FAST AND ACCURATE VISIBILITY PREPROCESSING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Shaun Nirenstein
October 2003

Supervised by
E.H. Blake

This thesis is dedicated to my father Issadore Nirenstein, my mother Rosanne Nirenstein and to the memories of Jack Nirenstein and Sidney Sapire.

Abstract

Visibility culling is a means of accelerating the graphical rendering of geometric models. Invisible objects are efficiently culled to prevent their submission to the standard graphics pipeline. It is advantageous to preprocess scenes in order to determine invisible objects from all possible camera views. This information is typically saved to disk and may then be reused until the model geometry changes. Such preprocessing algorithms are therefore used for scenes that are primarily static.

Currently, the standard approach to visibility preprocessing algorithms is to use a form of approximate solution, known as *conservative* culling. Such algorithms over-estimate the set of visible polygons. This compromise has been considered necessary in order to perform visibility preprocessing quickly. These algorithms attempt to satisfy the goals of both rapid preprocessing and rapid run-time rendering.

We observe, however, that there is a need for algorithms with superior performance in preprocessing, as well as for algorithms that are more accurate. For most applications these features are not required simultaneously. In this thesis we present two novel visibility preprocessing algorithms, each of which is strongly biased toward one of these requirements.

The first algorithm has the advantage of performance. It executes quickly by exploiting graphics hardware. The algorithm also has the features of output sensitivity (to what is visible), and a logarithmic dependency in the size of the camera space partition. These advantages come at the cost of image error. We present a heuristic guided adaptive sampling methodology that minimises this error. We further show how this algorithm may be parallelised and also present a natural extension of the algorithm to five dimensions for accelerating generalised ray shooting.

The second algorithm has the advantage of accuracy. No over-estimation is performed, nor are any sacrifices made in terms of image quality. The cost is primarily that of time. Despite the relatively long computation, the algorithm is still tractable and on average scales slightly super-linearly with the input size. This algorithm also has the advantage of output sensitivity. This is the first known tractable exact solution to the general 3D from-region visibility problem.

In order to solve the exact from-region visibility problem, we had to first solve a more general form of the standard stabbing problem. An efficient solution to this problem is presented independently.

Acknowledgements

I would like to thank my supervisor, Professor Edwin Blake. Without his unrelenting confidence in my abilities and his guidance, my life would have taken a very different track. I also wish to thank Dr. James Gain for the immense amount of feedback I received while writing this thesis. I can only hope to achieve his standards of excellence.

I would also like to thank the National Research Foundation for funding my research.

I am indebted to my students, Adrian Sharpe and Matthew Hampton, who performed the implementation and evaluation of the accelerated ray-tracer/preprocessor (see Section 4.4), and contributed several key ideas. I would also like to extend my gratitude to my students, John Lewis and Gary Oberholster, who implemented the parallel preprocessor outlined in Section 4.5.

I am grateful to all those who have given input over the years. This includes (in roughly chronological order) Gerhard van Wagenin, Nicholas Holzschuch, Ashton Mason, Fabian Nunez, Patrick Marais, Yiorgos Chysanthou, Henri Laurie, Rudi Penne and Jiri Bittner. I would also like to thank the crew at EM Software and Systems for the opportunity to work at such a dynamic company. In particular, I would like to thank them for the understanding shown while this thesis was completed. I also thank all members of the CVC lab, for the opportunity to work with such an incredible group of people.

I would like to thank my grandparents, Myer and Thelma Saker, for providing me with endless inspiration and valuable advice. I wish to thank my parents, Issy and Rosanne, my brother Grant, Hilton and Leonie Saker, and Margaret and Bradley Bouwer for their encouragement, emotional support and inspiration. Last, and certainly not least, I would like to thank my beautiful wife Athena. Through her, I find myself truly alive.

Preface

We adhere to certain conventions within this dissertation. In general, vectors are capped with arrows (e.g., $\vec{a}$), points, coordinate components and integers are referenced by the lower case letters of the standard English alphabet (e.g., a). Differentiation between points and integers should be clear from the context. Sets are denoted by the upper case letters of the standard English alphabet (e.g., A). Scalars are represented by lower case Greek letters (e.g., α). The notation for polyhedra may vary, depending on whether they are treated as sets of points explicitly (e.g., $\ldots \in P$), or as elements of a set themselves (e.g., $p \in \ldots$). Other particulars are defined in context.

All proofs that we consider interesting, yet non-essential are included in Appendix A for the reader.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

When viewing 3D scenes of high depth complexity, only a small proportion of scene primitives are visible from any given view-point. It is desirable to remove these invisible primitives as early in the graphics pipeline as possible. This prevents unnecessary wastage of system resources. The principal advantage of such *output sensitive* rendering, is that the resulting rendering time is proportional to the visible primitive count. This feature allows rendering time to become invariant with the overall complexity of the scene and, in practice, allows for higher frame rates and/or more complex scenes.

The class of algorithms that perform such a partitioning of a scene into visible and invisible primitives (*polygons* for the purposes of this dissertation), are known as *visibility culling* algorithms. It is important to differentiate between *hidden surface removal* [SSS74] algorithms, that determine which fragments of the scene geometry are visible, and visibility culling algorithms that remove invisible primitives from the graphics pipe-line before they are even transformed from object space. Hidden surface removal is mandatory and is still applied, but only to those primitives considered visible by the culling algorithm.

Visibility culling algorithms may be categorised according to their accuracy in differentiating between visible and invisible primitives. We augment the taxonomy of Cohen-Or *et al.* [COCSD03] and discriminate between *conservative*, *approximate*, *exact*, and additionally, *aggressive* visibility algorithms (Table 1).

Conservative techniques consistently overestimate and incur what we term *false visibility* errors. These occur when primitives that are not visible are considered visible. This results in sub-optimal run-time performance, due to the unnecessary submission of invisible primitives to the rendering pipeline[1].

[1]In this context, the term “optimal” refers to rendering time only. We acknowledge that practical concerns, such as

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

Table 1:
Classification of visibility culling algorithms.

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

Approximate visibility techniques incur both false visibility and false invisibility errors. For some applications such algorithms may offer a desirable compromise between the advantages and disadvantages of conservative and aggressive techniques.

Exact visibility solutions provide both accurate images and optimal rendering performance. An exact visibility query will produce a set containing no more or less than the set of visible primitives.

Another dimension in the classification of visibility culling algorithms is the distinction between *from-point* and *from-region* visibility. The terminology introduced so far has been with reference to from-point visibility, which is simply the determination of visibility from a single camera view-point. This terminology can be extended naturally to from-region visibility.

The set of primitives visible from a *region* is defined as those that can be seen from at least one view-point within that region. Equivalently, from-region visibility can be defined as the union of the visible primitive sets from every point within the region. If this visibility set is overestimated, the from-region visibility algorithm is conservative, if it is underestimated, then it is aggressive. If both visible and invisible primitives are incorrectly classified, then the algorithm is approximate. Finally, if the set of primitives reported visible by an algorithm is exactly the set of primitives that are visible, then the algorithm is exact.

