Aggressive Visibility Pre-processing with Adaptive Sampling

S. Nirenstein, J. Gain and E. Blake

Department of Computer Science, The University of Cape Town
{snirenst,jgain,edwin}@cs.uct.ac.za

Technical Report
CS01-01-00

Abstract. At the expense of a small error in visibility classification, we remove *all* invisible polygons. Thresholding and heuristics allows fine control over the behaviour of this error. Our technique is applicable to both concave and convex polygons. It exhibits sublinear computational complexity in the number of scene polygons and logarithmic complexity in the number of cells, while effectively exploiting graphics hardware.

The technique is classified as an aggressive from-region method in that it falsely excludes a small subset of visible polygons and estimates visibility on a per cell basis. A *kd*-tree hierarchy of visibility cells is built by sampling visibility across their surfaces adaptively. Sampling is guided by a novel error heuristic and produces, by adaptive sub-division, a quad-tree like structure.

We have applied our technique to both a standard building scenes and a highly complex natural scene. The results demonstrate that significant culling (94.5% average) with low error rates (0.687% average) can be achieved with such scenes in a reasonable period of time.

1 Introduction

Despite the power of modern graphics hardware, software techniques are still required to reduce highly complex polygonal scenes to a manageable sizes. Visibility culling is one such technique. Polygons which are not visible from a particular viewpoint do not contribute to the final image, and may therefore be removed from consideration at an early stage in the graphics pipeline. The challenge in visibility culling is to remove as many unseen polygons as efficiently as possible.

Visibility culling may be solved as an (off-line) pre-process, at run-time, or as a combination of the two. We advocate the use of a pre-processed visibility solution, since a single (albeit lengthy) pre-process may yield considerable run-time improvement. More importantly, a superior visibility solution demands non-trivial resources, and cannot generally be implemented at run-time. However, one disadvantage of an off-line solution is that dynamic scenes become more difficult to handle. As a consequence, we only consider the pre-processing of static scenes.

An object can be deemed unseen if it is outside the view frustum or is *occluded* by some combination of scene polygons. In this paper, we consider occlusion only, since frustum culling may be applied at run-time. We note that, in a scene populated by manifold (closed) objects, back-facing polygons are a subset of occluded polygons because they are always blocked by the front faces of their associated object.

To date, there are no tractable exact visibility culling solutions for large models. The difficulty of such a solution is briefly discussed in [14]. This has encouraged the development of many approximations. In such solutions, two types of error occur: *false visibility*, where invisible polygons are included (considered visible) and *false invisibility*, where visible polygons are excluded (classed as invisible).

False visibility errors result in sub-optimal performance caused by the unnecessary rendering of unseen polygons. On the other hand, false invisibility errors result in sub-optimal image quality due to the omission of visible polygons. Techniques which potentially cause *only* false visibility errors are termed *conservative*. Traditionally, techniques which potentially cause false invisibility errors have been called *approximate* solutions [10]. However, the latter techniques usually result in both types of error. Instead, we refer to techniques which potentially cause false invisibility errors, but never false visibility errors, as *aggressive visibility* solutions. The relationships between these classifications is illustrated in Table 1.

	Opt. Perf.	Sub-opt. Perf.
Opt. Quality	**Exact**	**Conservative**
Sub-opt. Quality	**Aggressive**	**Approximate**

Table 1. Properties of our four visibility solution classifications.

It is impossible to explicitly process the infinite number of camera positions in the view-point space (VPS). Visibility pre-processing techniques therefore partition the VPS into a finite number of sub-spaces, or *cells*. We define the set of visible polygons belonging to a cell as the union of the polygon sets visible from each and every point in that cell. Our aggressive and conservative visibility terminology applies to both point and cell based visibility. Cell based algorithms are also referred to in the literature as *from-region* visibility techniques.

In this paper we outline a novel solution to visibility culling. Our approach is an aggressive from region method. We pay special attention to pre-processing performance and error minimization via adaptive techniques. Our principle contributions are:

1. *Efficient culling*. We use aggressive culling to remove *all* redundant polygons from a cell's visibility set.
2. *Generality*. No *apriori* scene structure is assumed. Our algorithm is applicable to both convex and concave polygons.
3. *Theoretical efficiency and scalability*. Our novel sampling approach enables a computational complexity which is logarithmic in the number of cells, where other techniques are generally linear. Also, the computational complexity is roughly sub-linear in the number of polygons rendered for a fixed number of cells, as contrasted with usually linear to super-linear methods.
4. *Exploiting hardware*. We use the considerable polygon rendering capabilities of modern graphics hardware to perform sampling rapidly. This aids both performance and accuracy.
5. *Error Control*: We provide both a novel adaptive sampling method and an associated error heuristic for controlling sample refinement. Sampling is only refined in areas of potential error. This prevents redundant sampling.

