Eurographics Symposium on Rendering (2004)
H. W. Jensen, A. Keller (Editors)

Hardware Accelerated Visibility Preprocessing using Adaptive Sampling

S. Nirenstein[†] and E. Blake

University of Cape Town, South Africa

Abstract

We present a novel aggressive *visibility preprocessing technique for general 3D scenes. Our technique exploits commodity graphics hardware and is faster than most conservative solutions, while simultaneously not overestimating the set of visible polygons. The cost of this benefit is that of potential image error.*
In order to reduce image error, we have developed an effective error minimization heuristic. We present results showing the application of our technique to highly complex scenes, consisting of many small polygons. We give performance results, an in depth error analysis using various metrics, and an empirical analysis showing a high degree of scalability. We show that our technique can rapidly compute from-region visibility (1hr 19min for a 5 million polygon forest), with minimal error (0.3% of image). On average 91.3% of the scene is culled.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three Dimensional Computer Graphics and Realism]: Visible line/surface removal

1. Introduction

In this paper we present our *aggressive* visibility preprocessing technique. The term *aggressive visibility* was introduced by Nirenstein et al.[NBG02] as an augmentation of the taxonomy of Cohen-Or et al.[COCSD03] as an additional distinction to their terms *conservative*, *approximate*, and *exact*.

Conservative techniques consistently overestimate and incur what we term false-visibility errors. These occur when primitives that are not visible are considered visible. Approximate visibility techniques incur both false-visibility (as above) as well as false-invisibility errors: where visible primitives are excluded erroneously. Exact visibility solutions avoid both types of error and provide both accurate images and the tightest possible visible sets.

In contrast, aggressive methods generate a subset of the exact visible geometry and so exhibit false-invisibility. As noted by Nirenstein et al.[NBG02], aggressive visibility causes image error, but can be useful in practice if: (a) the visual impact of the error is acceptably small for the given application, (b) the algorithm has benefit in computational efficiency or (c) it handles scenes that cannot be treated effectively with conservative alternatives, due to excessive overestimation.

Our technique uses a sampling approach and gains speed by exploiting graphics hardware. We show that it is able to handle very complex scenes with many small occluders far better than existing from-region techniques.

Aggressive algorithms are suited to the extremely complex scenes that arise in actual use of visibility preprocessing where the amount of time spent on preprocessing becomes a significant issue. Exact algorithms[NBG02, Bit02] cannot compete in such cases. We consider those situations where the amount of very fine grained geometry is so large that conservative algorithms tend to overestimate visibility sets excessively. We show this with a forest scene that cannot effectively be handled, to our knowledge, by any visibility preprocessing technique.

The issue that then arises is how to minimise the visual impact of the visibility errors inherent in aggressive techniques. We quantify this error via a number of error metrics that we develop. We also present error minimization heuristics that exploit adaptive sampling.

Specifically, our contributions are:

- An aggressive hardware accelerated technique that rapidly

[†] {shaun,edwin}@cs.uct.ac.za

determines visibility from a surface or region. This technique fits a niche, treating scenes for which conservative techniques overestimate excessively, or for which exact algorithms are too slow.
- A heuristic that guides adaptive subdivision to prevent excessive sampling and reduce under sampling.
- A divide and conquer frame work for sample-based visibility. Our strategy provides effective cache management for samples, and provides efficient visibility culling to sample points.
- An effective general purpose visibility preprocessor that has a relatively simple implementation.

1.1. Overview

We begin with a brief discussion of previous work (Section 2). Next, (Section 3) in a bottom up fashion, we show how the visibility from a surface may be computed efficiently, using standard graphics hardware. We then address the issues of sampling and error measures that arise (Section 3.2). With these preliminaries in place we can present the complete algorithm (Section 4) and an analysis of its complexity (Section 5). Finally we show how the algorithm performs.

2. Previous work

Visibility culling algorithms are often categorised as either those that compute visibility from a *point* or those that compute visibility from a *region*. Since our focus is from-region techniques, for the sake of brevity we refer the interested reader to the comprehensive survey of Cohen-Or et al.[COCSD03].

The vast majority of visibility algorithms are conservative in nature. Exact algorithms for general scenes do exist[NBG02, Bit02, Nir03], however, they are ill suited to rapid preprocessing. The accuracy of conservative techniques is determined by their ability to perform occluder fusion. Older techniques[COFHZ98, SVNB99] are no longer effective. Earlier occluder fusion techniques[SDDS00, DDTP00] often greatly overestimate visibility.

In order to achieve effective culling various assumptions have to be made about the scene structure. For indoor scenes it is possible to partition the scene into cells separated by portals. In this case, the visibility of a cell is reduced to determining the existence of a stabbing line through a sequence of portals between two cells[ARJ90, TS91]. Other assumptions include those of a 2D or $2\frac{1}{2}$D scenes[KCCO01, WWS00, BWW01] and even $2\frac{1}{2}D+\varepsilon$ scenes[LSCO03]. None of these techniques can be applied to large, general 3D scenes.

Our technique is similar to the ray casting approach of Gotsman et al. [GSF99]. They use a sequential analysis to control the termination of a random ray sampling process. In contrast, we use a hardware approach that allows ray samples to be taken several orders of magnitude faster, however, our ray distribution is not uniform: relatively few ray origins are used. We compensate by selecting these intelligently using heuristic guided adaptive sampling. We cannot compare error results since Gotsman et al. did not report any. Sayer et al.[SLCO*04] also use aggressive visibility. Imposters are used to mask false-invisibility error in their aggressive visibility approach.

Van de Panne and Stewart[vdPS99], Schaufler et al.[SDDS00] and Leyvand et al.[LSCO03] have implemented naïve viewpoint sampling techniques as reference solutions. Wilson and Manocha[WM03] use view point sampling to build incremental textured depth meshes.