There are several advantages to using from-region visibility. The most commonly cited, is that the overhead of visibility culling can be relegated to a pre-process. The camera view-point space is partitioned into a *finite* set of regions and the set of visible primitives for each region can then

the additional time taken to query visibility itself, could outweigh the benefits of using a more accurate visibility set. However, since this dissertation is concerned primarily with preprocessed visibility, this run-time query time is trivial.

be computed and stored. At run-time, only the set of primitives visible from the region containing the current camera view-point is rendered. This saves the often expensive cost of run-time visibility determination. Pre-processing is best suited for scenes that are largely static, and so, this dissertation focuses primarily on handling static scenes. Since from-point visibility may be applied at run-time, such algorithms are usually well suited to dynamic scenes.

From-point visibility has the advantage of only ever having to solve visibility from a single point. A point may be considered to be a degenerate region, implying that the from-point problem may be considered a special case of the from-region problem. From-point solutions take advantage of special structure, often applying optimisations that cannot be applied to the general from-region case. This is critical, since the application of from-point visibility algorithms occurs at run-time on a per-frame basis. To be effective, from-point techniques have to execute in significantly less time than it would take for the graphics hardware simply to render all scene primitives. To date, all from-point algorithms have been either conservative or approximate in nature.

By contrast, the pre-processing stage of from-region techniques occurs once only, and is usually computed offline. This allows more time to be allocated to solving for more accurate visibility. Exact quantification of this time, however, depends on the nature of the application.

It should also be noted that for many applications the final pre-process is generally performed not by the end user, but rather the producers of the scene model. In such scenarios, computing facilities such as high-end machines, or even machine clusters are often readily available.

During the development of the scene model, however, it is often desirable to pre-process the scene many times for the purposes of previewing or prototyping. For this type of application it is necessary to perform the pre-processes rapidly.

At present, conservativity is the industry standard for culling as a pre-process. Typically, conservative algorithms are used both during development and as a final pre-process.

We recognise two opposing problems. Firstly, we note that for some applications, very fast visibility pre-processing is required – often faster than what is available. Indeed, for large models, state of the art techniques may take several hours to pre-process. Secondly, after this pre-process, there is the undesirable consequence that the result is still an overestimation of what is truly visible from the pre-processed regions.

Yet another issue is the lack of consistency in this overestimation. Different algorithms respond differently to the characteristics of different models. Some combinations of algorithm and model may allow very accurate results to be obtained, while other combinations may be completely ineffectual. It is inconvenient and difficult to manage this inconsistency. Many applications are limited

to certain types of models in order to accommodate the visibility subsystem. The fact that the output of any exact algorithm is unique, removes such inconsistency and unpredictability. Furthermore, the existence of an exact algorithm defines a yard stick, against which the output of non-exact visibility algorithms can be measured for evaluation.

In this dissertation we present two new visibility pre-processing algorithms. The first is an aggressive algorithm that solves for visibility more rapidly than any previous solution. It also has the advantage that it is not prone to the overestimation of approximate or conservative solutions. Our algorithm exploits standard graphics hardware and use adaptive sampling guided by heuristics in order to minimise error.

The second algorithm is a tractable exact visibility algorithm. We consider this to be our major contribution, since finding such a solution has been considered an important open problem. Indeed, even the existence of a tractable solution has been doubted [Pla92]. The pre-processing time of our exact algorithm is longer than most conservative techniques, but it does maintain near-linear scalability. In order to solve the problem we have had to extend existing algorithms and develop new ones. We detail our approach and list our contributions in the next two sections.

1.1 Approach

For our aggressive algorithm, our primary aim is performance. In order to achieve fast preprocessing, we generalise a graphics hardware technique known as *item buffer*[2] rendering to a region. This is accomplished by aggregating item buffer samples over the surface of the region.

To minimise the error resulting from aliasing while sampling the surface, we use adaptive sampling. We develop several heuristic error measures that guide the sampling and provide user control over a performance/error tradeoff.

The adaptive sampling also increases performance, since over-sampling is avoided. To exploit the reuse of samples, and to allow a progressive construction of the cell partition efficiently, we introduce a sophisticated sampling scheme that only maintains samples that will definitely be required again. By integrating the visibility information available during the preprocess with rapid from-point rendering techniques, we are able to accelerate the item buffer evaluation significantly.

We observe that one of the advantages of the sampling process, is that the item buffer encodes directional information. We consider this to have many potential advantages for ray-tracing. By subdividing the item buffer into several directional strata, we are able to compute a subdivision of

[2] Item buffer rendering is an efficient method for computing the set of visible primitives from a point [HA00].

ray-space, that allows the fast lookup of ray intersections, while also accounting for occlusion.

When solving the exact from-region visibility problem, our aims are both tractability and accuracy. Our first step is to determine the type of processes required to compute the mutual visibility between two polygons. Since visibility is defined in terms of sight-lines, we turn to a direct representation of line space. We use a formalism, known as *Plücker coordinates*, that allows lines to be represented as points on a hypersurface in a five dimensional projective space.

The advantage of this parameterisation is that it conveniently represents the set of lines through a set of polygons as a single polyhedral volume. We develop several mathematical results that allow for a closed form computation of this polyhedron in the important case where only the set of lines through two polygons are required.

Similarly, the occluders between a pair of polygons can each be represented as a polyhedral volume. Performing polyhedral set subtraction in Plücker space allows us to compute the the set of lines that do not intersect a particular set of occluders. This solves the selective stabbing problem[3], of which the mutual visibility problem is a special case.

To achieve efficient polyhedral set subtraction in five dimensions, we improve on an existing polyhedral complex splitting algorithm and adapt it to become a context aware polyhedral subtraction algorithm.

Given that we can now compute a representation of the set of lines visible between two polygons, we integrate this *query* algorithm into a framework that uses it selectively and effectively. Experimentally, we verify that the resulting technique is scalable and tractable in nature.

1.2 Contributions

We divide our contributions into three sections: *Aggressive Techniques*, *Selective Stabbing* and *Exact Techniques*. Selective stabbing is not related to visibility exclusively, but is an important geometric problem that required a solution in order for us to solve the exact from-region visibility problem. The theoretical results arising from this research should contribute greatly to any work requiring geometric manipulations in Plücker line space.

1.2.1 Aggressive Techniques

We have contributed a new aggressive algorithm that may be used to preprocess very large, very complex models rapdily. We present the results of analysis and experiments that quantify both the

[3]Compute a representation for those lines that intersect a set of polygons S, but do not intersect a set of polygons M.

performance and the accuracy of this algorithm.

In the process of developing this algorithm, we have also contributed the following:

- *From-surface visibility.* We have developed a new aggressive visibility algorithm that determines the set of polygons visible from a surface. The algorithm exploits hardware accelerated rendering to sample visibility rapidly. A novel adaptive subdivision heuristic is used to minimise the error.

- *Framework.* We present, implement and verify a general divide and conquer strategy for sample based visibility. Our strategy provides a novel method for effective sample cache management, allowing us to process the scene progressively at no additional cost. This ensures that our aggressive from-surface visibility algorithm is used effectively.