1.1 Paper Overview

We begin with a critical discussion of previous work in the field of visibility pre-processing. This is followed, in Section 3, by a demonstration of the use of adaptive sampling to solve visibility from a two-dimensional rectangular region in space. In Section 4, we discuss and justify our error estimation heuristic. In Section 5, we show how this "from-2D-region" technique may be used to generate a top-down hierarchical subdivision of a scene efficiently. An important optimization is presented in Section 6. In Section 7, we analyze the scalability and complexity of this algorithm. In Section 8, we present statistical results showing the performance and error results of our technique. Finally, we conclude in Section 9.

2 Previous Work

The field of visibility has advanced considerably over the past decade. It would be impossible, in this context, to cover the entire discipline adequately. Instead, we focus on "from-region visibility pre-processing". For an in-depth survey of the broader field, including real-time techniques, analytic visibility and compression, the reader is referred to the excellent surveys of Durand [7] and Cohen-Or *et al.* [5].

Cell-portal rendering attempts to establish the relative visibility of entire cells. Here, the visibility of one cell from another depends on the existence of a line of sight between them intersected only by portals, which are non-opaque boundaries between cells. Airey *et al.* [1] use an approximate ray-casting approach. Teller [16, 17] extend their work by deriving an analytic solution to the problem. These algorithms are well suited to architectural scenes, but do not provide a general visibility solution.

Cohen-Or *et al.* [6] and Saona-Vásquez *et al.* [15] provide slightly more general solutions restricted to convex occluders. They classify an object as invisible from a particular view-cell if a single polygon occludes the object from all points inside that view-cell. This is highly conservative and only achieves significant results if view-cells are small relative to the size of scene polygons. For highly detailed scenes, this often requires a large number of view-cells and hence prohibitively long run-times.

To counter this, it is necessary to consider aggregate occlusion by a set of smaller occluders. It is worth mentioning that such *occluder fusion* was first treated (implicitly) by the techniques of Airey [1] and Teller [16, 17]. Recent work has focused either on fusing occluders, or on the construction of larger virtual occluders, which represent the occlusion of multiple smaller occluders.

Durand *et al.* [7, 8] have recently developed a general solution to visibility culling, using an extended projection operator. With respect to a view cell, this operator projects occluders and occludees onto a selected plane. The extended projection of an occluder is the *intersection* of its projections from all viewpoints within the current cell. The extended projection of an occludee, however, is the *union* of its projections. An occludee is blocked if its extended projection is covered by that of an occluder. This is a natural extension of point based to "from region" visibility.

With this technique it is possible to achieve occluder fusion by considering the union of extended projections. In order to maximize the size of an extended projection (and therefore occlusion), it is necessary to select multiple projection planes. Occlusion is aggregated from preceding planes by using a reprojection operator which may be implemented in hardware efficiently.

Koltun *et al.* [12] make use of separating lines to build larger, more effective *virtual occluders*, which represent the occlusion of many smaller occluders. They note that by building their virtual occluders as a pre-process and performing the occlusion culling at run-time (on a per cell basis), the cost of storing the visibility sets for each cell is removed. Unfortunately, they only implement a $2\frac{1}{2}$D (height field) solution. Although the theory extends to 3D separating planes, the general method is far more complex and may not be tractable.

Law and Tan [13] use *occlusion preserving simplification* to generate lower level of detail (and thus larger) polygon occluders. They show how *occlusion preserving* simplification may be used to ensure the conservativity of this approach. This technique improves occlusion culling, however, the occlusion achieved by the fusion of these coarser representations is not considered. This technique may prove more efficient if integrated with that of Wonka *et al.* [19].

Wonka *et al.* [19] have another $2\frac{1}{2}$D solution which is not easily extensible to 3D. They begin by *shrinking* a subset of occluders. They then sample visibility (effectively

fusing occluders) along the surface of the view cells and aggregate invisible polygons. Occluder shrinking allows the sampling process to maintain conservativity. There is a trade-off between samples required and the degree of shrinkage. This algorithm is highly conservative unless occluders can be aggregated before shrinking (as done for their city model).

The techniques surveyed above are all conservative in nature. They provide optimal image quality, but there is a large margin between the size of the visibility sets they generate and those of an exact solution. The extended projections of Durand *et al.* are most likely to determine the tightest visibility sets of the techniques mentioned. However, even here improvement is possible. In particular, the union of the extended projection of many objects is at most, but generally not, as large as the extended projection of the union of those same objects. Thus, the largest possible occluders are not being used effectively. This is ameliorated by the use of reprojection on many planes. However, their solution remains sub-optimal.

Recently, there has been concerted development of techniques that admit false invisibility errors. Run-time solutions which support approximate culling are provided by Zhang *et al.* [20] and Bartz *et al.* [3]. Andújar *et al.* [2] use *hardly visible* sets to cull or simplify scenes where only a small proportion of the geometry is visible. Klosowski and Silva [11] present a *prioritized layer projection* algorithm, which established a heuristic priority ordering that attempts to draw visible polygons first. This is an excellent time-critical solution, since reasonable results are obtained if rendering is prematurely terminated. The motivation behind these techniques is that rendering costs may be reduced by removing objects which do not contribute significantly to the image.