3. Visibility From a Surface

In this section our novel sampling approach to “from-region” visibility is presented. We describe a sampling method that is based on the hemi-cube[CG85, HA00] and exploits the performance of common graphics rendering hardware.

3.1. The Visibility Cube

A *visibility sample* is defined to be the set of polygons visible from a given point. A visibility cube (closely related to a radiosity hemi-cube) is used to generate such samples (Figure 1). This is created by treating each of the six sides of a tiny cube enclosing the sample point as independent depth and frame buffers onto which the scene is rendered. Each polygon is assigned a distinct 32 bit colour. The set of polygons mapped by at least one pixel in any of the six frame buffers is considered to be the set of polygons visible from the sample point. The visibility cube can be considered a high density sampling over the angular domain, for a fixed spatial position. The intended application for visibility

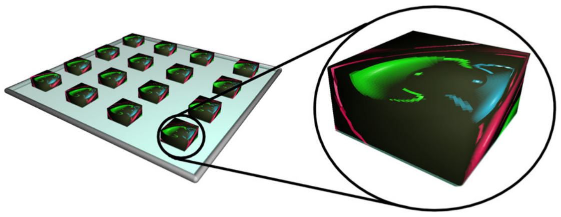
Figure 1: The Visibility Cube. *A sample of several visibility cubes over a surface. The visible geometry (of several teapots) has been projected onto the cubes.*

culling is invariably a rasterization engine. A useful heuristic for obtaining good accuracy for visibility samples in practice is to set parameters (frame buffer resolution, bit depth of depth buffer and near and far planes) similar to that of the desired output parameters. For full accuracy, these factors should be set at the Nyquist limit.

Sub-sampling the intended rendering resolution is beneficial, however, since it enhances performance by minimizing frame buffer reads and reducing the required fill rate.

This allows accuracy to be traded for speed. Although accuracy is reduced, sub-sampling results only in the occasional omission of polygons which only contribute to few visible pixels, and therefore have little visual impact. This is known as *approximate* culling or equivalently *contribution* culling[ASVNB00, BMH98, Zha98].

We only consider visibility at the triangle level, since our algorithm is an aggressive from-region approach. Object level visibility could be achieved easily by assigning a colour per object, however, the algorithm then becomes approximate, and the degree of conservativity would depend on object granularity. This may be preferable though, if the scene consisted of many sub-pixel sized triangles, and was intended to be rendered at high resolutions with multi-sample anti-aliasing.

3.2. Sampling for From-region Visibility

The "from-region" visibility set can be defined in terms of visibility samples. This is simply the union of the visible sets of all possible visibility samples taken within the rectangular region. In theory, it would be possible to find a finite set of samples from which exact visibility could be determined by partitioning the region into areas of constant visibility and generating one sample per area. This would be ideal, however, the high combinatorial complexity of evaluating such a partition is prohibitive[Pla92]. Instead, we use heuristic guided adaptive sampling that examines the structure of the visibility samples in order to generate a sample pattern that approximates the theoretically ideal subdivision. We present our heuristic based approach in this section.

Uniform sampling is a naïve solution to our sampling problem (see Figure 2a). With this approach, under-sampling may manifest as unacceptable false-invisibility errors, while over-sampling may lead to prohibitive execution costs.

Our adaptive technique proceeds as follows: we assume a rectangular sample domain embedded in 3-space. To begin with, visibility samples are evaluated at the corners of the rectangle. Then a decision is made whether or not to refine the rectangle into four subregions based on our error minimization heuristic. The user specifies a threshold (see below). The subdivision proceeds recursively, in a manner equivalent to the depth-first generation of a quad-tree. A rectangular subregion, with visibility samples at its four corners, is treated as a node in the quad-tree. Corners are shared between parents and children and among siblings in the quad-tree. It is important to cache shared visibility samples in order to prevent redundant computation. A typical adaptive subdivision is illustrated in Figure 2b.

The subdivision heuristics we use are based on visibility sample similarity. This is derived from the simple observation, that if two viewpoints see similar item-buffer images, then any viewpoint between the two, will also most likely see a similar image. We begin with a metric based on the

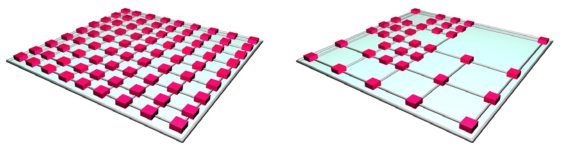

Figure 2: Uniform vs. Adaptive Sampling. *(a) A uniform distribution of visibility cubes on a 2D surface. (b) A non-uniform distribution of visibility cubes generated by an adaptive subdivision. The adaptive sub-division attempts to minimise both error, and the number of samples required. A quad-tree structure is effectively built on the surface.*

sampled visible set only. We then consider a strict metric that directly compares each pixel of the panoramic visibility sample. Finally, we relax the per pixel equivalence constraint by subdividing the image into subsections, and applying a visible set similarity criterion to each subsection. We also consider the more obvious visual benefits of an image based heuristic, although a full analysis is beyond the scope of this paper.

3.3. Basic Error Metric

Adaptive subdivision requires a decision at each quad-tree node (rectangular subregion) whether or not to continue subdividing. This decision is based on a heuristic that employs a sample-error metric to establish, given four corner visibility samples, if any interior view points are likely to contain additional polygons.

Ideally, areas with high frequency changes in visibility should be sampled more densely. Our first attempt is to explicitly encode the normalised difference between visibility samples. Given the visible sets from four visibility samples, s_0, s_1, s_2, s_3, we define:

$$Err(s_{0..3}) = 1 - min_{i=0}^{3}\left(\frac{|\bigcap_{j=0}^{3} s_j|}{|s_i|}\right) \tag{1}$$

Err returns 1 *iff* there are no elements common to all the visibility samples and 0 *iff* they are identical.