- *Aggressive ray tracing.* We present, implement and experimentally verify an extension to the classic *5D Ray Classification* algorithm [AK87], that makes it more suited for interactive rendering. The algorithm is implemented as a natural extension of our aggressive visibility algorithm to 5D. The visibility algorithm efficiently solves the occlusion problem encountered by Arvo and Kirk.

1.2.2 Selective Stabbing

This problem is a natural generalisation of the well researched *polygon stabbing problem* from computational geometry. The problem is cast as a constructive solid geometry problem in Plücker space. We present and experimentally analyse our efficient solution to this problem.

Apart from the solution to this problem, we have made the following contributions:

- *Two polygon case.* We give a best and worst case optimal $O(nm)$ algorithm for directly constructing the 5D polytope representing the set of lines through two polygons. Our solution is more efficient than the general solution, and takes advantage of novel results specific to the two polygon case. In particular, we prove that the 5D polytope has the structure of a 4D polyhedron extruded along a particular direction. We give a *closed form* solution for computing this direction and this polytope.

- *Polytope Splitting.* An improvement of the polytope-complex splitting algorithm of Bajaj and Pascucci [BP96] that yields an efficient splitting algorithm, fully sensitive to the zone of the splitting hyperplane.

- *CSG in general dimension.* We present an optimised polyhedral set subtraction algorithm that allows for the rapid subtraction of a polyhedron from a polyhedral complex, using our improved polytope splitting algorithm.

1.2.3 Exact Techniques

We present the first tractable exact from-region visibility algorithm. We present empirical evidence addressing the scalability of this algorithm. We experimentally verify its performance on large and realistic models.

In the process of developing this algorithm, we have also contributed the following:

- *Visibility query.* We detail how an output sensitive, exact polygon-to-polygon visibility query may be cast as a special case of the selective stabbing problem. Using our efficient solution to this problem, we attain an efficient algorithm for rapdily querying the mutual visibility between two polygons.

- *Occluder ordering.* We have developed effective heuristics for selecting good orders of occluder processing.

- *Framework.* We develop a novel framework for the visibility query algorithm. The framework effectively utilises the query, in order to prevent redundant computation. Virtual occluder generation, and the reuse of line space structures are used to ensure that previous computations are exploited whenever possible.

1.3 Overview

We begin with a broad survey of visibility culling and exact global visibility techniques in Chapter 2. Chapter 3 then presents certain mathematical preliminaries that are required to fully understand this thesis, but typically are not covered in undergraduate studies.

We begin the presentation of our work in Chapter 4, where we detail our aggressive visibility algorithm in its entirety. Accompanying results are included in the chapter.

We then present our solution to the selective stabbing problem in Chapter 5. This includes detailed descriptions of the extensions we make to existing algorithms and the new algorithms developed in order to achieve a solution.

In Chapter 6 we cast the polygon to polygon visibility query problem as a selective stabbing problem. We show how this query may be used to solve efficiently for the exact visibility set of a region. Results illustrating the performance of our technique are presented at the end of this chapter.

Finally, we finish with a brief conclusion and a description of future work.

Chapter 2

Background

In this chapter the relevant background literature in the field of visibility culling is discussed. We classify visibility culling algorithms into two classes: *from-region* and *from-point* visibility in addition to the classification presented in Table 1. This is because, geometrically speaking, these are two very different problems.

We discuss the most significant visibility culling techniques for both of these classes. We do not however discuss hidden surface removal algorithms. Sutherland *et al.* [SSS74] have published a survey of such techniques.

Examples from both classes are reviewed, although most emphasis is placed on from-region techniques, since it is in this class of algorithm that the contributions of this thesis lie. We also include a detailed discussion of analytic visibility methods since our solution is strongly related.

Notable alternative background reading for visibility includes the excellent in depth survey of visibility for walkthrough applications by Cohen-Or *et al.* [COCSD03] and the multi-disciplinary survey by Durand [Dur99].

2.1 From-point vs. From-region Visibility

The geometric differences between from-point and from-region visibility have several implications on their use in practice. We discuss some of these implications briefly, since they govern the methodologies used in the background literature.

2.1.1 Accuracy

From-point techniques have the potential to compute exactly the visible subset of geometry from a given view-point [WA77]. For real-time applications, however, this is infeasible, since the time taken to compute such a subset is typically greater than that taken to render the whole scene using a z-buffer, and is certainly greater than the time required to render an efficiently computed conservative estimation of the visible scene geometry. For this reason, most of the literature focuses on conservative estimations for run-time from-point visibility.

From-region techniques, in contrast, do not in general possess the potential to compute the exact set of visible geometry from a view-point. This results from the definition of a region. Namely, that it contains many view-points that may see different geometric primitives. It is possible to subdivide the view-point space into regions of constant visibility, however, such subdivision quickly becomes combinatorially complex and is infeasible in practice. The result being, that in practice, from-region visibility is in general, a conservative estimate of from-point visibility. This relative conservativity can be reduced, by reducing the size of the viewing regions.

The advantage of from-region techniques is that the visibility computations can be performed offline, as a preprocess. When sufficient preprocessing time is available, it is possible to preprocess scenes heavily enough, such that the estimates become less conservative. This is achieved by more sophisticated algorithms that account for complex occluder interactions and the visibility of fine geometry (e.g., per triangle).

2.1.2 Time Bounded Visibility

It is often desirable to compute a lower bound on the amount of time before a set of geometric primitives can become visible. Such a time bound has many uses in practice, and is an effective tool for handling complexity. The most common application is to define priorities on geometry and textures (also light, bump, normal maps, etc.), such that the priority is highest for currently visible geometry (and associated mappings), and lowest for geometry/mappings with the longest lower time bounds.

The layout of geometry and mappings in memory can be structured such that highest priority elements are in the highest performance memory (i.e., video card or AGP), and the lower priority elements are in the lower performance memory (i.e., main memory, secondary storage, or even on a remote host [COZ98, TL01]). This allows from-region visibility to optimise not just the rendering pipeline, but all the resources of the system. Effectively, this process can be considered to be a form

of *predictive cache management*.

In order to obtain these lower bounds on "time till visibility", it is necessary to constrain the movement of the camera to some degree. Shortest path finding algorithms [FST92] and bounds on camera movement speed can be used to bound the time until a camera leaves a region, or can possibly reach another given region. Wonka *et al.* [WWS01], use this concept to remove the pre-processing time and storage requirements for their solution to from-region visibility. They compute from-region visibility at run-time, concurrently on another machine. The visibility of the surrounding view-point space is computed before the camera can cross over into them.

Correa *et al.* [CKS02] use predictive heuristics to look ahead for potentially visible objects. Overestimation results in inefficiency, while under estimation results in popping artefacts. A complete out-of-core solution is implemented.

From-point visibility techniques cannot provide such management, since they are not, by definition, aware of the camera locale. We believe that from-region and from-point visibility algorithms may be combined to achieve both a higher degree of culling and complexity management.

2.2 From-point Visibility

Run-time visibility algorithms have a non-trivial run-time component but often also a small pre-processing component. In this section we survey the various run-time visibility culling techniques proposed in the literature[1].

2.2.1 Shadow Frusta Culling

Hudson *et al.* [HMC+97] describe a visibility algorithm based on the construction of shadow frusta. A spatial subdivision is generated to partition the view-point space. A set of occluders is chosen for each partition or *cell*. These occluders are selected as a pre-process, and selection is based on a heuristic that attempts to maximise occlusion.

