Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Exact From-Region Visibility Culling

S. Nirenstein, E. Blake and J. Gain[†]

Department of Computer Science, University of Cape Town,
Cape-Town, Rondebosch 7701, South Africa

Abstract

To pre-process a scene for the purpose of visibility culling during walkthroughs it is necessary to solve visibility from all the elements of a finite partition of viewpoint space. Many conservative and approximate solutions have been developed that solve for visibility rapidly. The idealised exact *solution for general 3D scenes has often been regarded as computationally intractable. Our exact algorithm for finding the visible polygons in a scene from a region is a computationally tractable pre-process that can handle scenes of the order of millions of polygons.*
The essence of our idea is to represent 3-D polygons and the stabbing lines connecting them in a 5-D Euclidean space derived from Plücker *space and then to perform geometric subtractions of occluded lines from the set of potential stabbing lines. We have built a* query architecture *around this query algorithm that allows for its practical application to large scenes.*
We have tested the algorithm on two different types of scene: despite a large constant computational overhead, it is highly scalable, with a time dependency close to linear in the output produced.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Exact Visibility Culling

1. Introduction

It is important, given the burgeoning complexity of rendered scenes, that geometry not visible from the viewpoint is identified and removed from the rendering pipeline as early as possible. Over the past two decades many *visibility-culling* techniques have been developed to perform this task. These solutions can be loosely categorised into those applied either at run-time or during a pre-process. With a few exceptions, the former determine visibility from a single point, while the latter establish the subset of geometry visible from any point within a region.

From-region visibility partitions the view-point space (VPS) into regions or cells, rather than attempting to preprocess the infinite number of possible camera positions. The principal advantage of these techniques is that their considerable computation cost can be shifted to a pre-process, consequently removing significant run-time visibility computation. The results may then be saved to disk for future use. The disadvantage is that, as a pre-process, only static scenes can be fully treated.

From-point visibility algorithms are less costly computationally than from-region approaches, allowing them to be applied on a per-frame basis at run-time. They are also better suited to unpredictably dynamic environments, where the shape or position of objects may change between frames.

Visibility culling algorithms may be further categorised according to their accuracy in differentiating between visible and invisible polygons. We extend the taxonomy of Cohen-Or *et al.*[8] and we discriminate between *conservative*, *aggressive*, *approximate* and *exact* visibility algorithms (Table 1).

	Run-Time Performance	
Image Quality	Optimal	Sub-optimal
Correct	*Exact*	*Conservative*
Errors	*Aggressive*	*Approximate*

Table 1: *Properties of from-region visibility solutions.*

† {snirenst,edwin,jgain}@cs.uct.ac.za

Conservative techniques consistently overestimate visibility and incur a false visibility error: invisible polygons are considered visible. This results in sub-optimal run-time performance because these polygons are unnecessarily submitted to the rendering pipeline.

In contrast, *aggressive* methods always underestimate the set of visible geometry and exhibit false invisibility, where visible polygons are erroneously excluded. Aggressive visibility causes image error but can be useful in practice if: (a) the perceptual impact of the error is acceptably small, (b) the algorithm is computationally efficient or (c) it handles scenes that cannot be solved effectively with a conservative alternative due to excessive overestimation.

Approximate visibility techniques give both false visibility and false invisibility errors and are thus most useful when efficiency is the overriding concern.

Exact visibility solutions provide both accurate images and optimal rendering performance. An exact visibility query will produce no more or less than the union of the polygons visible from all viewpoints within the region. Of course, this is somewhat conservative with respect to a single viewpoint, incorporating as it does visibility from the region as a whole.

Exact visibility solutions have several advantages beyond the obvious rendering benefits:

- The magnitude of visibility error (both *over* and *under* estimation) in conservative, aggressive and approximate schemes is highly dependent on the type of scene. This is not true of exact visibility. By definition, it is a unifying treatment that deals with *all* scenes in the same way.
- Exact visibility culling provides a benchmark for comparing other approaches. The typical way of assessing the quality of a visibility technique, namely the percentage of polygons that it culls in a given scene, is a very crude and imperfect metric. Such measures do not account for the type (false visibility or false invisibility), distribution (clustered or widespread) or exact magnitude, of visibility error.

In this paper we present what is, to our knowledge, the first exact from-region visibility-culling algorithm tractable for realistically sized scenes (as many as 1.5 million polygons). This represents a three order of magnitude improvement over the previous state of the art[12, 30], albeit these previous approaches were targeted at a slightly different application.

Our approach is not a panacea. It offers an optimal rendering time, by only submitting the necessary polygons in a region to the rendering pipeline. It also yields correct images, by not falsely excluding visible polygons. Although tractable for large scenes, the pre-processing costs are still considerable. In circumstances where fast pre-processing is more critical than image quality or run-time rendering speed, a conservative, aggressive or approximate technique may be more appropriate.

Our contribution depends on two novel techniques:

- *Localised Exact Visibility.* A novel algorithm which accurately queries the visibility between two convex polygons. A principal efficiency advantage of this algorithm is that it can be applied on-demand to a localised swathe of scene geometry between the two polygons. The nature of the computation is such that it is considerably more suited to from-region computation, than any obvious adaptation of existing exact global visibility techniques[11, 12].
- *Query Driven Architecture.* In terms of computational load, it is not sufficient to naively query each polygon in the scene. A combination of conservative and aggressive queries are used to cull groups of polygons and to trivially accept others. Results from previous computations are exploited in order to further accelerate the query process. This architecture also has the benefit of output-sensitivity.

The rest of this paper is as follows: After a review of previous work and a short background on the relevant mathematics and terminology, we discuss our exact visibility algorithm. This is followed by details of the localised exact visibility queries. We then discuss the architecture which allows us to handle large scenes efficiently. Finally, we present the results of a series of experiments used to quantify the overall performance of our exact visibility technique.

2. Previous Work

This paper focuses on both from-region and exact visibility. From-point visibility methods are not relevant and we refer the interested reader to the excellent surveys by Cohen-Or *et al.*[8], Durand[10] and Zhang[33].

Cell-portal rendering[1, 28, 31] attempts to establish the relative visibility of entire cells. The visibility of one cell from another depends on the existence of a line of sight between them, intersected only by portals – the non-opaque boundaries between cells. Teller[28, 31] derived an analytic solution to this problem. Teller[30] further extended the cell-portal technique in order to determine exactly which parts of visible cells are indeed visible. These algorithms are suited to architectural scenes, but aren't a general visibility solution. They work well for scenes where the combinatorial complexity of portal sequences is low.