This metric admits an efficient implementation and works well in practice. However, it does not account for the angular distribution of error across the field of view. If error does occur, a more uniform distribution of this error has perceptual benefit, in contrast to a (potentially) clustered distribution.

3.4. Strict Error Metric

Before we detail our stratified error metric, we would like the reader to consider an alternative. We assume that the pixels (directional subdivisions) of each visibility cube are enumerated consistently from 1 to N. We define $s_j(i)$ as the polygon mapped by pixel i of visibility sample j. We define a new heuristic as follows:

$$Err'(s_{0..3}) = 1 - \frac{\sum_{i=1}^{N} \mathrm{diff}(s_0(i), s_1(i), s_2(i), s_3(i))}{N} \tag{2}$$

where

$$\mathrm{diff}(a,b,c,d) = \begin{cases} 1 & \text{if } a=b=c=d \\ 0 & \text{otherwise} \end{cases}$$

Err returns 0 *iff* every sample sees the same polygons, Err' return 0 *iff* each sample generates the same visibility cube. This adds an extra constraint: it is not sufficient for a polygon to simply be seen by all samples, it must be visible in exactly the same directions/pixels. This heuristic defines a measure of image based similarity. It is even possible for each sample to see exactly the same set of polygons, but for Err' to still return 1. It should be clear that if four samples see the same polygons, at the same pixels (i.e., they see the *same images*), then it is highly improbable that further refinement is necessary.

This measure is infeasible in practice, since 0 error will only occur when samples are taken very near each other. This would most likely lead to excessive sampling. In the next section we present a stratified metric that provides a beneficial compromise between Err and Err'.

3.5. Stratified Error Metric

The error distribution problem can be solved by partitioning each visibility cube into a fixed number of regular subregions. To ensure a reasonably uniform error distribution, the similarity among corresponding sub-regions must be above a certain threshold. Let $s_a(b)$ be the set of polygons visible from visibility sample a within the angular subregion b. The revised error metric over d sub-regions is defined as:

$$Err''(s_{0..3}) = max_{k=0}^{d}\left(1 - min_{i=0}^{3}\left(\frac{|\bigcap_{j=0}^{3} s_j(k)|}{|s_i(k)|}\right)\right) \quad (3)$$

If $d=1$, then Err'' is equivalent to that defined by Err. Similarly, if $d=N$ (for N pixels on the visibility cube), then this metric is equivalent to that defined by Err'.

In practice, a minimum (sample) distance constraint is necessary to enforce termination where an arbitrarily small movement in the view-point results in a large change in visibility. Without this, excessive subdivision may result. Although this implies that we may not refine areas of very high change, we take the union of these samples when calculating the visibility set for the cell thereby aggregating these differences, i.e., it is only those objects that are invisible at the corners, yet become visible within the approximated surface that will be erroneously omitted. Given that the size of this surface is small (as determined by the minimum distance constraint), the magnitude of the error and the (temporal) duration of the error *tends* to be small, although we have not yet found a theoretical bound on the maximum error.

3.6. Treating and Exploiting Manifold Meshes

Another approach that improves the error heuristic is the exploitation of specific scene properties. For instance, many scenes consist only of manifold surfaces with interiors that do not represent valid view-points. In this case, each of the four visibility samples can be classified as interior or exterior. Equation 3 is applied to each case independently. We denote the set of exterior and interior samples as E and I, respectively. Some simple properties are: $|E|+|I|=4$ and $E \cap I = \emptyset$. Two thresholds t_e and t_i are defined. We subdivide *iff* any of the following conditions hold:

- $|E|=1$ A single exterior point does not provide sufficient information for a final decision.
- $|E|>1$ and $Err''(E)>t_e$. The error threshold is exceeded for those samples at *valid* camera positions.
- $|I|>1$ and $Err''(I)>t_i$. The interior difference/error is high, therefore there is a good chance that intermediate visibility samples will be external. E.g., the interior samples might lie inside different objects.

To classify sample points as internal or external, a half-space comparison is made against the plane of any polygon in the visible set. A point in the same half-space as the normal of a visible polygon is considered exterior. Caveat: in practice, discretization errors typically cause a few pixels from back-facing polygons to be visible along the silhouette of an object. To counter this, the polygon that contributes the most pixels is chosen as the half-space classifier. This classifier selection can be efficiently integrated into the processing of the visibility cube buffers.

4. Algorithm Framework

We have shown how an adaptive algorithm may be used to sample visibility from a rectangular surface efficiently. In this section, we detail how this algorithm is used in traditional cell partitioning.

4.1. Visibility From a Volumetric Region

Consider a generic bounding box P for which a visible set has been computed. This can be partitioned by a single rectangle R, orthogonal to an axis of P. The partition can be situated anywhere along this axis. We refer to the two partitions as P^- and P^+. All sight lines from P^+ to P^- must intersect R. A polygon visible at the end of a sight segment is visible from all points along it. It follows that any polygon that intersects P^- and is visible from a point in P^+, must be visible from R. Now, the set of visible polygons $V(P^+)$ from cell P^+, can be expressed as:

$$V(P^+) = V(R) \cup I(P^+) \quad (4)$$

$V(R)$ represents the visibility from the rectangle R, and can be evaluated with the method discussed in Section 3.2. The set $I(P^+)$ is simply those polygons that intersect P^+, and can be computed with a simple polygon-cuboid intersection algorithm. Similarly, $V(P^-)$ may be expressed as:

$$V(P^-) = V(R) \cup I(P^-) \quad (5)$$

In general, the visibility set, $V(C)$, of a cell, C, is the union of those polygons that intersect C and those polygons visible from the surface of C. Next, we discuss at the implementation details of this hierarchical subdivision.