Van der Panne and Stewart [18] investigate compression techniques for pre-computed visibility data. As a reference algorithm they implement a naïve bottom-up aggressive technique which uniformly samples visibility using an item buffer.

Gotsman *et al.* [10] present a novel sample-based visibility solution. They use a 5D sub-division over three spatial and two angular dimensions. Each 5D cell maps to a *beam* in 3D space. The use of two angular divisions is intended to accelerate frustum culling at run-time. To determine from-region visibility, rays are cast from a random point in the cell to random points on an object's bounding box.

A statistical model based on whether the rays do or do not reach the target object, this is used to decide if the object is visible, invisible or whether more rays need to be cast. Error thresholding allows a trade-off between pre-processing time and accuracy. This is probably the most accurate solution to date.

In provenance our work is closest to that of van der Panne and Stewart [18] and Gotsman *et al.* [10]. However, unlike the former our technique is adaptive and top-down. With regard to the latter, our approach differs in several key respects: our cell construction admits a logarithmic time dependence on the number of cells; we exploit graphics hardware for visibility sampling (as do Wonka *et. al.* [19]) and this allows rapid per triangle (instead of conservative per object) consideration; and our heuristics focus directly on reducing error.

3 Sampling for "From-Region" Visibility

In this section our novel sampling approach to "from-region" visibility is outlined. We describe a sampling method, which is based on the hemi-cube [4] and exploits rendering hardware. We demonstrate its application in deriving an aggressive visibility set from a rectangular region in 3D space. Finally, adaptive subsampling is introduced.

A point sample can be defined as the set of polygons visible from a given point. A visibility cube (strongly related to a radiosity hemi-cube) is used to generate such

samples. This is created by treating each of the six sides of a tiny cube enclosing the (view) point as independent depth buffers onto which the scene is rendered. Depth buffers are supported by modern graphics hardware and ensure that only the pixels of polygons visible from the point in question are rendered. Each polygon is assigned a distinct colour index and this allows a given pixel to be uniquely associated with a visible polygon. The set of polygons which contribute to the frame-buffers of a visibility cube is then treated as the set of visible polygons.

The concept of a "from-region" visibility set can be extended to encompass a rectangle. This is simply the union of all possible point samples in the rectangular region. Since there are an infinite number of these points, an exact brute force evaluation is infeasible. Instead, we form the union of a finite subset of points.

This is effectively a sampling over the rectangluar domain. Insufficient sampling leads to aliasing artefacts, which manifest as the non-inclusion of visible polygons (*false invisibility* error).

A naïve solution is to perform uniform sampling (Figure 1a). With this approach, under-sampling may result in unacceptable error, while over-sampling may lead to prohibitive execution costs. An adaptive sampling method is called for.

To begin with, point samples are evaluated at the corners of the rectangle. Then a decision is made whether to refine (subsample) the rectangle into four subregions based on an error heuristic and user specified threshold (covered in Section 4). This subdivision proceeds recursively, in a manner equivalent to the depth first generation of a quad-tree. A rectangular subregion, with point 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 quadtree, so it is important to cache point samples in order to prevent redundant computation. A typical adaptive subdivision is illustrated in Figure 1b.

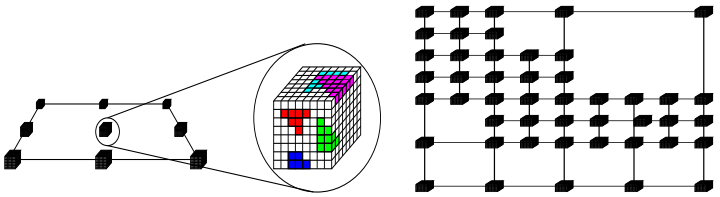


Fig. 1. [a] A uniform distribution of visibility cubes on a 2D surface. [b] A non-uniform distribution of visibility cubes generated by an adaptive sub-division, which attempts to minimize error. The sub-division effectively builds a quad-tree.

4 Error Heuristic

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 error measure, which seeks to establish, given four corner point samples, if any interior view points are likely to contain error.

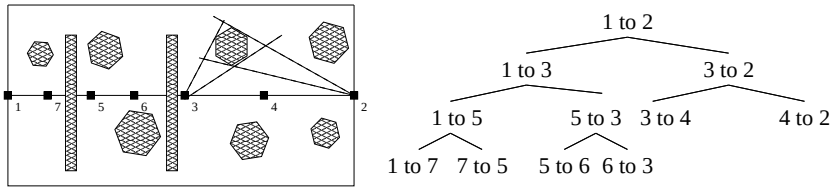
Platinga [14] proposes an algorithm with which exact visibility from a polygon may be determined by enumerating and traversing all visibility events over the polygon. These events define a sub-division over the polygon consisting of partitions of qualitatively equivalent visibility. Unfortunately, the algorithm proposed has $O(n^4 \log n)$ time complexity. Implicitly, our algorithm attempts to efficiently reconstruct this subset of the aspect graph [9] through adaptive sampling.