Cohen-Or *et al.*[9] and Saona-Vásquez *et al.*[24] provide a more general, but more conservative solution. They only classify an object as invisible from a particular view-cell if a single polygon occludes it from every point inside that view-cell. This is so conservative that it only achieves significant results if view-cells are small, relative to the size of scene polygons. For highly detailed scenes, this often requires a prohibitive number of view-cells and hence excessive run-times.

To counter this, it is necessary to consider the aggregated occlusion by a set of smaller occluders. 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.*[10, 13] have recently developed a general solution to visibility culling using an extended projection operator.

Law and Tan[20] use *occlusion preserving simplification* to generate a lower level of detail and thus larger polygon occluders while ensuring conservativity. The occlusion achieved by the fusion of these coarser representations is not considered.

Koltun *et al.*[18] make use of separating lines to build larger more effective virtual occluders to represent 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. They only implement a $2\frac{1}{2}$D (height field) solution. Schaufler *et al.*[25] present a similar conservative approach in 3D, where virtual occluders are generated from a volumetric discretisation of the interior of scene objects.

Wonka *et al.*[32] have a conservative $2\frac{1}{2}$D solution. They *shrink* a subset of occluders and then sample visibility (effectively fusing occluders). Occluder shrinking allows the sampling process to maintain conservativity. There is a trade-off between samples required (and hence time) and the degree of shrinkage. This algorithm tends towards an exact solution as the number of samples, and hence the amount of time required, tends towards infinity.

The techniques surveyed above are all conservative in nature. They provide accurate images, but for many scenes there is a large margin between the size of the visibility sets they generate and those of an exact visibility solution.

Recently, *aggressive* from-point based solutions have been developed that admit false invisibility errors using approximate culling[34, 5]. These techniques aim to reduce rendering costs by removing objects that contribute little to the image. Andújar *et al.*[2] use *hardly visible* sets to cull or simplify scene objects where only a small proportion of their geometry is visible. Klosowski and Silva[17] present a *prioritised layer projection* algorithm that uses a heuristic priority ordering that tries to draw visible polygons first. This is a good time-critical solution, since one gets reasonable results even if rendering is prematurely terminated.

Gotsman *et al.*[16] 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 hit the target object, is then 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.

Turning to exact visibility, various solutions have been developed for answering general visibility queries. These techniques typically build a structure in some form of dual line space that directly exposes *visibility events*. A visibility event occurs when a topological change in visibility occurs in the scene (eg. the set of view-points from which a particular vertex of a previously occluded triangle may cross an edge of another and become visible). Structures representing this for general scenes are the *aspect graph*[15](changes in aspect are also encoded) and the *3D visibility complex*[11], an extension of the 2D visibility complex[22] and the *visibility map* [23]. Earlier, Teller[30] showed how the construction of the anti-penumbra through a set of portals may be computed via a similar structure, using the *Plücker* parameterisation of line space. The usefulness of these algorithms is severely limited by combinatorial complexity and robustness issues.

Durand *et al.*[12] present a construction of visibility event surfaces through the direct computation of the lower dimensional elements (skeleton) of the visibility complex. The nature of this construction is that errors resulting from degeneracies are localised, making the construction of the visibility skeleton more robust than the construction of the full visibility complex.

Koltun *et al.*[19] build a representation of the rays between segments in a dual ray-space. Occlusion is computed by determining whether the space of occluded rays contains all rays between the view-cell and the object in question. This *2D* exact visibility solution is approximated efficiently via discretisation through rendering hardware. Independently, Bittner *et al.*[6] also developed an exact visibility solution in the plane, based on similar principles.

The query algorithm presented in this paper is effectively an efficient 3D approach to the *on-demand* construction of part of a visibility complex-like structure. Unlike Durand, we follow Teller's convention and use the Plücker coordinate line space parameterisation. Further clarification (in Section 3.4) of the relationship with previous work must be deferred until after our algorithm is presented.

2.1. Plücker Line Space

In order to describe our exact visibility algorithm, we first give a brief overview of the fundamental underlying theory of our algorithm, *Plücker line space*. Teller[28] gives a discussion of this in a similar context.

Plücker space is a special case of a Grassmann coordinate system[26, 27, 7]. Grassmann coordinates allow for the parameterisation of a k-dimensional affine sub-space embedded in an n-dimensional space as a point in a projective $\binom{n+1}{k+1} - 1$

dimensional space. Plücker space in particular corresponds to lines ($k = 1$) in $\mathbb{R}^3$ ($n = 3$). This results in a projective five dimensional space $\mathbb{P}^5$. This parameterisation exposes a natural and elegant means of dealing with directed lines in $\mathbb{R}^3$. The Plücker mapping of a directed line ℓ passing through the point (p_x, p_y, p_z) and then through (q_x, q_y, q_z), is defined as $\Pi(\ell) = (\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$, where:

$$\begin{array}{ll} \pi_0 = Q_x - P_x & \pi_1 = Q_y - P_y \\ \pi_2 = Q_z - P_z & \pi_3 = Q_z P_y - Q_y P_z \\ \pi_4 = Q_x P_z - Q_z P_x & \pi_5 = Q_y P_x - Q_x P_x \end{array}$$

This homogeneous six-tuple of $\mathbb{P}^5$ is a unique representation of ℓ to within multiplication by a positive scale factor. Negating the scale factor flips the orientation of the line.

We next consider a duality mapping within $\mathbb{P}^5$. Given $\pi, x \in \mathbb{P}^5$, we define $D_\pi(x) : \mathbb{P}^5 \to \mathbb{R}$ to be the quantity $\pi_0 x_3 + \pi_1 x_4 + \pi_2 x_5 + \pi_3 x_0 + \pi_4 x_1 + \pi_5 x_2$. This is a permuted inner product of π and x. The set of solutions $x \in \mathbb{P}^5$ of $D_\pi(x) = 0$ gives rise to the so-called *dual hyperplane of* π *in* $\mathbb{P}^5$.

Given lines ℓ_1 and ℓ_2, let $\pi^1 = \Pi(\ell_1)$ and $\pi^2 = \Pi(\ell_2)$; ℓ_1 and ℓ_2 are incident if and only if π^1 lies on the dual hyperplane of π^2 (and vice versa). Formally, they are incident if and only if $D_{\pi^1}(\pi^2) = 0$. If $D_{\pi^1}(\pi^2)$ is not equal to zero, then the relative orientation of ℓ_1 and ℓ_2 is directly specified by the sign of $D_{\pi^1}(\pi^2)$. See Figure 1 for an illustration.