4.2. Hierarchical Subdivision

In practice, we begin with the scene bounding box as the root cell of the hierarchy. The hierarchy is evaluated by splitting in a top-down depth-first manner. The axis of subdivision is cycled as standard for *kd*-trres. Only the visibility in the current leaf nodes are maintained at any point. The subdivision process of a cell is illustrated in Figure 3. Our method for maintaining, reusing and distributing samples is also discussed in this section. The grid of cells is generally non-

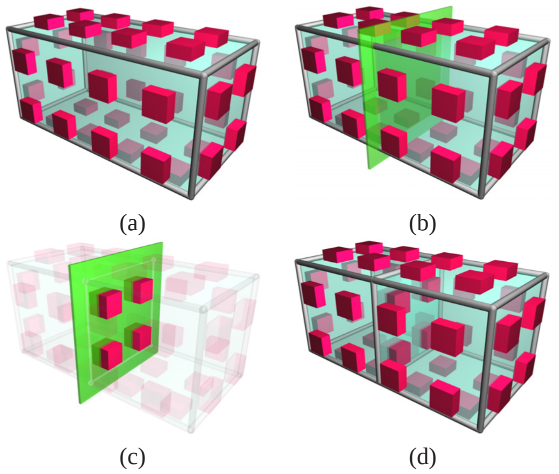

Figure 3: Hierarchical Subdivision. *(a) A typical cell within the hierarchy. (b) The chosen splitting plane within the cell. (c) New samples are generated on the sub-division. (d) The original cell is now partitioned. Note, that when a cell is subdivided, only those samples on the partition plane need to be evaluated. The samples shown in (a) are cached. Should both cells terminate subdivision after (d), then the samples shown in (c) would be obsolete and can then be deleted from the cache.*

uniform, since subdivision is terminated when the number of visible polygons falls below a set threshold, or *triangle budget*[KS99]. We adopted this threshold technique, from Saona-Vásquez et al.[SVNB99], since this is a straightforward solution to enforce upper bounds on rendering computations (and hence frame rates).

Infinite subdivision could occur if the number of polygons visible from some point is greater than the triangle budget. This is prevented by setting a *maximum depth* for the implicit binary hierarchy. In practice, this event is unlikely since the polygon throughput required for acceptable frame-rates is generally much greater than the number of polygons visible from any single point (or small neighborhood around a point). For certain scenes (e.g., a flight simulation over a terrain, where a lot of geometry is visible from high altitudes), a worst case situation can occur. A level of detail approach is often more suited for this type of scene.

When the subdivision of a cell ceases, all samples on the surface of the cell are aggregated. The union of the set of visible polygons for all samples is computed and stored with the cell.

An advantage of top-down hierarchical subdivision is that smaller sub-cells that do not contribute significantly to culling, are never evaluated (in contrast to the bottom-up subdivision of van de Panne and Stewart[vdPS99]). A second advantage is that previously computed information in the upper levels of the hierarchy may be exploited to accelerate the evaluation of the lower levels.

4.2.1. Superset Simplification

The set of polygons visible from a cell, C, is a superset of those visible from any view-point, q, within C: $V(q) \subseteq V(C)$ $\forall q \in C$. This allows the process of splitting a view cell to be optimised. When generating samples on the rectangle that splits the cell, only polygons known to be visible ($V(C)$), need be rendered. This implies that the cost of building a visibility sample decreases as the current depth in the binary hierarchy increases. This technique of superset simplification is also used by Durand et al.[DDTP00] and Saona-Vásquez et al.[SVNB99].

Although superset simplification can be applied to any from-region technique, the rate of decay for the size of the superset as subdivision occurs is maximal only for exact and aggressive algorithms. Indeed, many conservative algorithms perform poorly when applied to large view cells.

Since our visibility preprocess is aggressive, there may be potential omissions of visible polygons in the superset as calculated. However, due to the effectiveness of our sampling heuristic, such error is minimal and superset simplification yields a significant preprocessing performance benefit. Should a significantly faster output-sensitive renderer be used to perform the sampling, superset sampling may become redundant.

4.2.2. Cache Management

Using a top-down approach for many existing algorithms is a non-trivial optimisation problem. Firstly, it is necessary to consider the efficiency of the algorithm with respect to the size of view cells in the upper level of the hierarchy since these tend to be rather large. Secondly, it is important to consider whether or not sufficient benefit is gained by evaluating a given cell, rather than evaluating its children directly.

Typical implementations result in a set of cells being generated on a uniform grid. Each grid element can then be expanded into a hierarchy. Cohen-Or *et al.*[COFHZ98] use a fixed two level hierarchy. Durand et al.[DDTP00] also use an initial grid.

Our approach, however, has an interesting property that allows us to evaluate a parent cell, and then split the parent (if so desired) into two child cells, at the *same* cost as generating the child cells initially. This requires the ability to partition an existing rectangular region, while being able to disseminate the correct visibility information to the partitions, without (significant) additional computation. Since we use a sample based approach, all the samples belonging to the parent region are distributed to their incident partitions. In order to keep the cost low, we *cache* all samples until they are no longer required.

The process proceeds as follows (illustrated in Figure 3): First, we assume that the visibility (surface samples) from some cell C has been computed (Figure 3a). Second, if subdivision is indicated (depending on subdivision criteria), a splitting plane is chosen (by a heuristic) that splits the cell into C^- and C^+ (Figure 3b). Third, the required samples on the shared boundary of C^- and C^+ are computed (subject to the visibility set $V(C)$) (Figure 3c). Fourth, the samples in the negative and positive half-spaces (as defined by the splitting plane) are propagated to C^- and C^+, respectively (Figure 3d). Finally, cell C is deleted.

In order to save memory resources, it is necessary to compute which of the samples associated with the current (now finalised) cell are redundant, and which will be used again. We associate a counter with each sample. When subdivision terminates for a cell, the counter of each sample on the cell is updated. Once all cells adjacent to a sample have finished subdividing, the sample is deleted. The depth-first traversal that builds the hierarchical subdivision thus allows for the early removal of samples. Also note that all sample data is RLE compressed.