At run-time, a shadow frustum is created for each selected occluder (see Figure 1) with respect to the view-point. If an object falls completely inside a single shadow frustum it is considered to be occluded (*shadowed* if you consider the view-point to be a light source). If an object intersects the boundary of one of these frusta it is said to be *partially visible*. Since this algorithm is conservative, it treats all partially visible objects as visible and lets the depth buffer determine the visible fragments in image-space.

[1] We discuss from-point cell-portal rendering in Section 2.3.3 for clarity.

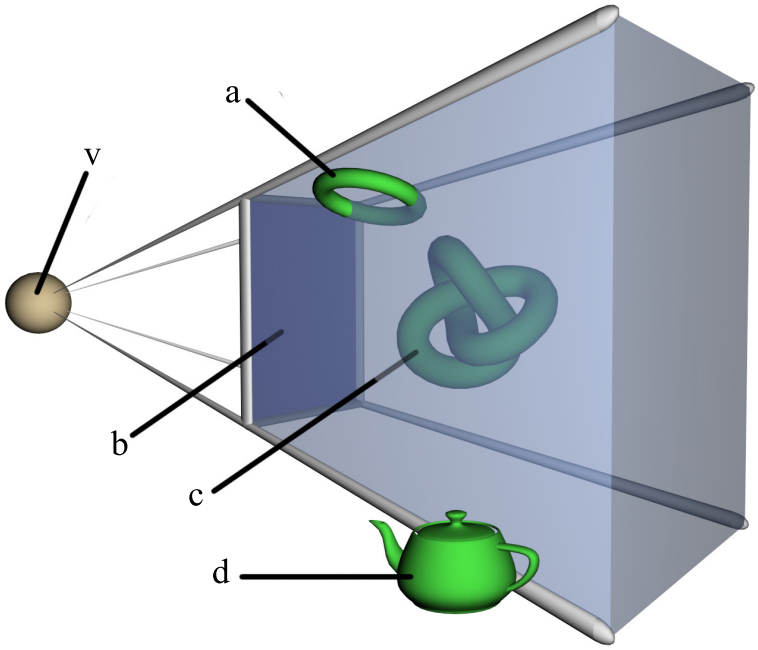


Figure 1: *Shadow Frustum* The shadow frustum cast by occluder b with respect to view point v. Any object falling *completely* inside this frustum is considered invisible. Respectively, objects a, c and d are partially visible, occluded and fully visible.

A hierarchy of bounding volumes is created over the model geometry to enable efficient *containment* tests on the shadow frusta. The test is only refined to the level of cells. Individual polygons are not treated. This results in a tradeoff, namely an increase in rendering time for a decrease in algorithm execution time.

This algorithm is conservative because:

1. Only a subset of potential occluders are considered
2. Objects that are necessarily occluded by more than one occluder are always considered visible
3. *All* geometry within a visible (by criteria 1 and 2) leaf bounding box is considered visible

As the size of the bounding boxes increase, they becomes less likely to be occluded by a single object. This algorithm performs best for scenes consisting of a combination of large and small polygons, where the large polygons can be used as occluders for clusters of small polygons.

2.2.2 An Incremental Aspect Graph Approximation

Coorg and Teller [CT96, CT97], show how a tractable approximation (subset) of an *aspect graph* (detailed in Section 2.5.5) can be generated using a predetermined subset of occluders that implicitly represent a coarse approximation of the graph. This approximation is evaluated lazily as the camera crosses the boundaries of the approximate aspect graph cells.

Where the complete aspect graph subdivides the view-point space into regions of constant aspect, a considerably more coarse representation may be generated using only a small subset of the full spatial subdivision. This subdivision is that defined precisely by the set of *relevant planes*. These account for a small subset of *visibility events* (see Section 2.5.2).

The relevant planes implicitly define a view-cell for the current position of the camera. When one of the relevant planes bounding this cell is crossed, the maintained set of visible objects is updated with the associated event. As the camera moves, occluders are incrementally added and subtracted to a set of active occluders. As this occurs, relevant planes are dynamically added to and subtracted from a maintained plane set. In order to determine visibility events, queries are preformed to determine the event associated with each relevant plane. To perform these queries efficiently, an octree bounding box hierarchy is generated over the model. The visibility of the cells is evaluated hierarchically.

Despite the different nature of the implementation, the relevant planes are chosen such that the handled events are equivalent to that of an object moving completely interior a shadow volume (or equivalently, become partially exterior to a shadow volume). Thus the conservativity of this technique is equivalent to the shadow frustum technique of Section 2.2.1

2.2.3 Occluder Trees

Bittner *et al.* [BHS98] describe a similar algorithm to the two discussed above. However, where the above algorithm considers each shadow frustum individually, Bittner *et al.* implement a modification to account for occluder collusion/fusion. In a similar manner to Hudson *et al.* [HMC+97], a set of potential occluders is determined in a pre-process and a subset of these is selected at run-time.

BSP trees are commonly used as a mechanism to perform set operations on polyhedra [TN87, NAT90]. In the context of occluder fusion, the desired operation is the union of shadow frusta. A shadow volume BSP structure is built from the selected occluders. Using this, the shadow frusta are merged implicitly. A containment test is performed by traversing the BSP tree and determining whether or not the object in question (a node in a bounding volume hierarchy) falls into the union of

the shadow frusta. This check is more complex, and thus more computationally intensive than the one described in the original shadow frusta algorithm, however, considerably more culling is likely to be achieved.

2.2.4 Occlusion Maps

There is a set of techniques that use image based methods to compute whether or not an object is occluded. The general approach is to render a small subset of the scene from the current viewpoint. From this rendering, an occlusion map is synthesised. Each presently un-rendered object is tested against the occlusion map. If an object is deemed occluded (i.e., behind the occlusion map), then it is omitted, otherwise it is rendered. The rendered object may be integrated into the existing occlusion map in order to account for what it may further occlude. Temporal coherence is exploited, by rendering the visible object from the previous rendering first.

The techniques that follow this general pattern differ in various ways:

1. The method used to decide the order of in which the objects are rendered
2. The method for testing objects against the occlusion map
3. Whether or not rendered objects can be integrated into the occlusion map

The particular techniques we consider are the hierarchical Z-buffer/tiling [GK93, Gre96], hierarchical occlusion maps [ZMHI97, Zha98], a conservative augmentation of the prioritised-layered projection algorithm [KS01] and new hardware based approaches [BMH98, BMH99, SOG98, Reg02].

Hierarchical Z-buffering/Tiling

Greene and Kass [GK93] present the hierarchical Z-buffering algorithm. The algorithm begins with a pre-process: an octree bounding box structure is built over an existing scene. A Z-pyramid is used as an occlusion map. The base of the Z-pyramid is a Z-buffer in the traditional sense. The buffers above this base are each a quarter the size of the buffer directly below them and are generated recursively from the level below in a technique similar to bilinear filtering. Instead of taking the average, from each group of four adjacent buffer elements, the *farthest* Z value is extracted. This is illustrated in Figure 2.