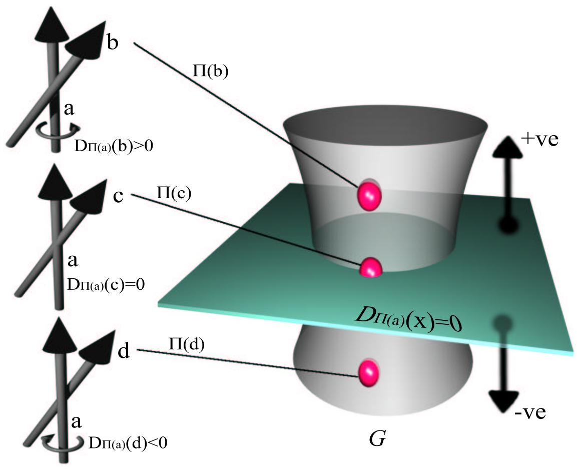


Figure 1: Line orientation and Plücker space. *The three diagrams on the left shows the three qualitatively different ways for one directed line to pass another. The figure on the right is a visualisation of how these lines relate in the dual Plücker space. The surface G is a visualisation of the Plücker hypersurface embedded in* $\mathbb{P}^5$. *Lines b and d pass by line a on the right and left respectively. Line c is incident on a. If lines b, c and d are mapped to* $\mathbb{P}^5$ *via* Π *(visualised as the three dots), then respectively they will lie above, on and below the plane defined by* $D_{\Pi(a)}(x) = 0$.

Although all lines in $\mathbb{R}^3$ map to points in $\mathbb{P}^5$, not all points in $\mathbb{P}^5$ map to lines in $\mathbb{R}^3$. Rather, Π is a bijection between the lines in $\mathbb{R}^3$ and a particular four-dimensional quadric surface embedded in $\mathbb{P}^5$ known as the *Grassmann manifold*, the *Klein quadric* or the *Plücker hypersurface*. This surface is described by the following set of points:

$$G = \{D_x(x) = 0 : x \in \mathbb{P}^5\} \setminus \{\mathbf{0}\} \qquad (1)$$

Since, at least for the purposes of this paper, we are only interested in real lines, this surface is used consistently.

3. Visibility Query

3.1. Overview

In order to compute exact visibility, we begin by constructing a representation of the space of lines *stabbing* (incident on both of) a pair of polygons. This is equivalent to the construction performed by Teller and Hohmeyer[29]. Ideally, we would like to deal with the line *segments* between these polygons. It is, however, sufficient to simply clip all polygon geometry to the interior of the convex hull defined by the vertices of the polygon pair, and then to deal only with infinite stabbing *lines*.

Every occluder polygon within this convex hull “blocks” a set of lines between the pair of query polygons. If every line between the query polygons is blocked by some occluder, then the polygons are mutually invisible.

The analogue presented in this section, is that the space of lines between two polygons is represented as a connected subset of points on the Plücker hypersurface, G (see Section 2.1). Similarly, each occluder is represented by a set of points on G, which corresponds directly to the set of lines it blocks.

The algorithm presented here is one which incrementally removes, for each occluder, the set of occluder points from that of the currently un-obstructed volume. The initial un-obstructed volume is the space of lines between the two query polygons. This removal is achieved by using constructive solid geometry(CSG) in five dimensions. After all occluder points have been removed, the points in the remaining volume are the set of lines that are not obstructed by occluders. If no such points exist, the objects are mutually invisible. If at least one such point does exist, then the query polygons are mutually visible, and a full description of the space of un-obstructed lines between them has been computed.

3.2. The Space of Lines Between Two Polygons

Ideally, we would like a generic representation of the space of lines between two polygons that is both easy to represent and to manipulate. Working directly on the surface of the Plücker hypersurface is difficult due to its curvature. Instead, we build a polyhedral representation of this line space, whose intersection with the Plücker hypersurface gives the desired set of points. This polyhedron may be constructed as follows:

For a line s to stab two given polygons, it is necessary and sufficient for all edges passed by a directed version of s to have the same particular relative orientation. This is illustrated in Figure 2. Given the (appropriately directed) set of edges of these two polygons $e_1, e_2, \ldots, e_n$, verifying whether s is a stabber is equivalent to testing whether:

$$D_\pi(\Pi(s)) \geq 0, \forall \pi \in \{\Pi(e_1), \Pi(e_2), \ldots, \Pi(e_n)\} \quad (2)$$

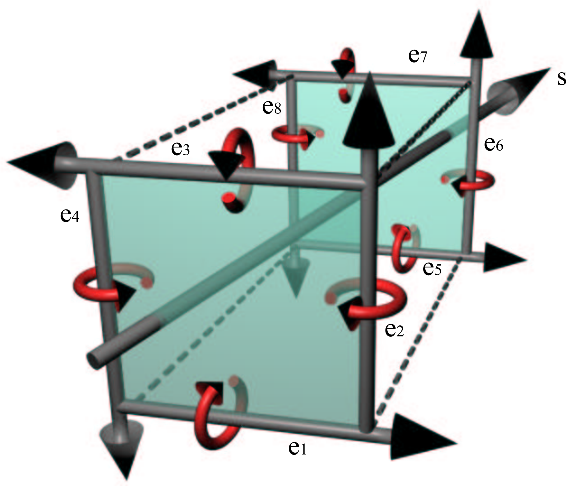


Figure 2: Stabbing constraints. *Directed lines e_1 to e_8 correspond to the edges of two quadrilaterals. Note that the stabbing line s passes all e_1 to e_8 to the left. This condition is both necessary and sufficient for stabbing.*

The projective space $\mathbb{P}^5$ is simply $\mathbb{R}^5$ with two additional hyperplanes at positive and negative infinity. These additional hyperplanes are useful as an elegant means of uniformly handling cases which appear as singularities in other parameterisations. Recall that the Plücker six-tuples are unique to within a positive scale factor. It is therefore possible to normalise the projective six-tuple by dividing through by one of its components, and then excluding that component (normalised to one). This of course assumes that the normalisation component is non-zero. It is trivial to pre-rotate the scene geometry so as to prevent this possibility from occurring. This process is equivalent to projecting down to $\mathbb{R}^5$. This approach was also taken by Teller and Hohmeyer in[29] and is necessary (although not sufficient) to represent the structure as a *bounded* polytope. Our motivation for boundedness is discussed in Section 3.2.1.