5. Algorithm Analysis

The adaptive nature of the algorithm makes it difficult to obtain a useful upper bound. Given that k samples are generated in total, the overall complexity is $O(kf(n))$, where $O(f(n))$ is the complexity of the from-point approach used. If $O(f(n))$ is fully output sensitive, the complexity becomes $O(kv)$. Using superset simplification, our implementation approaches $O(kv)$ from above, since the first few samples are not output sensitive.

The value of k itself is a complex function of the scene size and particular scene configuration. Intuitively, k characterizes the degree of variation of visibility within a scene. It should grow as the combinatorial complexity of those visibility events that result in the appearance or disappearance of a polygon in a scene grow. The growth of k would be far less than proportional to the number of these events, however, since a single visibility sample can pick up the change in visible polygons resulting from many such visibility events. For our scenes, of the order of millions of polygons, results have indicated that for low error in practice, k is in the order of tens of thousands.

6. Implementation

The computation of a visibility cube consists of six renderings of scene geometry from a single point. For each render, the frame buffer needs to be read in order to obtain the visible polygon indices. In this section we examine the performance issues and propose several optimizations.

Any acceleration technique that can be applied to traditional point rendering(e.g., Zhang et al.[ZMHI97] or Govindaraju et al.[GSYM03]) can also be applied to visibility cube rendering. The algorithm presented in this paper therefore scales with the performance of the point renderer utilised. We incorporate frustum culling into our implementation and we utilise information already computed by the preprocess to accelerate rendering (see Section 4.2.1). This occlusion method can be used to enhance most point based visibility techniques. Using our GeForce4 Ti 4600 we achieve 17 million triangles per second throughput.

Frame-buffer reading is often a bottle neck. Wonka et al.[WWS00] claim that it takes approximately 54% of their run-time. This is most likely due to their only rendering simplified scenes (their 8 million triangle scene is actually represented by a much smaller building "facade"), and thus may exaggerate the frame buffer read times for general scenes. Frame buffer reading consists of a considerably smaller part (20%) of our run-times, although this may grow with faster hardware and/or a superior point rendering implementation. The performance of frame buffer reads has not improved at the same rate as triangle rendering, is due to limitations on bus technology. This should be alleviated soon with the advent of the *PCI Express* bus[Int03].

7. Results

In this section we present empirical results illustrating the practicality of our aggressive algorithm. We present results to quantify the performance of our technique. We also consider it necessary to quantify the error produced, and we give empirical evidence showing that for the scenes tested, error can be contained in order to give acceptable image quality.

7.1. Performance

We begin by showing that the algorithm can be used to preprocess scenes that cannot be processed effectively by existing solutions. We use a *large* forest scene consisting of 5 million polygons (see Figure 4). Nearly 200 trees, each highly detailed, consisting of 25 000 polygons each. A "small" forest scene, consisting of 2 million polygons is also tested. This scene consists of 80 trees, also of 25 000 polygons each. Note: Internally, we make no use or assumption of instanced geometry. For reference, we pre-process the forest scene used by Durand[Dur99]. This scene consists of approximately 1450 trees, at 1000 polygons each Although, simpler in terms of polygon granularity, this scene has a far higher depth complexity than our forest scenes, implying that more culling can be performed. We also test a large, complex *town* scene used by Nirenstein et al.[NBG02] comprising 1.35 million triangles This scene is realistic, and consists of a mix of simple and detailed objects. The hardware

Exp.	Scene	Size	Threshold	Time/Cell	No. Cells	Visible Set
1	Forest	5m	0.2	9.23s	512	8.68%
2	Forest	2m	0	5.48s	512	14.84%
3	Forest	2m	0.999	4.17s	512	13.63%
4	Durand et al.	1.45m	0.99	2.57	400	0.79%
5	Town	1.35m	0	2.5s	512	1.45%
6	Town	1.35m	0.999	2.46s	512	1.35%

Table 1: Aggressive Algorithm/Preprocess – Performance Results. *The* Experiment *column is the experiment reference number. Each experiment number corresponds to one preprocess and analysis. They may be used to cross-reference the error results in Table 2. The* Scene *column indicates the type of scene. The* Size *column gives the size of the scene (in triangles (m = millions)). The* Threshold *column indicates the error threshold used in the preprocess (see Section 3.5). The* Time/Cell *column gives the time taken per cell. The* Number of Cells column *shows the number of cells into which the bounding box was subdivided (*$512 = 8 \times 8 \times 8$ *and* $400 = 20 \times 20$*). The* Visible Set *column provides the percentage of visible geometry averaged through all cells. The scenes were all sampled using* 512×512 *pixel item buffers.*

Exp	Avg. Error	Max. Error	CR Count	Avg. Max. CR	Tot. Max. CR
1	0.338% (886)	6.293% (16497)	267.27	0.032% (84)	0.607% (1591)
2	0.315% (826)	1.3% (3408)	453.42	0.016% (42)	0.09% (236)
3	0.888% (2328)	5.88% (15414)	605.91	0.083% (218)	0.66% (1730)
4	0.561% (1471)	1.71% (4490)	110.23	326.23% (326)	0.915% (2400)
5	0.116% (304)	1.034% (2711)	14.2	0.107% (80)	0.745% (1953)
6	0.117% (307)	1.034% (2711)	14.97	0.105% (275)	0.745% (1953)

Table 2: Aggressive Algorithm/Preprocess – Error Results. *The* Experiment *column is the experiment reference number. Each experiment number corresponds to one preprocess and analysis. They may be used to cross-reference the performance results in Table 1. The* Average Error *column gives the average image error for a sampled camera path (details in Section 7.2). The values in parentheses are the absolute number of pixels corresponding to the percentage (a* 512×512 *pixel view is used for our tests). The* Maximum Error *column is the image error for the frame with the largest error. The* CR Count *column gives the minimum number of connected regions (CR) into which the erroneous pixels may be partitioned. The CRs are averaged over the frames in our test path. The* Average Maximum CR *column is the average size of the* largest *CR in each frame. The* Total Maximum CR *column is the size of the maximum CR over all frames of the walk through. The scenes were all sampled using* 512×512 *pixel item buffers.*