The visibility test begins at the root node[2]. If the bounding box of the node is visible, the visibility of its children are evaluated. If the bounding box of the node is occluded then it and its

[2]In practice the root node cannot be occluded.

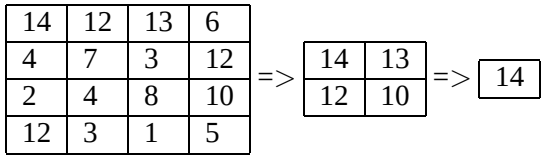


Figure 2: *Hierarchical Z-Buffer*. An example of the three top levels of a Z-pyramid. The 4x4 grid on the left is transformed to the apex on the right.

descendents are culled. This process is recursively applied to all visible nodes. If a leaf node is determined to be visible, the contained geometry is rendered.

The visibility test for a node begins by testing the front facing polygons of its bounding box against the tip/root of the Z-pyramid. For each node it is possible to test whether the nearest vertex of an octree node is further than the furthest pixel in the Z-pyramid. If this is true, all polygons associated with the octree node are culled. If the nearest vertex is closer than this value at the root/tip, then the depth of the octree node is compared to the values in the lower level in the same way. This continues recursively, until the comparison is done at the leaf level. If the object is visible, it is rendered and integrated into the Z-pyramid.

When testing the children of a potentially visible octree node, the children are visited in a front-to-back order (easily determined from the relative orientation between the camera and the octree), since drawing the closest nodes first increases the probability of the occluded far nodes quickly being classified as occluded.

Hierarchical Occlusion Maps

Zhang presents an algorithm [ZMHI97, Zha98] that exploits standard graphics rasterisation hardware. A preselected[3] set of occluders are rendered onto the screen. The image is then read back. This gives a bitmap containing the aggregate renderings of the occluders. An opacity value is used as the bitmap elements. Each bitmap element is either opaque (it has been rendered to) or fully transparent. In general, however, the opacity of a rectangular set of pixels is defined as the ratio of the sum of opaque pixels to the total area of the block.

A *hierarchical occlusion map* is derived from this bitmap by recursively averaging the bitmap values in order to create parent levels (also achieved by the efficient utilisation of texture hardware).

[3]The criterion is that these occluders are located nearer to the viewer, than the geometry they can be considered to occlude.

The projections of the bounding volume representations of the scene objects are tested for visibility against the hierarchical occlusion map. The occlusion map hierarchy allows for efficient overlap querying. Should the bounding volume of some object be completely covered by the occlusion map, then it can be said to be invisible.

An opacity threshold may be set to allow for *approximate culling*. This implies that if a bounding box is visible through an area of the occlusion map that is largely opaque, the geometry inside it may be culled even though it is potentially visible. Coherence factors generally make the resulting image acceptable. This is a typical tradeoff between quality and performance.

Prioritised Layered-Projections

The techniques of Section 2.2.4 try to build up good occlusion maps by rendering the fewest possible objects. The heuristic used is a front to back ordering. This heuristic fails in certain cases, namely in those cases where most geometry is nearby, yet occluded. In such cases, those invisible occluders rendered into the occlusion map are redundant.

Klosowski and Silva [KS00, KS99] present an algorithm known as the Prioritised Layered Projection (PLP) algorithm. This algorithm presents a better heuristic that attempts to render visible geometry first. The scene is tesseletated into cells, and rendered in approximate layer by layer order. The first layer consists of geometry that is most likely to be visible, the second layer consists of geometry that is likely to be occluded by the first layer, etc. Geometry in the n-th layer is more likely to be visible than that of the $(n + 1)$-th layer. Given a triangle budget of k polygons, the first k elements under this ordering gives a good approximation to the visible set. Such rendering is useful for rendering complex scenes at a set frame-rate, while still attempting to maximise the visual quality, even if not all visible geometry is drawn.

Klosowski and Silva [KS01] use the PLP algorithm to render better occlusion maps. They also consider several hardware extensions to enhance from-point visibility techniques.

Hardware Techniques

A significant problem with the techniques discussed so far, is that they require either the maintenance of a distinct occlusion map (Hierarchical Z-Buffer), or the reading of the frame-buffer (Hierarchical Occlusion Maps) at run-time. Ideally, it is desirable to maintain every rendered object (i.e., the occlusion map is updated as each object is rendered) and still avoid the overhead of auxiliary structures in main memory.

Bartz *et al.* [BMH98, BMH99] take a very similar approach to that of Zhang *et al.*, using a stencil buffer. They also however, suggest the usage of an *occlusion bit* (first proposed by Scott *et al.* [SOG98]) that can be integrated into hardware. The purpose of this bit is to return whether or not the frame buffer had changed between the setting and testing of the bit.

This functionality has been integrated into current consumer level hardware. It provides the functionality to test whether or not the rendering of some object would modify the contents of the frame-buffer. This is accomplished by determining whether or not any rendered geometry will pass the Z-buffer test. In typical usage, Z-buffer and frame-buffer writes are disabled. Next, the bounding box of a complex set of geometry is rendered. If this bounding box does not alter the occlusion bit, then the bounding box, and hence the geometry inside, cannot be seen. This test may be performed on a bounding box hierarchy in order to reduce the number of tests performed. These tests are non-trivial since they require the rendering of large bounding volumes, thus adding potentially expensive (fill-rate dependent) costs. Secondly, this visibility query requires a round trip to and from the graphics hardware before feedback is given. This introduces *latency*. Thirdly, for those visible objects, the bounding box renderings are wasted. As noted by Bartz *et al.*, better fitting bounding boxes will reduce such conservatism.

In addition to the suggestions of Scott *et al.*, Bartz *et al.* suggest the addition of extensions that will give further information, such as the number of visible pixels, the number of projected pixels and the furthest Z-value. The extension that gives the number of visible pixels has been implemented on current hardware [Reg02]. Having the number of visible pixels allows a non-trivial upper bound to be placed on the pixel error resulting from not-rendering an object. This allows for what is known as *approximate* [Zha98] or *contribution* [Reg02] culling to be performed. This is when an error tolerance (a set number of pixels) is set. If the number of visible pixels are below this tolerance, then the object is not rendered. This saves resources that would otherwise have been spent rendering geometry that does not contribute significantly to the image.

2.2.5 Summary

The goal of practical from-point visibility is to seek an efficient means to determine which geometry is occluded by a given subset of occluders. These technique may compute shadow frusta explicitly. More advanced techniques will seek to fuse the frusta together to achieve more culling.

Many techniques exploit graphics hardware to build some form of occlusion map. This is effectively a discretisation of the shadow frusta. One advantages of the occlusion map approach is that complex computations on the CPU are avoided. Most of the work reduces to simple rendering that is

very efficient on graphics hardware. Another advantage is that accurate occluder fusion is implicit. The cost of such a technique comes in the form of latency, due to the necessary communication between the graphics hardware and the CPU.

2.3 From-region Visibility

In this section we examine from-region techniques. We reserve any discussion on analytical from-region techniques until Section 2.5.