After this projection, the function $D_\pi(x) : \mathbb{P}^5 \to \mathbb{R}$ becomes $D'_\pi(x) : \mathbb{R}^5 \to \mathbb{R}$ where $D'_\pi(x) = \pi_0 + \pi_1 x_4 + \pi_2 x_5 + \pi_3 x_0 + \pi_4 x_1 + \pi_5 x_2$. In this case, the third element has been selected as the normalisation component, and thus $x_3 = 1$ and may be omitted. The six tuple which was $x = (x_0, x_1, x_2, x_3, x_4, x_5)$ becomes the tuple $x = (x_0, x_1, x_2, x_4, x_5)$ in $\mathbb{R}^5$. Where the old form of the equation $D_\pi(x) = 0$ ($x \in \mathbb{R}^6$) defined a plane through the origin in $\mathbb{R}^6$, $D'_\pi(x) = 0$ ($x \in \mathbb{R}^5$) defines a less restricted plane in $\mathbb{R}^5$. Using D' rather than D, the solution space for the constraints of a stabbing line s given in Equation 2 may be transformed to define a volume in $\mathbb{R}^5$. This volume completely represents the space of lines between the two query polygons. This construction and mapping is depicted in Figure 3a and Figure 3c.

There are various techniques which may be used to extract a polyhedral representation from several half-space intersections. We used a version of the double description method[3] to extract the extreme points. With this and the hyperplane information, it is possible to construct the full face lattice/graph of the polyhedron using a combinatorial face enumeration algorithm such as that presented in[14]. The full face lattice is a requirement of the CSG algorithm presented in Section 3.2.1.

We note that the polyhedron defined by the intersection of these spaces is, in general, unbounded. We observe that although the polyhedron is unbounded, the intersection with the polyhedron and the Plücker hypersurface cannot be. This, the space of *real* lines between the two polygons, is indeed bounded. It is trivial to cap the polyhedron by adding additional constraints which will not alter the intersection of the polyhedron with the hypersurface.

3.2.1. CSG in Plücker Space

To remove an occluder from the polyhedron, we have to find a representation for it. In the same fashion as for the initial polytope, we map the edges of the occluder to hyperplanes in $\mathbb{R}^5$ using the Π and D' operators. The volume enclosed by the intersection of the half-spaces described by these hyperplanes is used. We maintain the volume as a set of oriented hyperplanes, $\mathcal{O}$. This volume is unbounded. We depict the mapping of an occluder to $\mathbb{R}^5$ in Figure 3b, and the subtraction of this volume from the initial polytope in Figure 3d.

Subtraction of one polyhedron from a polytope in five dimensions is a non-trivial task. Given an algorithm that splits any polytope into two, each half falling on either side of a specified hyperplane, it is possible to partition a polytope into a set, or *complex*, of polytopes that have at most one face incident on any hyperplane of $\mathcal{O}$. The polytopes of the complex that fall within the volume enclosed by $\mathcal{O}$ can then easily be identified and removed. An intersection of the remaining complex with the Plücker hypersurface provides the set of lines unblocked by the occluder $\mathcal{O}$.

This approach is applied iteratively, until either there are no polytopes left in the complex, or there are no occluder volumes left to subtract. The former implies that the query polygons are mutually occluded, the latter implies that they are mutually visible if and only if at least one of the poly-

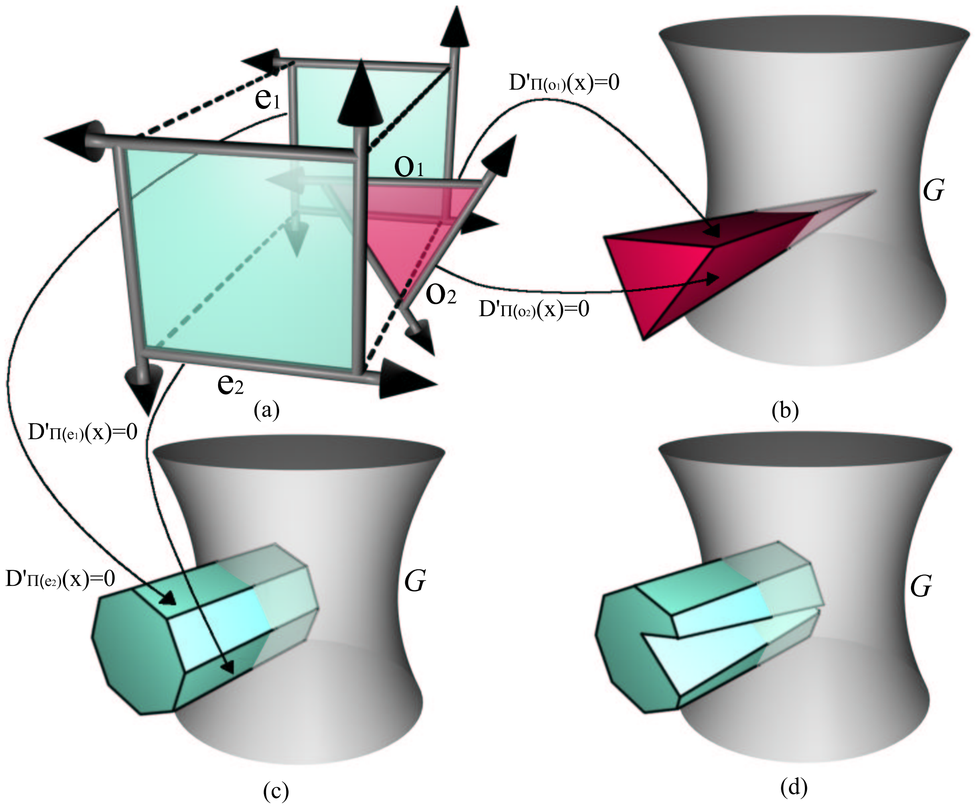


Figure 3: Plücker-complex construction. *(a) A typical query setup between two polygons (the quadrilaterals), and one occluder (the triangle). (b) A visualisation of the mapping of the edges of the triangle occluder to a volume in* $\mathbb{R}^5$*. Again, G is a visualisation of the Plücker hypersurface. (c) A visualisation of the mapping of each edge of the quadrilaterals to form a volume in* $\mathbb{R}^5$*. (d) The subtraction of the occluder volume from the initial volume. This fully represents the space of lines between the quadrilaterals that* miss *the occluding triangle.*

topes, within the complex, intersects the Plücker hypersurface.