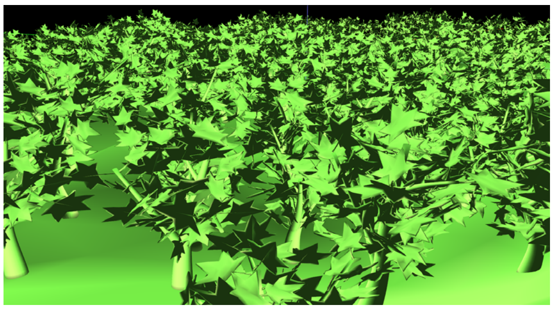

Figure 4: Test Scene – Forest Model (5m). *A very complex forest model consisting of 5 million polygons. Each tree consists of 25 thousand polygons. The image shows the output of our algorithm. 11.9% of the scene is rendered from the region containing the view point.*

used for our tests is a Pentium 4 1.7Ghz, with an NVidia GeForce4 Ti 4600 and 1.2GB of memory.

First, we consider the visible set size. Being an aggressive algorithm, it is to be expected that the visible subset is less than or equal to the results of an exact solution. Durand et al.[Dur99] cull 75% of their scene, resulting in a visible set of 25%. In comparison, we are able to cull 99.21% from the model (see Figure 5). Where the extended projection algorithm took 17 seconds per cell, our technique takes 2.57 seconds per cell. It is difficult to compare timings directly due to hardware differences. For rendering, the SGI Onyx2 can render 11m tri/sec in comparison to our 17m tri/sec.

Second, we consider the effectiveness of the algorithm when applied to our large forest scene. 91.32% of the scene is culled on average (see Figure 6a). This allows for an acceleration of 11.5 times that of naïve rendering. We have no knowledge of any existing alternative algorithm capable of preprocessing this scene to a significant degree. Indeed, although the exact algorithm presented by Nirenstein et al.[NBG02] is capable of processing it, without the possibility of error, the time required is excessive for a single workstation. The large forest model is processed at a rate of 9.23 seconds per cell for a total time of 79 minutes.

The town scene is processed at 2.5 seconds per cell for a

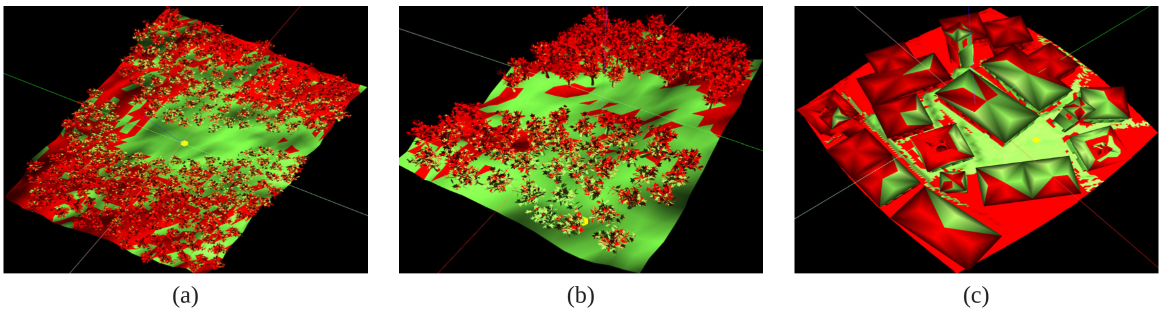

Figure 6: Aggressively Culled Scenes – Large Forest, Small Forest and Town. *Sample output from our aggressive algorithm. Green polygons are visible and red polygons are invisible. The view point used is that of the yellow sphere. (a) 4 million polygons are culled (21.6% of the scene is visible) from the given view point. (b) 1.9 million polygons are culled (10.3% of the scene is visible) from the given view point. (c) 1.1 million polygons are culled (2.5% of the scene is visible) from the given view point.*

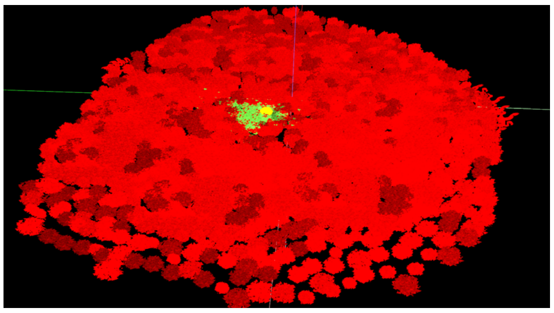

Figure 5: Test Scene – Aggressive Culling of Durand's Forest. *The forest scene used by Durand* et al.*[Dur99]. From the cell of the given view point (yellow sphere), 0.8% of the scene is visible. Visible and invisible polygons are shown in green and red respectively.*

total time of 21 minutes. An average of 1.45% of the scene was determined to be visible (see Figure 6c).

Using the error threshold, we see that it is possible to trade quality for performance. Using the 2 million polygon forest scene we see that a 32% performance increase was gained by increasing the threshold. This tradeoff was less effective for the town scene. We discuss this and the quality tradeoffs in Section 7.2. Given the exponential shape of this sensitivity, end user application (i.e., a person wishing to preprocess a model) should use a logarithmic scale to manipulate the threshold scale.

In order to ascertain scalability with respect to the number of cells, we have executed the preprocess for our 2 million polygon forest scene using a varying number of cells. The results are shown in Figure 7. The decrease in per-cell evaluation results from the increased output sensitivity as the hierarchy depth is increased.