2.3.1 From-Region Visibility as a Light Source

A useful analogy for from-region visibility, is to treat the view region as a (direct) volumetric light source. The geometry that is fully illuminated is the geometry that can see the lightsource fully. Geometry that is partially lit (in *penumbra*) can only see part of the light source, and geometry that is in shadow (*umbra*) cannot see the light source at all.

The goal of from-region visibility, is to compute which primitives lie completely in umbra (as efficiently as possible). In order to achieve this, many solutions have been presented. Typically, these are either sample based, or they are analytic techniques that seek to evaluate the umbra volume explicitly.

Building the umbra volume is a difficult task. It is not sufficient to fuse the umbra of a selection of occluders. Indeed, it is often the case that the penumbra of distinct occluders interact with each other, and form a significant extension to the umbra volume. In other words, the umbra of a set of occluders, encompasses *and* extends the joint umbra of the individual occluders.

Computing only a subset of occluder interactions results in a smaller umbra volume, and is thus a conservative overestimate of what is visible. Computing a volume larger than the exact joint umbra would result in an aggressive algorithm where some visible primitives are considered invisible. Researchers of accurate illumination techniques, explicitly compute the boundaries on a surface where discontinuities in illumination take place. These discontinuities account for transitions between umbra, penumbra and full illumination. This is discussed further in more detail in Section 2.5.7. In what follows in this section, we discuss those techniques that are designed as practical means to compute from-region visibility.

2.3.2 Strong Occluders

Cohen-Or *et al.* [COFHZ98] define what is referred to as a *strong occluder*. A strong occluder is a convex polygon that by itself blocks all sight lines from one region to another. The algorithm subdivides the scene into a grid of cells (regions). The algorithm proceeds by iterating through all regions. The visible subset of the scene is computed for the region of the current iteration. The visible set is conservative, and an object O is considered visible from a region R if and only if no strong occluder exists between O and R.

Cohen-Or *et al.* [COKT02] have developed a technique that generates a partition of large convex regions that cover a simple non-convex polygon. This allows this technique (and indeed *any* technique that requires convex occluders) to be applied to scenes with simple non-convex occluder polygons.

The authors produce a statistical model (see Nadler *et al.* [NFLYCO99] for more details) for scene geometry. They show that in a scene with a certain object distribution (an approximate minimum-distance Poisson distribution) and a view region/cell smaller than the average object size, it becomes exponentially more likely for an object to be strongly occluded as its distance from the view-point increases.

This provides good motivation for using strong occluders for such scenes. These scenes are, however, becoming less and less common. The trend is towards detailed scenes with very small primitives. It is infeasible to build cells that are smaller than these small scene polygons, since the number of cells required would be excessive .

Both the algorithms of Cohen-Or *et al.* and Saona-Vásquez *et al.* [SVNB99] use this type of conservative visibility. We continue by investigating those techniques that utilise occluder fusion to generate better approximations of the exact visible set.

2.3.3 Cell-Portal Visibility

Airey *et al.* [ARJ90] developed a *cell-portal* system founded on the following observation: In a model, the contents of a cell/region (resulting from some form of spatial partitioning) is potentially visible only if a portal (convex connected sub-region of the cell boundary) of that cell is visible. Cells relate roughly to the concept of rooms, and portals roughly to the concept of doors or windows. As a pre-process, portals and cells are determined, and the mutual visibility relationships between all cells are computed. At run-time this may be refined by determining the visible set from a single view-point and the view frustum.

Airey *et al.* determine visibility (approximately) using a ray casting approach. Airey *et al.* were also the first to propose a hardware assisted visibility pre-process. Teller *et al.* [TS91] and Teller [Tel92a] show how an exact solution to cell-to-cell and cell-object visibility can be accomplished.

Cell-portal techniques rely on the scene having a roughly architectural structure. The worst case for such algorithms are scenes that consist predominantly of detailed objects. In such cases, the algorithm breaks down due to the complexity of the BSP tree subdivision, and the further complexity of computing the existence of stabbing lines (see Section 2.5.6) through many long portal sequences.

Cell-portal based algorithms have traditionally not been thought of as "occluder fusion" type algorithms. Although no occluder fusion is performed explicitly, we consider the explicit treatment of the holes through occluders (opaque parts of cells) as implicit occluder fusion.

Pre-process

In the pre-processing phase of a cell-portal algorithm, the view-point space is partitioned into convex cells separated by portals. This is usually achieved using a BSP. Other partitioning systems have been developed to generate a cell structure that relates more closely to the structural abstraction of the scene. One example is the partitioning scheme of Meneveaux *et al.* [MMB98] for architectural environments. Haumont *et al.* [HDS03], Lerner *et al.* [LCCO03] and Chrystanthou *et al.* [CCOZ98] present more general schemes for cell-portal partitioning.

Once the cell structure has been determined, it is necessary to find portals. This process is called *portal enumeration* and involves a search for polygons shared by cells that do not correspond to (opaque) scene geometry.

A set of visible cells[4] is computed for each cell. Teller determines whether a sightline exists from one cell to another by searching for the existence of a *portal sequence* to the cell in question. A portal sequence is a set of portals, all intersected by a common un-obstructed sight-line and ordered by position of intersection. Teller [Tel92a] presents solutions to the stabbing problem for 2D floor plans, axially aligned 3D environments, and arbitrary oriented polygons. The stabbing problem is discussed in detail Section 2.5.6.

In practice, not all scene geometry is treated during the visibility pre-process. It is impractical to partition the scene around highly detailed objects, since this would result in a combinatorial explosion in the size of the BSP tree. Rather, detail objects (typically defined as such by the modeller) are

[4] A visible cell is commonly called a Potentially Visible Sets (PVS)

omitted from the visibility pre-process. Conservatively, detail objects are considered to be visible if the cell containing them is visible.

Teller did however provide one higher degree of refinement. By computing the visible volume, or *anti-penumbra* [Tel92b, Tel92a] from a portal, it is possible to compute exactly which parts of the visible cells are visible. Only objects within the anti-penumbra volume are considered to be visible. This technique was considered insufficiently robust for complex scenes. Teller and Hanrahan [TH93a] later developed a relatively robust, but conservative, test that generates an overestimation of the anti-penumbra by pivoting separating tangent planes over the edges of a portal sequence. For the purposes of global illumination, this volume is refined to account for the required polygon to polygon visibility query. A line space test is used to achieve this refinement (we utilise a similar test in Section 6.2.2).

Run-time

The run-time phase can be broken down into two refinement stages. First the *coarse* phase occurs, where cell-to-cell visibility is extracted from a pre-computed visibility structure. Secondly the *precise* phase occurs, where eye to object visibility is determined.

The first phase finds the cell containing the view-point. When determined, a set of visible cells is extracted (from the stored pre-processed output data) and for each cell a set of visible objects is extracted (also from the stored output data resulting from the pre-process). The second phase involves determining visibility from a single view-point. This is effectively a clipping operation that determines those portals visible within the view frustum.

From-point Cells and Portals

Luebke and Georges [LG95] have developed an algorithm that determines at run-time which cells are visible. This is accomplished by the recursive clipping of the bounding boxes of projected portals against each other, in front to back order, to determine (conservatively) which portals are visible and hence which cells are visible.