Ideally, areas with high frequency fluctuations in visibility should be more highly

sampled. Our first criteron, therefore, explicitly encodes the normalized difference between visibility samples. Given four point samples, s_0, s_1, s_2, s_3, we define:

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

Equation 1 is 1 *iff* there are no elements common to all the visibility samples and 0 *iff* they are identical. Figure 2, demonstrates the use of an error threshold on Equation 1 to control 2D subdivision.



(a) (b)

Fig. 2. [A] A 2D scene (plane view) consisting of hexagons and rectangles, bisected by a cell boundary (the horizontal partitioning line) with 7 sample points (visibility cubes). Assuming an error threshold allowing roughly a three edge disparity and beginning from samples 1 and 2, the subdivisions 3-7 are generated in order. The first sub-division results from the dissimilarity of sample points 1 and 2. Our error heuristic would return 1 to force a sub-division. [B] A binary tree generated by this 2D adaptive subsampling along the cell boundary.

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 merit, in contrast to a clustered distribution.

The distribution problem can be solved by dividing each visibility cube into a fixed number of angular sub-regions[1]. To ensure a uniform error distribution, the error among corresponding sub-regions must fall below a certain threshold. Let s_{ab} be the the visibility set of point sample a in angular sub-region b. The revised error metric over d sub-regions is defined as:

$$Err'(s_{0..3}) = max_{k=0}^{d} \left(1 - min_{i=0}^{3} \left(\frac{|\bigcap_{j=0}^{3} s_{jk}|}{|s_{ik}|} \right) \right) \quad (2)$$

In practice, a distance restriction is needed to account for cases where an arbitrarily small movement in the viewpoint results in a large change in visibility. Without this, infinite sub-division 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.

Another means of improving the error heuristic is to exploit the properties of a scene. For instance, many scenes consist of manifold objects with interiors that do not represent valid viewpoint positions. In this case, each of the four point samples can be classed as interior or exterior and Equation 2 is applied separately. Given the set of interior (Int) and exterior (Ext) point samples and $intThreshold$ and $extThreshold$, their respective error thresholds, we sub-divide *iff.* any of the following conditions hold:

- $|Ext| = 1$. Since a single exterior point does not provide sufficient information for a final decision.

[1] An angular sub-region is the direct analog of a (generally convex) region of the pixelated surface of the visibility cube. We use only uniform rectangular partitions

- $|Ext| > 1$ and $Err(Ext) > extThreshold$. This is the difference between samples considered only at *valid* camera positions.
- $|Int| > 1$ and $Error(Int) > intThreshold$. If the interior error is high, then there is a good chance that intermediate point samples will be external. For example, 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 visibility 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 backfacing polygons to be visible along edges of an object's exterior. To counter this, the polygon which 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 buffer.

5 Spatial Sub-division

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

Firstly, consider a scene bounding box, P_0 This can be partitioned by a single rectangle, R_0, orthogonal to an axis of P_0. Although the partition can be situated anywhere along an axis, we have chosen for illustrative purposes, to generate a balanced subdivision with equal volumes. The two partitions are labelled P_1 and P_2. Observe that all sightlines from P_1 to P_2 must intersect R_0. A polygon visible at the end of a sight line-segment is visible from all points along it. It follows that any polygon which intersects P_2 and is visible from a point in P_1, must be visible from R_0. If S denotes the set of scene polygons, then the visibility, $V(P_1)$, from cell P_1, may be calculated as:

$$V(P_1) = V(R_0) \cup \{x \in S : x \cap P_1\} \tag{3}$$

$V(R_0)$ represents the visibility from the rectangle R_0, and can be evaluated with the method of Section 3. The set $\{x \in S : x \cap P_1\}$ can be calculate with a simple polygon-parallelepiped intersection algorithm. Finally, $V(P_2)$ may be found in a similar fashion.

In general, the visibility set, $V(C)$, of a cell, C, is the union of the intersection of the scene with C and the visibility from the surface of C. The latter is pre-calculated by previous subdivisions, except for visibility from the partitioning rectangle.

A 3D grid of cells is generated by alternating the axis of subdivision. Deeper levels of recursion repeat the process on these sub-cells. This is effectively equivalent to building a *kd*-tree (for $k = 3$), where only the leaf nodes are maintained.

The grid of cells is generally non-uniform, since sub-division is terminated when the number of visible polygons falls below a set threshold, or *triangle budget* [11]. This threshold technique, adopted from Saona-Vásquez *et al.* [15], is a straightforward solution to enforcing upper bounds on rendering computations (and hence frame rates).