We use an algorithm similar to the multi-dimensional polytope splitting algorithm of Bajaj and Pascucci[4]. The algorithm requires the highest dimensional elements to be bounded polytopes and the existence of the full face lattice. The original algorithm maintains the whole polytope-complex as a single face graph structure. Each split operation requires the traversal of every d-dimensional face (for $0 < d \leq 5$). Instead, after each split, we construct two *separate* polytopes. This introduces redundancy into the representation, since shared faces are duplicated, but we gain the performance advantage of quickly being able to isolate which polytopes are split, reducing the set of polytopes traversed to that of the set of polyhedra incident on the hyperplane. This is also known as the *zone* of the hyperplane. To query whether a polytope intersects a hyperplane, we use 5D bounding spheres for a conservative test. If this test shows no intersection, then the polytope is trivially classified as non-intersecting, otherwise the result is indicative of a potential intersection. In the latter case, an accurate vertex-hyperplane sidedness test is then used to determine whether any two vertices lie on opposite sides (or on) the hyperplane. If and only if this proves true, will the polytope intersect the hyperplane.

3.3. Optimisation Strategies

The query algorithm is the core of our visibility technique. We have therefore spent considerable effort in developing optimisation strategies. Firstly, when splitting the polytope-complex by a set of occluder hyperplanes $\mathcal{O}$, we split *only* those polytopes that cross the *boundary* of the polyhedron described by $\mathcal{O}$. After each split operation, if one of the newly created polytopes does not intersect the Plücker hypersurface, it is removed from the complex. This allows the complex to exist only in the zone of the Plücker hypersurface.

Our second strategy is aimed at trivial acceptance. By casting rays from one polygon to another it is possible to accept a visible polygon as being trivially visible. If a ray originating at one polygon reaches another (without intersecting any occluders), the polygon is accepted as visible. If no such ray exists, then we resort to the exact query algorithm.

If a large number of rays is cast without proving visibility, it is very likely that the object is invisible. In scenes where occlusion culling would be of most use, a majority of the scene geometry is invisible. Furthermore, by casting sufficient rays, a vast majority of the scene that is visible can be trivially accepted. We refer the interested reader to [16] to view a strategy for quantifying this likelihood. It is therefore of enormous benefit that the visibility query algorithm presented here manages to terminate early when complete occlusion is found.

Our subtractive 5D CSG is more efficient if it can remove as many potential stabbing lines as quickly as possible. So, our third strategy is to accelerate early termination by analysing the previously cast rays in an attempt to isolate the smallest set of polygons that are sufficient to declare the likely result of invisibility. Our approach is iterative: The occluder with the largest number of incident rays (i.e. those cast in the previous stage) is subtracted from the line-space volume first. The occluder with the next largest number of incident rays, *excluding* those rays accounted for by any previously subtracted occluders, is subtracted next. This continues until all rays have been accounted for. Any remaining occluders are then subtracted. At any point in this iteration, if there is no longer a complex from which to subtract, invisibility is established and the query may terminate.

3.4. Discussion of our Contribution

Both the visibility complex and skeleton have been designed as tools for performing exact visibility queries rapidly between *scene objects*, as opposed to queries between *cells* and *objects*. Furthermore, these approaches attempt to extract not just the qualitative states of visible or invisible, but also an exact description of which parts of each objects are visible. Their goal is to use visibility primarily for accurate illumination simulations, whereas ours is from-region qualitative per-polygon visibility.

Our visibility query algorithm is essentially a direct construction of a localised subset of the visibility complex. It is more suited to queries, since the construction is local to the pair of polygons in question, and is sensitive only to the smallest number of occluders necessary for occlusion, as opposed to sensitivity in the number of visibility interrelationships between all occluders.

Durand *et al.*[12] present an on-demand construction of the visibility skeleton. The construction of the skeleton involves an explicit combinatorial iteration through the various elements of the scene geometry, in order to construct the lower dimensional elements of the complex. As for the visibility complex, this computes more than is required for our purposes, namely the visibility relationships between all the occluders. The skeleton construction algorithm can also be adapted to terminate early when an object is found to be visible. As we have shown, this is generally unlikely for the scenes in which we are most interested. The lack of topological information in the skeleton (due to the ommission of the higher dimensional elements) makes it impossible to gauge whether an object is invisible during construction. Invisibility may only be ascertained when the whole skeleton (relative to the query polygons) has been constructed.

As for the visibility-complex, the Plücker hyper-plane arrangement[21] is a superstructure of the structure computed by our query algorithm. The respective combinatorial complexities of this hyper-plane arrangement and the visibility-complex are $O(n^4 \log n)$ and $O(n^4)$. Two lines are defined to be in *qualitatively distinct* regions if they are both passed by the directed lines extended from the scene polygon edges in the same way. The Plücker hyper-plane arrangement encodes *all* qualitatively distinct regions of line space (the visibility-complex accounts for line segments). A necessary implication of this, is that if two lines fall in the same region, then they must stab the same polygons.

To put our algorithm into context, we *only* compute the subset of qualitatively distinct regions relating to lines intersecting our pair of query polygons. This is explicit in our initial polytope construction. For our purposes it is also irrelevant *how* a region of line space is blocked (i.e. by which occluders). We are only interested in *if* it is blocked. By subtracting blocked regions, rather than simply enumerating them, we avoid the computation and internal representation of further refinements.

The first advantage of our algorithm, is that it terminates early in the most common case. Secondly, for all but the most contrived of scenes, the combinatorial complexity of the information in which we are interested, is very much smaller than that of global visibility techniques. Our algorithm exploits this fact, and is sensitive to this lower complexity. Experimentally, Durand noted that the average time complexity for the construction of the visibility skeleton appears to be $O(n^{2.4})$. Since we compute significantly less information, we expect the average case time complexity of the query algorithm to be significantly lower. We demonstrate this experimentally (see Section 5).

4. Query Architecture

The objective of our architecture is to group polygons together effectively, so that clusters of polygons may be classified using the fewest possible queries. We also wish to take advantage of previously computed results.

Our approach is most similar to that of Koltun *et al.*[19]. Once again, this solution is for the much simpler $2\frac{1}{2}$D problem. We use discrete sampling as a heuristic for determining an efficient order of subtraction. Our architecture is also somewhat different in that it takes advantage of previously computed results. We believe this last technique could further enhance the algorithm of Koltun *et al.*

4.1. Cluster Queries

There are many ways to order a scene hierarchically. We take advantage of natural scene coherence and use a simple two level hierarchy, where the scene consists of a set of objects, which in turn consist of individual polygons. If objects are very large (based on a volume and polygon count threshold), we split them into separate objects.