At present, this is one of the most commonly used approaches to visibility culling in practice. As discussed in Section 2.1, this is still no replacement for from-region techniques.

2.3.4 Extended Projections

Durand *et al.* [DDTP00, Dur99] present a from-region visibility technique that seeks to fuse occluders in order to process complex scenes effectively. Their technique works by transforming occluders such that their shadow volumes are valid, not for just a single point, but rather the entire cell (or at least one side of the cell, since the sides are treated separately and then combined). They call this transformation an *extended projection* (EP).

The extended projection for an *occluder* with respect to a particular cell and chosen plane is defined as the *intersection* of all possible projections of the occluder onto the chosen plane, where the center of projection is with respect to all view-points within the cell. Effectively, the extended projection is a cross-section of the umbra region of the occluder, with respect to the view cell (lightsource). This is illustrated in Figure 3a.

The extended projection for an "occludee" with respect to a cell and a plane is defined as the *union* of the projections of the occluder onto the chosen plane. Once again, the center of projection is with respect to all view-points within the cell. In terms of the light analogy, the extended projection is a cross section through the anti-penumbra cast through the occludee. An example of this is illustrated in Figure 3b.

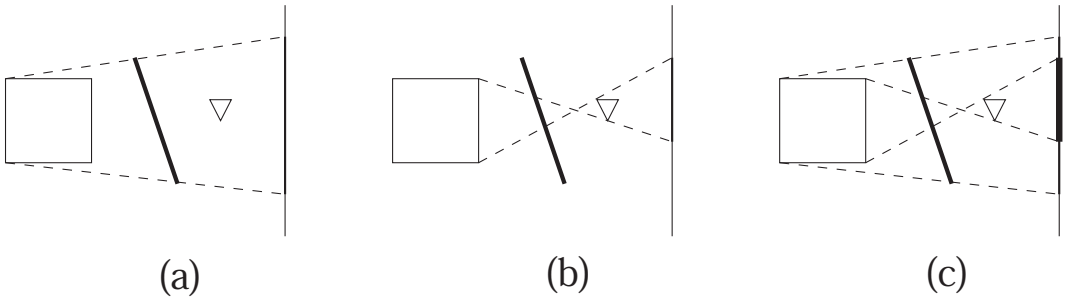


Figure 3: *Extended Projection.*(a) The extended projection of an occluder. (b) The extended projection of an occludee. (c) The extended projection of the occludee falls inside the extended projection of the occluder.

With respect to a given plane, if the extended projection of an occludee polygon is a subset of the extended projection of an occluder, then the polygon is occluded. This is also shown in Figure 3c. Durand *et al.* define $view_V^A$ to be the planar projection of A, with respect to view-point V. In his thesis Durand summarises this by the following relations: For any view-point V in a view-cell:

$$\underbrace{view_V^{occludee}}_{\text{projection of occludee}} \subseteq \underbrace{\bigcup_{W \in cell} view_W^{occludee}}_{\text{EP of occludee}} \subseteq \underbrace{\bigcap_{W \in cell} view_W^{occluder}}_{\text{EP of occluder}} \subseteq \underbrace{view_V^{occluder}}_{\text{projection of occluder}} \quad (1)$$

Durand *et al.* show that extended projections can be fused by taking the union of extended projections on the projection plane. The validity of this can be seen in the following:

$$\underbrace{view_V^{occludee}}_{\text{projection of occludee}} \subseteq \underbrace{\bigcup_{W \in cell} view_W^{occluder}}_{\text{EP of occludee}} \subseteq \underbrace{\bigcup_{occ \in occluders} \overbrace{\bigcap_{W \in cell} view_W^{occ}}^{\text{EP of occ}}}_{\text{union of occluder EPs}} \subseteq \underbrace{\bigcup_{occ \in occluders} view_V^{occ}}_{\text{union of projected occluders}} \quad (2)$$

The extended projection algorithm iterates through several planes. In order to use extended projections accumulated on a previous plane, they can be reprojected onto successive planes.

The extended projection algorithm is conservative. Consider the case where the scene consists of primitives that are significantly smaller than the view cells. When a set of small occluders are projected onto a plane, the umbra cast by the extended projections is significantly smaller than the umbra that would result from the original polygon set. Indeed, if the projection plane is too far away, the extended projections may not exist.

Cohen-Or *et al.* [COFHZ98] show that strong occlusion is most effective when the view cell is smaller than the scene objects. A similar consideration can be used to analyse the culling efficiency of the extended projection algorithm. Indeed, it is only when the accumulated extended projections and reprojections thereof grow larger than the view cell, that significant culling is achieved.

Durand *et al.* generalise reprojection into a technique known as the *occlusion sweep*. Hardware rendering is used to perform the fusion and reprojection of extended projections efficiently. This is necessary to handle large scenes consisting of many small occluders.

As an informal comparison, Durand *et al*, implement the technique of Cohen-Or *et al.* [COFHZ98] and attain results for their scenes showing that their routine culls out approximately four times more geometry and is 150 times faster.

2.3.5 Volumetric Visibility

Schaufler *et al.* [SDDS00] present a visibility technique similar to that of Yagel and Ray [YR96]. Both of these techniques discretise space into cells and both classify the cells based on whether they are opaque (interior to occluding objects), contain polygons or are empty. In their paper Schaufler *et al.* show how such a discretisation and classification can be generated from a reasonably general polygonal model: the model must consist of objects that have a *solid volume* (i.e., the surface representation for objects must be manifold meshes).

Yagel and Ray calculate the visibility sets for their cells by generating shafts between each pair of view-point cells and searching for interference by opaque cells. Similarly, Schaufler *et al.* calculate visibility by generating a shadow volume from each cell. Occluder fusion is realised by joining a subset of opaque cells together to form an extended blocker. Finally, they classify cells as being partially inside the shaft, completely in the shaft or completely outside the shaft. The cells completely inside are in the umbra of the view-cell and are therefore culled from the visibility set. They show how this can be determined efficiently in both two and three dimensions. An illustration of the discretisation and shadow frustum/shaft can be seen in Figure 4a.