Infinite subdivision will occur if the number of polygons visible from any point is greater than the triangle budget. This is prevented by setting a *maximum depth* for the implicit binary hierarchy. In practice, this event if very unlikely because the polygon throughput for acceptable frame-rates is generally much greater than the number of polygons visible from any single point (or small neighbourhood around a point).

An advantage of top-down hierarchical subdivision is that smaller sub-cells, which do not contribute significantly to culling, are never evaluated (in contrast to the bottom-up sub-division of van der Panne and Stewart [18]).

6 Superset Simplification

This section covers an important optimization of the spatial subdivision process. In theory, the set of polygons visible from a cell, C, is a super-set (allowing for potential omissions due to aggressive sampling), of those visible from any viewpoint, q, within: $V(q) \subseteq V(C)\ \forall q \in C$. This allows the process of splitting a view cell to be optimized. When generating samples on the splitting rectangle of a cell, only polygons visible from that cell, $V(C)$, need be rendered. This implies that the cost of building a point sample decreases as its depth in the binary hierarchy increases. As shown in Appendix A, there appears to be an exponential decay, converging on a constant p (where p is the average number of polygons visible from viewpoints in the scene).

Although this optimization can be applied to any "from-region" technique, the rate of super-set decay is maximal for only exact and aggressive algorithms. As stated by Cohen-Or *et al.* [6], shadow volume based from-region culling depends on the relative size of cells and occluders. In the upper levels of a hierarchy, the cells are large. For the original technique of Cohen-or *et al.* [6], evaluating visibility from such large cells would prove pathological. They attempt to use a hierarchy, but find that only the deepest levels closest to the leaves are beneficial. Occluder fusion could reduce this problem. However, it is only the complete culling of an exact or aggressive solution which results in the evaluation of a minimal super-set.

7 Scalability

This section briefly covers algorithm complexity in terms of the quantity of polygons in the scene. Further details are provided in Appendix **??**.

Our technique partitions the scene until the visibility set is reduced below a certain size. Similarly, adaptive sampling proceeds until error drops below a user specified threshold. In both cases subdivision depends on scene distribution and is difficult to quantify. In order to obtain an average case estimate, we make several simplifying assumptions. Given d, the depth of the partitioning hierarchy, and k, the number of samples required (sufficient) for the first partition, we assume that:

- Both partitioning and sampling subdivide uniformly.
- A constant sampling rate is sufficient throughout the scene.
- The scene bounding box is a cube.

From these assumptions it follows that $O(kn)$ polygons are rendered into item buffers for the initial partition. We observe that every sample consists of six standard perspective viewpoint renderings each with a ninety degree field of view. Applying a good from-point visibility culling algorithm, such as hierarchical occlusion maps [20] or the hardware occlusion test implemented on most high-end graphics cards, reduces the complexity of rendering a visibility cube from $O(n)$ to $O(m)$. In this case, m is roughly proportional to the subset of polygons visible from the given sample point. An average complexity of $O(km)$ rather than $O(kn)$ is achieved, where m is typically much smaller than n and usually dictated by the *type* rather than size of the scene. For scenes with high depth complexity this decoupling is an obvious advantage.

A crucial, and novel, aspect of the algorithm is the caching and reuse of point samples. Run Length Encoding is used to compress this cache. Further, the space complexity of each cache element is $O(m)$ and is thus independent of scene size.

Point sample caching means that once $V(R_0)$ is determined, $V(R_1)$ can be extracted at no extra cost, so long as R_1 is a sub-rectangle within R_0. Although the total surface area of cells doubles at each step in the partition hierarchy, the surface area of unsampled cell walls, which do not overlap previously evaluated regions, is the same

as the level above. Thus, k samples are needed for each of d levels (since a constant sampling rate is assumed). Consequently, at most $O(kmd)$ polygons need be rendered.

Uniform partitioning, and hence a balanced hierarchy, is assumed so that, for c cells $d = \log c$, therefore $O(km \log c)$ polygons are rendered. This logarithmic dependence on c is a considerable improvement over the typically linear dependence.

The relation between k and n is difficult to establish precisely. This is discussed in Appendix **??**. The findings are that k ranges between $O(1)$ and $O(n^{\frac{2}{3}})$ depending on the polygon distribution of the scene. This leads to an overall average complexity ranging from $O(m \log c)$ to $O(mn^{\frac{2}{3}} \log c)$. To our knowledge, this is the only algorithm which is sub-linear in n and logarithmic in c.

The cost of reading from the frame buffer imposes a large constant complexity coefficient. However, in future we expect the excellent scalability of this algorithm and the current rapid improvement in frame-buffer access speed to enable pre-processing of the largest and most complex scenes.

8 Results

We report on the execution time, culling and error behaviour of our technique. Our evaluation is based on two very different scene types: a conventional building scene (moderate depth complexity), and a much larger forest scene (low to moderate depth complexity). Low and high polygon count versions of each scene type were employed.