From a *source* view cell (an element of a partitioned 3-dimensional camera space), we first query the bounding box of an object. If the bounding box is invisible, then clearly, all its contained geometry is also invisible. If, however, the box is determined to be visible, then the geometry contained may be considered *potentially* visible from the source cell. The true visibility status of each child may then be queried individually (from each side of the source cell).

The source cell to bounding box query is effectively the combination of the queries between all pairs of faces of the two boxes. There are 36 possible pairings; however, back-face removal will quickly reject most of these as mutually invisible. If all side to side queries return “invisible”, then the object bounding box is invisible. If one such pairing returns “visible”, then the object bounding box is considered visible. It is sufficient to terminate the pair querying early if visibility between any one pair is shown. However, in the next section we will see the advantage of completing this query.

4.2. Parent Line-space Reuse

If the bounding box of a target object is visible, and we allow the completion of all side to side query pairs, we effectively have a representation of *all* un-obstructed line segments originating from the source cell and terminating on the target box. This may be visualised as a similar structure to Teller’s *anti-penumbra*[30]. By extending these line segments (anti-penumbra) through the bounding box, we have the set of lines for which it is *necessary* for any visible object within the bounding box to intersect.

This allows a fast, conservative, but relatively accurate rejection test to be applied: Each polygon within the bounding box is transformed to its hyper-plane representation in the Plücker coordinate system, and if each polytope of the previously computed complex (for the bounding box) does not intersect the polyhedron defined by this transformation, then the polygon is necessarily invisible. This can be efficiently, but conservatively, computed by testing to see whether every polytope complex falls exterior to at least one half-space defined by the transformed polygon. Any polygons that pass this test will have to be queried individually.

4.3. Virtual Occluders

During the process of computing the visibility status of bounding boxes, opportunities arise where it is possible to extract “virtual occluders”. As coined by Koltun *et al.*[18], virtual occluders are occluders that are not part of the geometry, but still represent a set of blocked lines. We observe that if the side of a bounding box is determined to be invisible from a source view cell, it may then be used as an occluder for any object behind it. In fact, if the whole bounding box of an object is invisible, then none of the polygon geometry within need be incorporated as occluders, since the bounding box is sufficient. In truth, the bounding box is *more* than sufficient, since a bounding box occludes at least as large a volume of line-space, as the geometry it contains.

By processing the scene objects in an approximate front to back order, it is possible to fully exploit this feature. This is key to our algorithm’s *output sensitive* nature. Geometry behind the nearest occluded “layer” (with a similar connotation to that of Klosowski and Silva[17]), is quickly rejected since the relatively large size of the virtual occluders imply that only very few occluder are necessary to confirm invisibility. Output sensitivity persists as long as the query algorithm can use virtual occluders for trivial rejection (as ours does).

5. Results and Discussion

We have implemented the exact visibility query algorithm of Section 3 and integrated it into the architecture proposed in Section 4. We present the results obtained from testing our algorithm on two different representative scenes.

Firstly, we tested the algorithm on a *town* scene depicting the geometry of a small 16th century town. The scene consists of 1.33 million triangles and includes several highly detailed component objects which are visible through doors and windows. This scene is fully 3-dimensional in nature. Our second scene is the *forest* scene used by Durand[10]. This scene consists of 1.45 million triangles, organised into 1450 trees each with 1000 triangles. This represents a much more difficult scene to cull. In terms of the algorithm presented here, it is an extreme case, since occlusion only occurs as the aggregate of a large number of triangles and consequently subtractions on the Plücker dual polytope. See Figure 5 for screen-shots of these scenes showing the output of our implementation.

For the town scene we choose a uniform subdivision of cells. The base grid is a 32x32 partition of the base of the bounding box, and we consider two such levels, for a total of 2048 cells. This selection of cells should cover the camera view-space used in any reasonable walkthrough application. 16384 (32x32x16) such cells would fully partition the scene bounding box. For the forest scene, we apply a 20x20 partition for compatibility with Durand.

We executed the tests on a dual Pentium 4 1.7Ghz, and solved for distinct cells in parallel. For both scenes we have culled and timed the visibility of a random selection of 100 cells. For the town and forest scene we considered a full solution to be one consisting of 2048 and 400 cells respectively. These timings, are presented in Table 2.

Scene	Time/Cell	Culling	Full solution
Town	2min 33sec	99.45%	3 days 15hrs
Forest	10min 30sec	99.12%	2 days 22hrs

Table 2: Experimental result summary.

To solve for the whole town scene, would require 3 days and 15 hours on our PC. Similarly, the forest scene would take 2 days and 22 hours. Such requirements are high by the standards of most existing approximate or conservative algorithms, but the preprocess is a once only cost, and is more than offset by the advantages of our approach. Note the large degree of culling achieved (Table 2).

When only a single workstation is available for preprocessing, this algorithm can be used as a final and permanent visibility solution applied after a large model has been fully generated; as an accurate solution for smaller models; or as the only possible acceptable solution for difficult models.

We advocate the use of this algorithm on machine clusters, which are often readily available. It is easy to parallelise our technique, since each cell can be solved independently. The computation is very loosely coupled, making it suitable even for informal clusters with a low bandwidth infrastructure.

The scalability of visibility pre-processing algorithms is of crucial importance. The run-time of our algorithm is dependent on the particular scene in terms of the number of visibility queries performed, and the time taken to solve these queries.

The visibility query algorithm quickly rejects occluders which fall outside of the space of lines between the query polygons. During line space subtraction, it also quickly rejects occluders which have already been accounted for. Hence there is little correlation between the number of polygons in the scene and the time complexity of the query algorithm. In order to quantify time complexity, we have developed an alternative measure, namely the *number of effective occluders* subtracted.

An *effective occluder* is an element of the set of occluders generated by the ray-casting performed in Section 3.3. This set approximates (one of) the smallest possible sets of occluders required to block the total occluded line space.