7.2. Image Error

The error results are found by analyzing a walkthrough of several thousand frames. We present error results for the various models in Table 2. For each frame, the visible geometry is rendered in green (flat shaded and unlit). Similarly, the geometry determined to be invisible is rendered in red. We count those pixels that are red, and consider this to be the number of erroneous pixels or *error* in the frame. See Figure 8 for an example.

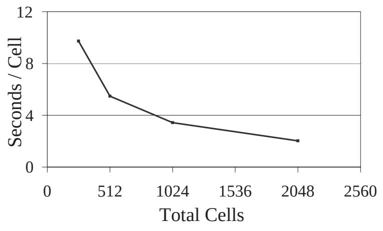

Figure 7: Scalability by Cell. *The average time taken per cell (vertical) is plotted against the number of cells by which the bounding box is subdivided. As evidenced, there is rapid decay in the time taken per cell. The model used is our 2 million polygon forest model.*

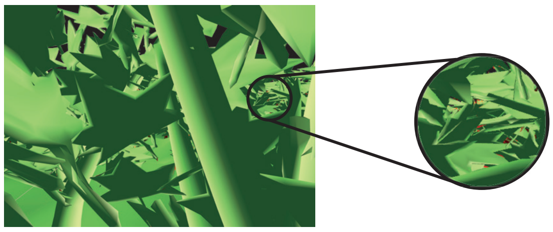

Figure 8: Error Measure. *A* 1600 × 1200 *screen shot from a walk through. Green polygons are those determined to be visible by our aggressive algorithm, while the red polygons are determined to be invisible. The appearance of red pixels mark erroneous rendering. Shading and lighting are used for this image, although our automated error evaluator uses constant colouring.*

We also utilise several *connected region* (CR) metrics. The *CR count* is the number of connected regions of erroneous pixels (found per frame, by recursive search). Given a number of erroneous pixels, the CR count constitutes information about the distribution of the error on the display. A high CR count (relative to error) implies that the error is fairly scattered, and will most likely manifest as noise (more easily filtered by our visual system than coherent artefacts[Coo86]). A low CR count (with respect to error) implies that the errors are clustered together, and are more likely to be noticeable. We present the average error and the average CR count over all frames. We also include the largest connected region in the walkthrough.

Consider the five million triangle forest model. An average of 8.68% of the geometry is visible. This is a subset of the "truly" visible geometry. When evaluating the results of a walk through, we found that 0.337% of the average frame comprised erroneous pixels. In isolation, this appears to be very little, however the distribution of error is also important. Indeed, the *largest* error on any frame was a significantly higher 6.293%. Our experiments have shown the error to comprise, of an average of 267.27 disjoint connected regions. Indeed, on average, the largest connected region in each frame (only those frames with error are considered), is only 0.016% of the frame. The largest connected region in any frame in the walkthrough (consisting of several thousand frames), is 0.607%.

The other experiments show how our technique fared for other scenes, using small and large thresholds. The town scene resulted in lower average and maximum error, while the forest scene resulted in more fragmented errors.

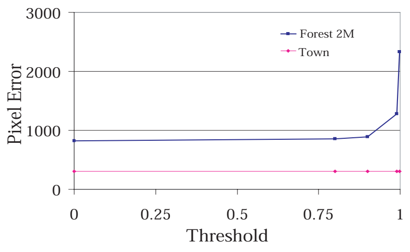

Figure 9: Error vs. Threshold. *The average pixel error for a walk through as a function of threshold. It should be noted that a higher threshold necessarily implies that a subset of the samples of a lower threshold are utilised, thereby making any error/threshold graph monotone. This allows only a few samples to depict the trend.*

We show how error is affected by error thresholding. The forest scene is particularly sensitive to this threshold, as depicted in the graph of Figure 9, while the town scene is not. The reason for this disparity is that the forest allows for gradual changes in visibility as a view sample moves through a region. In contrast, the town scene can have a very large change in visibility from one nearby view sample to another (e.g., if an adjacent view sample crosses a wall). This large change in visibility will force any reasonable threshold to continue subdividing, thus resulting in the constant, low image error shown in the figure.

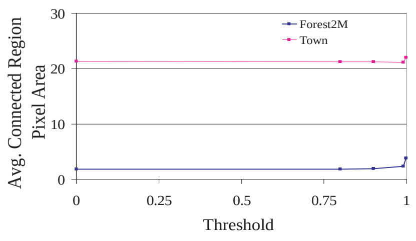

Figure 10: Average Connected Region vs. Threshold. *The average size of the connected error regions (in pixels) for a walk through as a function of threshold.*

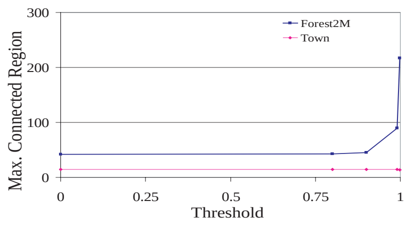

Figure 11: Maximum Connected Region vs. Threshold. *The largest connected error region (in pixels) for a walk through as a function of threshold.*

Similarly, we show how the threshold relates to the average size of the connected regions in Figure 10, and the maximum size of the connected regions in Figure 11. Despite the fact that the average error is lower, the town scene results in a larger average connected region error, since when an error does occur, it tends to be larger, because the town scene contains larger triangles than the forest. The maximum connected region error is larger for the forest scene, although this is because several viewpoints of the test path contain clusters of sub-pixel size polygons. Such regions are usually perceived as noise and polygon omissions here do not impact significantly on image quality.

8. Conclusion

We have presented an novel aggressive visibility algorithm that efficiently preprocesses difficult scenes at the cost of possible image error. The algorithm exploits graphics hardware for increased performance.

Adaptive sampling is used to obtain the set of visible geometry. An efficient scheme is used to share samples between cells and to minimise their memory resident lifespan. We have developed an efficient heuristic to guide the sampling process with the goal of minimizing error.