The building (Figure 3d) represents a typical scene common to visibility research. As such it can be considered a performance benchmark. In contrast, the forest scene (Figure 3a) is a very difficult case, representative of a natural environment. The 2 million polygon forest scene is composed of 80 trees, each consisting of $25k$ polygons. To our knowledge, only Durand *et al.* [8] attempt to tackle the complexity of natural scenes with a forest comprising 7500 trees of $1k$ polygons each. Our forest scene is considerably more complex as a consequence of the highly refined tree models (Figure 3b).

Our tests were conducted on a 650MHz AMD Duron processor with 768mb RAM and a GeForce 2 GTS graphics accelerator. The results of our tests appear in Table 2. See Figure 3c and Figure 3e for a visualization the resultant culling.

The occlusion culling (excluding frustrum culling) is averaged over all cells, is considerable and the pre-processing times are good (note the relatively low-end platform).

The error heuristic of Equation 2 was used with $d = 24$. Each scene was evaluated over a walkthrough, with each polygon given a unique identifier. The error of an individual frame was found by counting the number of pixels that were different between our culled image and the reference (unculled) frame. The error was expressed as a percentage of the total number of pixels in a frame.

Scene	Size	Avg. Cull.	Err. Thresh	Avg. Err.	Max Err.	Pre-proc.
Forest2	2 M	91.46%	0.8	0.649%	2.837%	6h 2m
Building2	2 M	97.45%	0.5	0.725%	5.578%	6h 48m
Forest1	1 M	84.45%	0.75	0.15%	0.585%	1h 8m
Building1	1 M	91.32%	0.75	0.149%	1.52%	56m

Table 2. Results from pre-processing. The size in millions of polygons, average culling, error threshold used, average error (over a walkthrough), maximum error and pre-processing time are presented. Both scenes of 2 million polygons are sub-divided into 512 cells. The scenes of 1 million polygons are sub-divided into 64 cells.

Figure 4 shows the average error against a varying error threshold. The highest threshold 0.99 is roughly equivalent to forcing a sub-division only when the samples

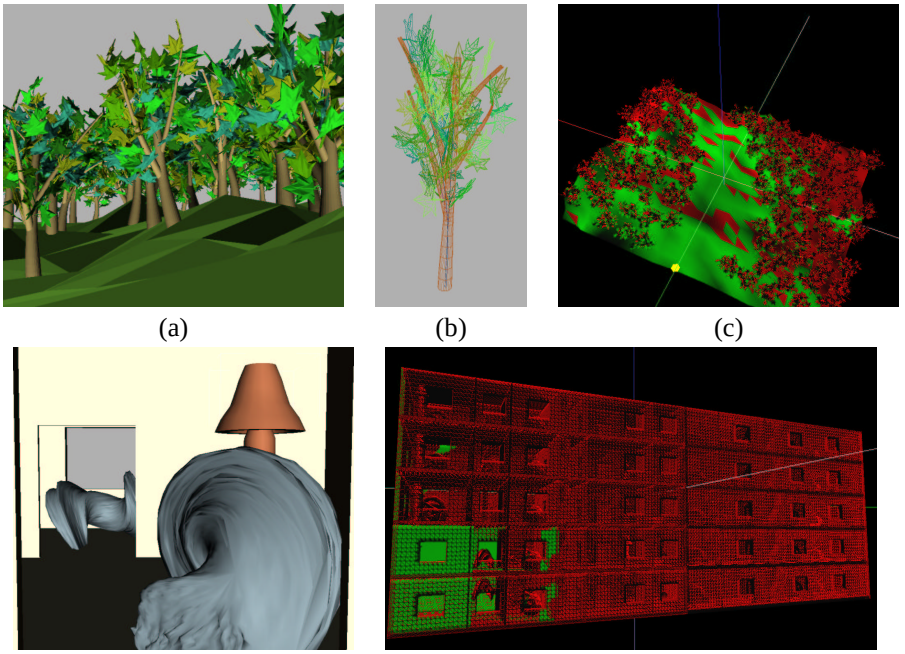


Fig. 3. [a] A view of the forest scene (2 million polygons). [b] A wireframe image of a single tree (25 thousand polygons). The complexity of the model is clear. [c] Visibility classification from the perspective of the yellow sphere. Polygons are classified as visible (green) or invisible (red). [d] Building scene (2 million polygons). The building is populated with highly detailed "art-works". [e] The camera is a room at the bottom left. The green polygons are visible from this viewpoint. The invisible polygons are drawn as red wireframes.

