Workshop on Rendering*.

JESCHKE, S., WIMMER, M., AND SCHUMAN, H. 2002. Layered environment-map impostors for arbitrary scenes. In *Proc. Graphics Interface*, 1–8.

LEVOY, M. 1995. Polygon-assisted JPEG and MPEG compression of synthetic images. In *SIGGRAPH 95 Conference Proceedings*, 21–28.

LINDSTROM, P., AND TURK, G. 2000. Image-driven simplification. *ACM Trans. on Graphics 19*, 3, 204–241.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygon environments. In *Proc. of ACM SIGGRAPH*.

LUEBKE, D., REDDY, M., COHEN, J., VARSHNEY, A., WATSON, B., AND HUEBNER, R. 2002. *Level of Detail for 3D Graphics*. Morgan-Kaufmann.

MACIEL, P., AND SHIRLEY, P. 1995. Visual navigation of large environments using textured clusters. In *ACM Symposium on Interactive 3D Graphics*, 95–102.

MAVER, J., AND BAJCSY, R. 1993. Occlusions as a guide for planning the next view. *IEEE PAMI 15*, 5, 417–433.

MAX, N., AND OHSAKI, K. 1995. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*.

MCMILLAN, L., AND BISHOP, G. 1995. Plenoptic modeling: An image-based rendering system. In *Proc. of ACM SIGGRAPH*, 39–46.

NYLAND, L., LASTRA, A., MCALLISTER, D., POPESCU, V., AND MCCUE, C. 2001. Capturing, processing and rendering real-world scences. In *Videometrics and Optical Methods for 3D Shape Measurement, Electronic Imaging*, vol. SPIE 4309.

O'ROURKE, J. 1997. Visibility. In *Handbook of Discrete and Computational Geometry*, CRC Press LLC, J. E. Goodman and J. O'Rourke, Eds., 467–480.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. *Proc. of ACM SIGGRAPH*.

PITO, R. 1999. A solution to the next best view problem for automated surface acquisition. *IEEE Trans. Pattern Analysis and Machine Intelligence 21*, 1016–1030.

REED, M., AND ALLEN, P. K. 1999. Constraint-based sensor planning for scene modeling. *Computational Intelligence in Robotics and Automation*.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. *Proc. of ACM SIGGRAPH*.

SCHAUFLER, G., AND STURZLINGER, W. 1996. A three dimensional image cache for virtual reality. *Computer Graphics Forum 15*, 3, C227–C235.

SHADE, J., LISCHINSKI, D., SALESIN, D., DEROSE, T., AND SNYDER, J. 1996. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proc. of ACM SIGGRAPH*, 75–82.

SHADE, J., GORTLER, S., WEI HE, L., AND SZELISKI, R. 1998. Layered depth images. *Proc. of ACM SIGGRAPH*, 231–242.

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

STURZLINGER, W. 1999. Imaging all Visible Surfaces. In *Graphics Interface Proceedings 1999*, pp. 115-122.

WILSON, A., MAYER-PATEL, K., AND MANOCHA, D. 2001. Spatially-encoded far-field representations for interactive walkthroughs. *Proc. of ACM Multimedia*.

WOO, M., NEIDER, J., AND DAVIS, T. 1997. *OpenGL Programming Guide, Second Edition*. Addison Wesley.

XIA, J., EL-SANA, J., AND VARSHNEY, A. 1997. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (June), 171–183.