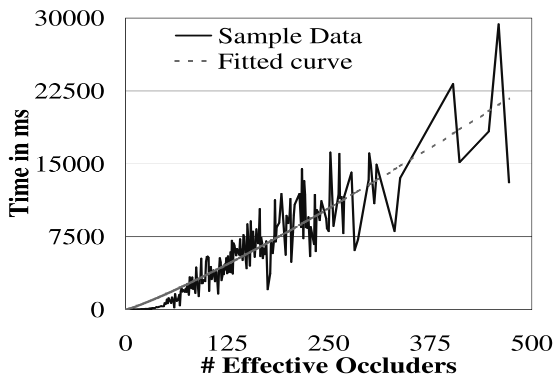
We timed approximately 48000 visibility queries, running on a single processor. For each of these we counted the number of effective occluders, and computed the average time taken to query this many effective occluders. We used a least-squares fit, and found the growth to be in the order of $O(m^{1.15})$. This is depicted in Figure 4a. To increase the confidence of our fit, we excluded timings corresponding to less than 5 samples. The data resulting from this noise reduction is plotted in Figure 4b. We note that this excludes most of the larger queries, since these are infrequent. We further note a distinct inflection where the number of effective occluders is 30. Further investigation has shown the cause to be that at this point, the storage required by our polytope complex becomes larger than the L2 cache of our test machine. This behaviour is reproduced with *both* the town and the forest scene. Fitting two curves, one from 0 to 30, and one from 31 to 57, we observe the order to be $O(m^{1.15})$ and $O(m)$ respectively. The second curve pertains to a larger, hardware induced constant. Given our justification for anticipating such a result (see Section 3.4), we do not expect this complexity to vary significantly for any realistic scene.

To estimate the global scalability of our approach, we consider first the complexity of a naïve solution, where visibility is computed for a single cell by simply querying every single polygon. This would require n queries. At most, each query may be given $n-1$ polygons as input, and find each of these to be effective occluders. This gives a computation cost of $O(n^{2.15})$. For a real scene, $n-1$ effective occluders is highly unlikely. For our forest and town scenes, we have found the *maximum* number of effective occluders to be 472 and 250 respectively, whereas the average number of effective occluders are respectively 3.6 and 1.7 per query (recall that large virtual occluders are used).

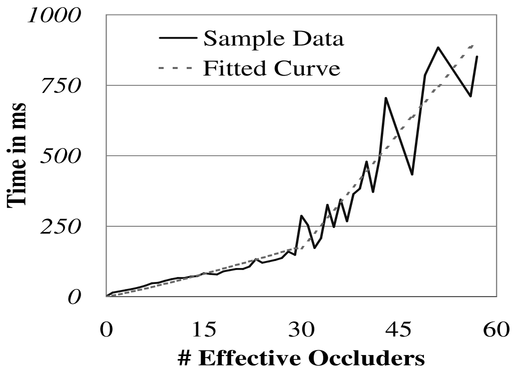
If we let, m be the maximum number of effective occluders required in *any* query for a particular scene, then we can expect a complexity of $O(nm^{1.15})$, where in practice, $m << n$. This algorithm is already scalable, however, by incorporating our general architecture, the number of queries can typically be greatly reduced (approx. 55000 queries per cell for the forest scene and approx. 42900 queries per cell for the town scene). Furthermore, m is an upper bound. We cannot give a full depiction of our algorithm's average case performance, since this would be a complex function of geometric distribution, cell configuration and scene size. Such a statistical analysis is beyond the scope of this paper. It is clear however, that the average order is below $O(nm^{1.15})$, thereby showing scalability.

The significant run-times of our experiments can be explained as a large constant factor introduced due to the initial complexity of the exact query algorithm. Indeed, the initial polytope of a query typically consists of hundreds of faces.

Returning to the number of queries performed; for the forest scene, the output sensitivity of our algorithm (see Section 4.3) results in approximately 97% of the preprocess computation time being spent on the first, and therefore nearest, 8% of the scene. For general high depth complexity scenes, we can expect the dominant component of the run-



(a)



(b)

Figure 4: Time vs. Effective occluders. *(a) The average time for queries consisting of varying numbers of effective occluders. The fitting curve shows growth of the order* $O(m^{1.15})$*. The average distance from the fitting curve to the sample data is 1285(ms). (b) The same data set as for (a), however those averages computed with less than 5 samples are excluded. A two curve fit is applied, showing* $O(m^{1.15})$ *growth up to 30 and* $O(m)$ *to 57. The average distance from the fitting curve to the sample data is 38(ms).*

time complexity to be a function of what is *visible*. Thus illustrating the output sensitivity of our algorithm.

6. Conclusion

We have presented a tractable exact visibility query algorithm and effectively integrated it into an architecture that allows for the efficient, output sensitive computation of accurate from-region visibility.

Our solution to the visibility problem can be used to obtain the smallest possible set of visible polygons (for the given partition), while making no sacrifice in image correctness. This is the first general 3D solution that allows for both optimal run-time performance and correctness.

Since it is an exact solution, we can expect our solution to be slower than that of approximate/conservative solutions. Our tests confirm this. Fortunately, the algorithm is scalable, implying that if the high constant cost of the algorithm is bypassed, either by simply accepting the run-time, or through amortisation via a distributed solution, then an exact and practical solution to the from-region visibility problem has been found.

6.1. Future Work

We intend to further analyse the time complexity of the general algorithm. We intend to catalogue the qualitative distinctions of line space which are made in global visibility structures, but not within our query algorithm. We believe such combinatorial analysis will lead to tight theoretical bounds. Needless to say, we will continue to investigate performance enhancement strategies which may reduce the required run-times.

6.2. Acknowledgements

We would like to thank Frédo Durand for allowing us the use of his forest scene. We would also like to thank Komei Fukuda for providing a free implementation of the double-description method. We also acknowledge the support of the South African National Research Foundation (NRF).

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. 2
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
3. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996. 5
4. C. L. Bajaj and V. Pascucci. Splitting a complex of convex apolytopes in any dimension. In *12th Symposium on Computational Geometry*, pages 88–97. ACM, May 1996. 6
5. D. Bartz, M. Meißner, and T. Hüttner. Opengl-assisted occlusion culling for large polygonal models. *Computers & Graphics*, 23(5):667–679, 1999. 3
6. J. Bittner, P. Wonka, and M. Wimmer. Visibility preprocessing for urban scenes using line space subdivision. In *9th Pacific Conference on Computer Graphics*