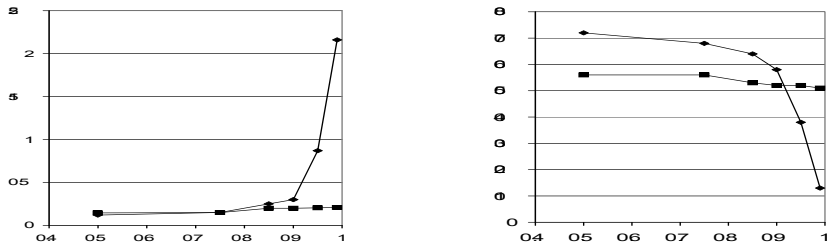
are completely different (disjoint visibility sets). The building scene responds well (avg. err. = 0.209%, thresh. = 0.99) because the heuristic effectively forces sampling in rooms which intersect a cell boundary. This property should extend to any scene comprised of many sub-regions with nearly or wholly disjoint visibility. In the forest model, error is very quickly dampened as the error threshold is reduced.

As the error threshold is reduced, more samples are generated, and processing time increases. This dependency is shown in Figure 4b.

A concern regarding aggressive and approximate visibility techniques is that flickering or *popping* artifacts may occur when cell boundaries are crossed. The heuristics of Equation 2 attempt to evenly distribute error across the image, thereby reducing perceptual artefacts. As a consequence, we experienced no popping during the walk-through of our forest scene. There was very little popping in our building walk-though (avg. err. = 0.725%), however we did experience the occasional artefact (max. err. = 5.578%). We attribute this to the very high frequency changes and reversions of the visibility state of polygons over the sampling domain in this model.

9 Conclusion

We have presented a new *from-region* visibility algorithm, which is classified as *aggressive* and therefore admits image error in the interests of optimal culling. This allows the



(a) (b)

Fig. 4. Diamond and squared marked series corresponds to the forest and building scenes respectively. [a] shows the error (in percent) plotted against the error threshold. [b] shows the time (in minutes) plotted against the error threshold.

treatment of previously infeasible scenes.

The algorithm is applied to a conventional architectural scene and a highly complex forest scene, which would be pathological to most other techniques. Both scenes (with 2×10^6 polygons) are handled successfully, with an average of 97.45% and 91.46% culling, respectively. We analyse the error and find that a readily acceptable average error of 0.725% and 0.649% (respectively) is introduced. This low error is attributable to our error heuristic which guide sub-division. All pre-processing is performed on a relatively inexpensive PC (Section 8).

Frame buffer access penalties impose a significant but constant cost, although hardware trends indicate that this is likely to decrease. The algorithm itself ranges between $O(m \log c)$ and $O(mn^{\frac{2}{3}} \log c)$, effectively a sub-linear dependence on scene size (n) and a logarithmic dependence on the number of cells (c). It follows that this algorithm is highly scalable and can pre-process visibility in even the largest scenes.

9.1 Future Work

We intend to continue improving our error heuristic so as to account for very high frequency fluctuations in visibility. We foresee two primary areas of performance enhancements in this research:

Optimizations. Any point based rendering optimization would further accelerate sampling. On-line visibility and frustum culling would enable efficient generation of upper levels in the cell hierarchy. However, in lower levels the superset simplification optimization is likely to dominate.

Recent graphics cards are capable of caching geometry in on-board memory to circumvent bus transfer delays. This is generally not possible with image-based visibility algorithms, since large models cannot fit into the (relatively) negligible video memory. However, superset simplification eventually results in sets which are small enough to fit into higher performance memory. This may offer a significant performance boost for the middle and lower levels of the hierarchy.

Parallelization. An attractive feature of our algorithm is the ease with which it can be distributed over multiple inexpensive machines. This is a consequence of the largely independent sampling process. In addition, this parallelization can be achieved on low cost machines. Memory access is minimal and most computation occurs on the graphics card, which only needs to support fast frame-buffer reading and high polygon count rendering for flat shaded (unlit) polygons(available on inexpensive game cards).