Results show this technique to be nearly three orders of magnitude faster than exact techniques, while still not suffering from the large overestimation exhibited by conservative techniques. Also, we show that this technique can be used to preprocess highly complex scenes effectively. Image error is minimal resulting in images that are more than sufficient for most applications.

Acknowledgements – We would like to thank James Gain for his input, and for building the town model. We would also like to thank the NRF for supporting this research.

References

[ARJ90] AIREY J. M., ROHLF J. H., JR. F. P. B.: Towards image realism with interactive update rates in complex virtual building environments. *1990 Symposium on Interactive 3D Graphics 24*, 2 (1990), 41–50. 2

[ASVNB00] ANDÚJAR C., SAONA-VÁZQUEZ C., NAVAZO I., BRUNET P.: Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum 19*, 3 (August 2000), 499–506. 3

[Bit02] BITTNER J.: *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University in Prague, October 2002. 1, 2

[BMH98] BARTZ D., MEISSNER M., HÜTTNER T.: Extending graphics hardware for occlusion queries in opengl. *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware* (1998), 97–104. Portugal. 3

[BWW01] BITTNER J., WONKA P., WIMMER M.: Visibility preprocessing for urban scenes using line space subdivision. In *9th Pacific Conference on Computer Graphics and Applications* (2001), IEEE, pp. 276–284. 2

[CG85] COHEN M. F., GREENBERG D. P.: The hemi-cube: A radiosity solution for complex environments. *Computer Graphics (Proceedings of SIGGRAPH 85) 19*, 3 (August 1985), 31–40. San Francisco, California. 2

[COCSD03] COHEN-OR D., CHRYSANTHOU Y., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. *IEEE TVCG 9*, 3 (2003), 412–431. 1, 2

[COFHZ98] COHEN-OR D., FIBICH G., HALPERIN D., ZADICARIO E.: Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum 17*, 3 (1998), 243–254. 2, 5

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG) 5*, 1 (1986), 51–72. 9

[DDTP00] DURAND F., DRETTAKIS G., THOLLOT J., PUECH C.: Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000* (July 2000), 239–248. 2, 5

[Dur99] DURAND F.: *3D Visibility, analysis and applications*. PhD thesis, U. Joseph Fourier, 1999. http://graphics.lcs.mit.edu/˜ fredo. 6, 7, 8

[GSF99] GOTSMAN C., SUDARSKY O., FAYMAN J. A.: Optimized occlusion culling using five-dimensional subdivision. *Computers & Graphics 23*, 5 (1999), 645–654. 2

[GSYM03] GOVINDARAJU N. K., SUD A., YOON S.-E., MANOCHA D.: Interactive visibility culling in complex environments using occlusion-switches. In *Proceedings of the 2003 symposium on Interactive 3D graphics* (2003), ACM Press, pp. 103–112. 6

[HA00] HOLZSCHUCH N., ALONSO L.: Using graphics hardware to speed-up visibility queries. *Journal of Graphics Tools 5*, 2 (2000), 33–47. 2

[Int03] INTEL: Intel developer network for pci express architecture, 2003. www.intel.com/technology/pciexpress/devnet. 6

[KCCO01] KOLTUN V., CHRYSANTHOU Y., COHEN-OR D.: Hardware-accelerated from-region visibility using a dual ray space. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering* (June 2001), Eurographics, pp. 205–216. 2

[KS99] KLOSOWSKI J. T., SILVA C. T.: Rendering on a budget: A framework for time-critical rendering. *IEEE Visualization '99* (October 1999), 115–122. 5

[LSCO03] LEYVAND T., SORKINE O., COHEN-OR D.: Ray space factorization for from-region visibility. *ACM Transactions on Graphics 22*, 3 (2003), 595–604. 2

[NBG02] NIRENSTEIN S., BLAKE E., GAIN J.: Exact from-region visibility culling. In *Proceedings of the 13th workshop on Rendering* (June 2002), Eurographics Association, pp. 191–202. 1, 2, 6, 7

[Nir03] NIRENSTEIN S.: *Fast and Accurate Visibility Preprocessing*. PhD thesis, University of Cape Town, October 2003. 2

[Pla92] PLANTINGA H.: An algorithm for finding the wealky visible faces from a polygon in 3d. In *Fourth Canadian Conference on Computational Geometry* (1992), pp. 45–51. 3

[SDDS00] SCHAUFLER G., DORSEY J., DECORET X., SILLION F. X.: Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000* (July 2000), 229–238. 2

[SLCO∗04] SAYER E., LERNER A., COHEN-OR D., CHRYSANTHOU Y., DEUSSEN O.: Aggressive visibility for rendering extremely complex plant ecosystems, 2004. http://www.cs.tau.ac.il/˜ alan/aggressive.htm. 2

[SVNB99] SAONA-VÁSQUEZ C., NAVAZO I., BRUNET P.: The visibility octree: a data structure for 3d navigation. *Computers & Graphics 23*, 5 (October 1999), 635–643. 2, 5

[TS91] TELLER S. J., SÉQUIN C. H.: Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91) 25*, 4 (1991), 61–69. 2

[vdPS99] VAN DE PANNE M., STEWART J.: Efficient compression techniques for precomputed visibility. *Eurographics Rendering Workshop 1999* (June 1999). Granada, Spain. 2, 5

[WM03] WILSON A., MANOCHA D.: Simplifying complex environments using incremental textured depth meshes. *ACM Trans. Graph. 22*, 3 (2003), 678–688. 2

[WWS00] WONKA P., WIMMER M., SCHMALSTIEG D.: Visibility preprocessing with occluder fusion for urban walkthroughs. *11th Eurographics Workshop on Rendering* (June 2000), 71–82. 2, 6

[Zha98] ZHANG H.: *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, UNC at Chapel Hill, 1998. 3

[ZMHI97] ZHANG H., MANOCHA D., HUDSON T., III K. E. H.: Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97* (August 1997), 77–88. 6