and Applications, pages 276–284. IEEE, October 2001. ISBN 0-7695-1227-5. 3

7. B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi. Lines in space: Combinatorics and algorithms. *Algorithmica*, 15(5):428–447, May 1996. 3
8. D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. Course 30, SIGGRAPH, August 2001. 1, 2
9. D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998. 2
10. F. Durand. *3D Visibility, analysis and applications*. PhD thesis, U. Joseph Fourier, 1999. http://graphics.lcs.mit.edu/ fredo. 2, 3, 8
11. F. Durand, G. Drettakis, and C. Puech. The 3d visibility complex: A new approach to the problems of accurate visibility. In *Eurographics Rendering Workshop 1996*, pages 245–256, Porto, Portugal, 1996. Eurographics / Springer Wien. 2, 3
12. F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In *Proceedings of SIGGRAPH 97*, pages 89–100. ACM, 1997. 2, 3, 7
13. F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000. 3
14. K. Fukuda and V. Rosta. Combinatorial face enumeration in convex polytopes. *Computational Geometry*, 4:191–198, 1994. 5
15. Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):542–551, 1991. 3
16. 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. 3, 7
17. J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, 2000. 3, 8
18. 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. 3, 8
19. V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Hardware-accelerated from-region visibility using a dual ray space. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 205–216. Eurographics, June 2001. 3, 7
20. 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. 3
21. M. Pellegrini. Stabbing and ray shooting in 3-dimensional space. In *6th ACM Symposium on Computational Geometry*, pages 177–186. ACM Press, 1990. 7
22. M. Pocchiola and G. Vegter. The visibility complex. In *The Ninth Annual Symposium on Computational Geometry*, pages 328–327. ACM Press, New York, NY, USA, 1993. ISBN 0-89791-582-8. 3
23. F.-T. Pu. *Data Structures for Global Illumination and Visibility Queries in 3-Space*. PhD thesis, University of Maryland, College Park, MD, 1998. 3
24. 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. 2
25. G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000. ISBN 1-58113-208-5. 3
26. D. Sommerville. *Analytical Geometry of Three Dimensions*. Cambridge University Press, 1959. 3
27. J. Stolfi. *Oriented Projective Geometry : A Framework for Geometric Computations*. Academic Press, 1991. ISBM 0126720258. 3
28. S. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley, 1992. 2, 3
29. S. Teller and M. Hohmeyer. Stabbing oriented convex polygons in randomized $o(n^2)$ time. In *Jerulsalem Combinatorics*, volume 178, pages 311–318. American Mathematical Society, Providence, RI., 1993. 4, 5
30. S. J. Teller. Computing the antipenumbra of an area light source. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 139–148, Chicago, Illinois, July 1992. 2, 3, 8
31. 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. 2
32. 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. 3
33. H. Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, University of North Carolina at Chapel Hill, 1998. 2
34. 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. 3

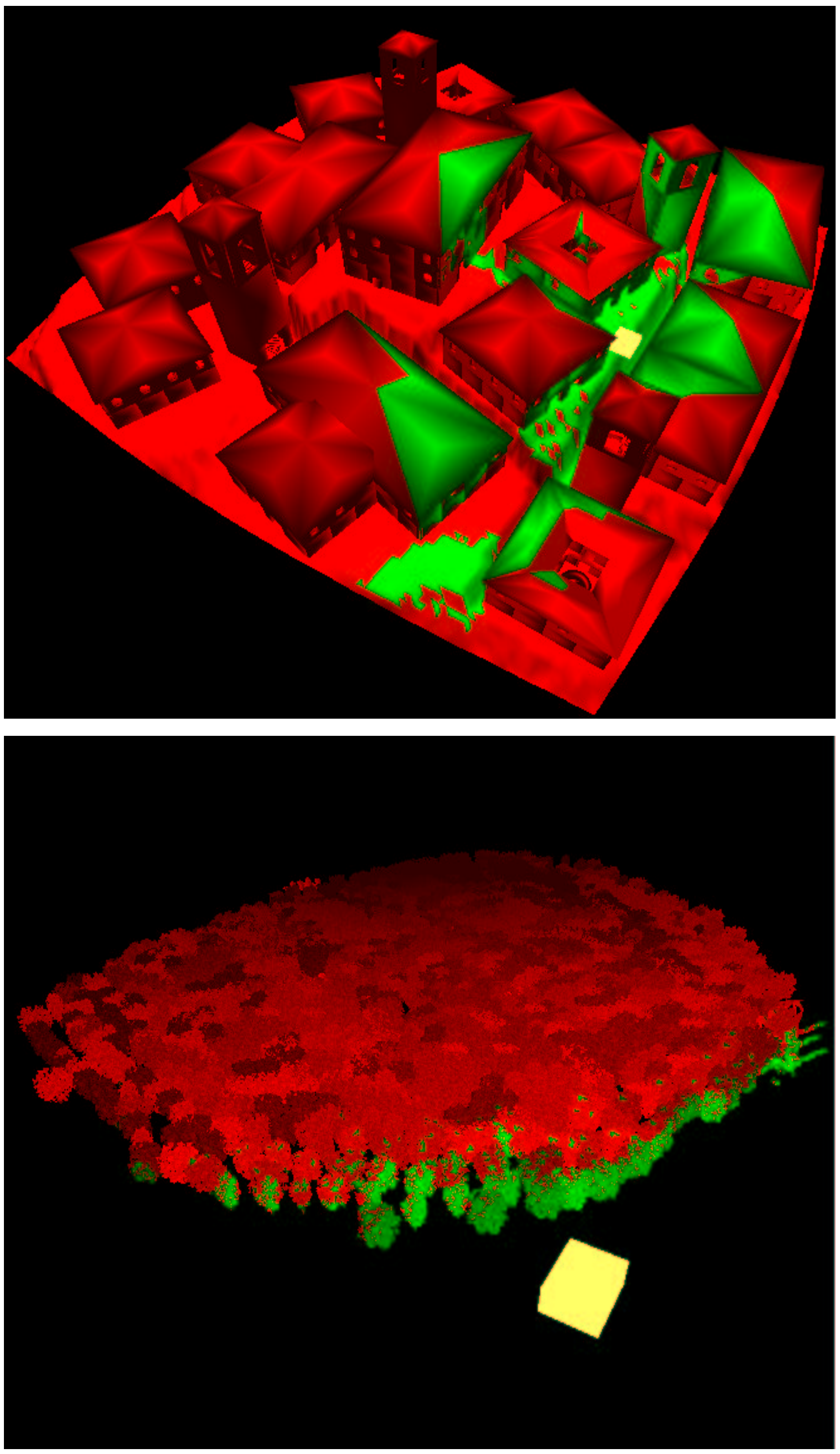


Figure 5: Algorithm results. *The top image shows a view of our town scene, while the bottom image shows a view of the forest. In both images, the yellow block corresponds to a view-cell. The geometry visible from the view cell is rendered in green, while occluded geometry is rendered in red.*