References

1. J. M. Airey, J. H. Rohlf, and J. Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *1990 Symposium on Interactive 3D Graphics*, 24(2):41–50, March 1990. ISBN 0-89791-351-5.
2. C. Andújar, C. Saona-Vázquez, I. Navazo, and P. Brunet. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum*, 19(3):499–506, 2000.
3. D. Bartz, M. Meißner, and T. Hüttner. Opengl-assisted occlusion culling for large polygonal models. *Computers & Graphics*, 23(5):667–679, October 1999. ISSN 0097-8493.
4. M. F. Cohen and D. P. Greenberg. The hemi-cube: A radiosity solution for complex environments. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):31–40, August 1985. Held in San Francisco, California.
5. D. Cohen-Or, Y. Chrysanthou, and C. Silva. A survey of visibility for walkthrough applications. submitted for publication, 2000. also in visibility, problems, techniques, and applications, 2000.
6. D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Eurographics*, 1998.
7. F. Durand. *3D Visibility, analysis and applications*. PhD thesis, U. Joseph Fourier, 1999.
8. F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000. ISBN 1-58113-208-5.
9. Z. Gigus and J. Malik. Computing the apsect graph for line drawings of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–112, 1990.
10. C. Gotsman, O. Sudarsky, and J. A. Fayman. Optimized occlusion culling using five-dimensional subdivision. *Computers & Graphics*, 23(5):645–654, October 1999. ISSN 0097-8493.
11. J. T. Klosowski and C. T. Silva. Rendering on a budget: A framework for time-critical rendering. *IEEE Visualization '99*, pages 115–122, October 1999. ISBN 0-7803-5897-X. Held in San Francisco, California.
12. V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 59–70, June 2000. ISBN 3-211-83535-0.
13. F.-A. Law and T.-S. Tan. Preprocessing occlusion for real-time selective refinement. *1999 ACM Symposium on Interactive 3D Graphics*, pages 47–54, April 1999. ISBN 1-58113-082-1.
14. H. Plantinga. An algorithm for finding the weakly visible faces from a polygon in 3d. (Extended Abstract).
15. C. Saona-Vásquez, I. Navazo, and P. Brunet. The visibility octree: a data structure for 3d navigation. *Computers & Graphics*, 23(5):635–643, October 1999. ISSN 0097-8493.
16. S. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley, 1992.
17. S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):61–69, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.
18. M. van de Panne and J. Stewart. Efficient compression techniques for precomputed visibility. *Eurographics Rendering Workshop 1999*, June 1999. Held in Granada, Spain.
19. P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 71–82, June 2000. ISBN 3-211-83535-0.
20. H. Zhang, D. Manocha, T. Hudson, and K. E. H. III. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97*, pages 77–88, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

A Rate of Decay by Superset Simplification

Consider a path P, from root to leaf of the *kd*-tree defined in Section 5. P is a set of d (depth) nodes in the *kd*-tree. We define P_n to be the element of P at depth n. The path contains: P_1, the root node, and terminates at P_d, the leaf node. Each node, P_n, is the set of visible polygons at depth n.

First let us assume: $\exists \alpha : \alpha = \frac{|P_{n+1}|}{|P_n|}, \forall\ 1 \leq n \leq d-1$. Since $P_{n+1} \subseteq P_n$ it follows that $0 \leq \alpha \leq 1$. $|P_n| = |P_1| \times \alpha^{n-1}$ converges to 0 at an exponential rate *iff* $\alpha < 1$.

The case where $\alpha = 1$ corresponds to a scene with 0 depth complexity. That is, all polygons are visible from all points in the scene. Scenes with low depth complexity are however unlikely candidates for a visibility algorithm.

Assuming a uniform scene distribution and splitting planes that divide cells in half, then $\alpha = 0.5 + \epsilon$. The 0.5 term is due to the containment of half the volume, and therefore half the polygons of the parent cell (by assumption of a uniform polygon distribution). The ϵ value corresponds to the number of polygons visible from, but not intersecting, the cell. For a high-depth complexity scene ϵ is generally much smaller than 0.5, resulting in exponential decay.

In practice however, there are many points with a similar visibility within a local neighbourhood. This is typical even of scenes with a high depth complexity. For example, in our architectural model, visibility is similar for most points in any one room. Clearly, the cells resulting from a sub-division occurring in this room would have visibility similar to their parent cell. In our tests we have found α to fall roughly between 0.55 and 0.65. However, there always exists an N such that $\alpha = \frac{|P_{n+1}|}{|P_n|}$ approaches 1 when $n > N$. This N depends on how quickly the cells converge to fall within a region of similar visibility. The existence of N can be easily proven, by observing that at least one polygon is visible from any point in a non-empty scene and it this which prevents cell visibility from converging to an empty set.

B Depedence of k on n

It is difficult to determine the exact dependence of k (initial number of samples) on n (size of scene). We make several simplifying assumptions to do so:

- Both partitioning and sampling subdivide uniformly.
- A constant sampling rate is sufficient throughout the scene.
- The scene bounding box is a cube.

Let us consider two different possibilities for scene growth, where the relative distribution of objects is maintained. Firstly, the scene may increase in detail, in that the surface area per unit volume remains constant, while polygon granularity grows. Secondly, the scene may expand spatially, growing in surface area but not polygon granularity.

For scenes which grow in detail, the relative structure of visibility events remains roughly constant. In particular, the occluder and occludee relationship of all objects is maintained, resulting in a similar distribution of visibility events. This implies that if k was sufficient, prior to growth, it remains sufficient ($O(1)$ growth).

For scenes which grow spatially, however, the situation is very different. In order to accommodate spatial growth and maintain a fixed distribution, the volume of the scene must increase. If we assume that volume is roughly proportional to the number of polygons in the scene, then for a cuboidal scene, this implies that that surface area

grows at a rate of $n^{\frac{2}{3}}$. Samples are taken on a surface and so k (the number of initial surface samples) is therefore of order $O(n^{\frac{2}{3}})$. It is worth mentioning, that in practice we have found k to be about almost three orders of magnitude smaller than n.