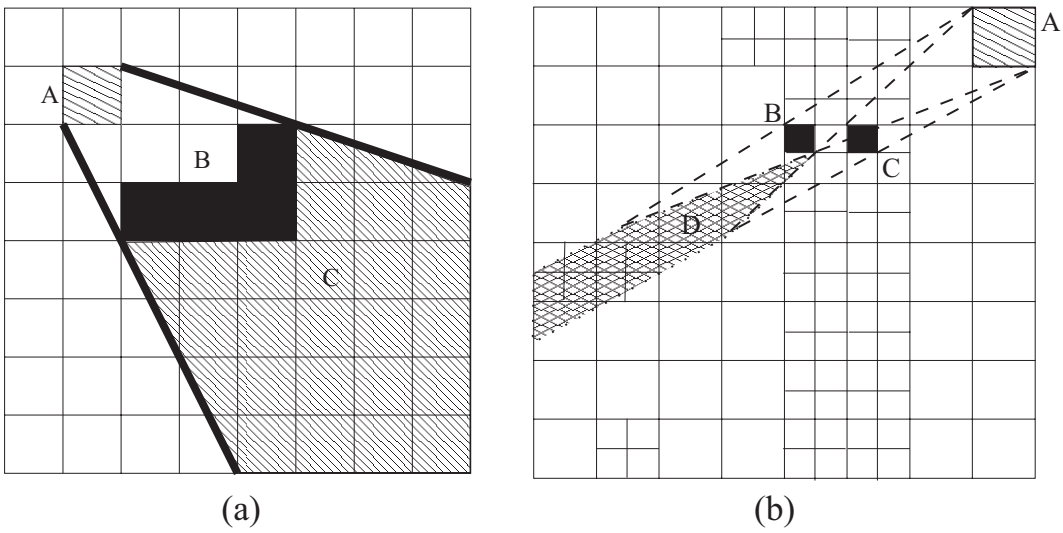


Figure 4: *Shadow Volume/Shaft Volumetric Occlusion.*(a) shows the shadow frustum/shaft C generated from cell A by object B. Note that the grid need not be uniform. (b) all cells intersecting area D cannot be seen from A, however, they would be marked as visible, since they do not fall entirely inside at least one of the shadow volumes cast by the volume occluders B or C.

The degree of conservativity for the algorithm of Schaufler *et al.* depends greatly on the type of scene to which it is applied. It performs best for scenes consisting of objects which have large volumes and are nearly convex (e.g., urban models, where building interiors are considered opaque). The algorithm fuses together the occluding cells where possible. Complex fusions, such as those resulting in a non-convex volume occluders and the fusion of disjoint cells are not performed.

2.3.6 Sampling Techniques

Gotsman *et al.* [GSF99] present a novel sample-based visibility solution. They use a 5D sub-division of ray space (three spatial and two angular dimensions). Each 5D cell maps to a *beam* in 3D space [AK87]. 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.

This technique may admit both false visibility errors and false invisibility errors. Neither performance, nor error rates are evaluated in this paper. We have ascertained that the largest model tested (450k polygons) took "in the order of hours" to pre-process [Sud].

2.3.7 5D Spatial Subdivision

Wang *et al.* [WBP98] propose a visibility culling scheme based on a point-angle 5D subdivision (x, y, z, θ, ϕ). In order to determine cell-to-polygon visibility they generate an *outer rectangular volume* (ORV) around each spatial cell. They then proceed to subdivide this ORV adaptively into rectangular patches. A *beam* is the convex hull of the union the vertices of an ORV patch and the spatial cell. Visibility is computed using a standard shaft interference algorithm [Hai00]: if a single occluder blocks the beam, then all polygons behind it (and completely contained) in the beam are not visible. If the number of visible polygons for a beam is above a set minimum, the beam is further subdivided. If a beam exceeds this minimum and is very small, the average colour of its contained polygons is used as an approximation for the beam.

The rendering process works in a similar fashion to a cubic environment map. The ORV of the view-cell containing the current view-point is mapped onto the view-plane. This occurs directly if the beam corresponding to a pixel or group of pixels is a single representative colour (as described above). If the beam contains a PVS, then this PVS is rendered as per usual onto the view-plane. Polygons in the PVS are tagged to prevent a polygon that may be in more than one PVS being rendered. The beams that do not project into the view-frustum are culled.

2.3.8 Hoops

Brunet *et al.* [BNRSV01] present a technique for finding *virtual occluders* (i.e., occluders that are not actually a part of the scene)[5] within a scene. In this case the virtual occluders are defined by *hoops*. A hoop is a closed polyline that denotes a region that *appears* to be both convex and simple from a given region. A convex umbra can then be extracted from each hoop. All geometry within the umbra (or within the union of several umbra) is occluded. The algorithm is tested on a conservative volumetric approximation of the interior of the model.

Various sufficiency conditions are presented in the paper for defining these hoops. In particular, the polygons within the mesh region bounded by the hoop define a *cap* polyhedron to be the intersection of all the positive half-spaces embedding the polygons. That the view-region falls within this polyhedron is one criterion for sufficiency.

The conditions for the sufficiency of a polyline are relaxed in [NR03]. A topological condition is presented that allows the above mentioned capping condition to be relaxed.

2.3.9 Temporal Bounding Volumes

Surdarsky [Sud98] and Sudarsky and Gotsman [SG99] present a technique for integrating dynamic objects into from-region visibility pre-computations. This is achieved by creating a *temporal bounding volume* around a moving object. This is only possible if the motion of the object is somehow constrained. The temporal bounding volume is the swath of space traversed by the motion of the object.

Conservative bounding volumes may be put around the whole motion swath. If a temporal bounding volume is occluded, then so is the moving object it bounds. Using this technique, moving objects can be occluded, but moving objects cannot occlude.

2.3.10 $2\frac{1}{2}D$ Visibility Solutions

Koltun *et al.* [KCCO00] make use of separating lines to build larger more effective *virtual* (i.e., not part of the scene geometry) occluders to represent the aggregate 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

[5] Many techniques (e.g. Schaufler *et al.* [SDDS00], Bernardini *et al.* [BKES00] and Durand *et al.* [DDTP00]) use a similar concept to that of virtual occluders. Where the algorithms differ, is in how the virtual occluders are constructed. For Durand *et al.*, the virtual occluders are the fused extended projections. Similarly, for Schaufler *et al.* and Bernadrini *et al.*, the virtual occluders are opaque cells within the volumetric objects.

cell is removed. They implement a $2\frac{1}{2}$D (height field) solution. While the theory extends to 3D separating planes, the general method is far more complex and may not be tractable.

Wonka *et al.* [WWS00] have another 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 remain conservative. There is a trade-off between the number of 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 shrinking process has been improved by Décoret *et al.* [DDS03]. Where Wonka *et al.* shrink using a sphere (optimal only for spherical cells), Décoret *et al.* generalise the technique to allow shrinking by any convex object.

This algorithm can be extended to a general 3D solutions, by tessellating and shrinking volumetric occluders. This has not yet been implemented or evaluated. The scenes best suited, are those with large volumetric objects. Scenes consisting predominantly of small volumetric objects can be treated effectively only if minimal shrinking occurs. This requires an excessive number of samples to maintain conservativity. The sampling process exploits graphics hardware for efficiency.

Wonka *et al.* intended for this technique to be applied to urban scenes. A facade of building faces is generated and used to greatly decreases the complexity of the environment. This is necessary for acceptable preprocessing time.

This work has been further extended by Wonka *et al.* [WWS01]. The shrinking factor allows for a conservative visibility estimate to be made within a specified locus (further shrinking implies a greater locus and higher overestimation) of the sample point. Rather than pre-processing by sampling the surfaces of cells/regions, a separate server is used to compute what is visible in the region around the camera. As long as the camera speed is sufficiently bounded, the server can generate visibility information before the camera moves out of the currently evaluated region. The general idea is to amortise the cost of from-region visibility over the several frames, in which the camera must remain in the computed region.

2.4 Ray Space Factorisation

Leyvand *et al.* [LSCO03] observe that the 3D from-region visibility problem is coupled with the intrinsic complexity of the four dimensionality of the space of lines in 3D [MO88]. They decompose the from-region problem into a series of 2D visibility problems resulting from the factorisation of ray space into horizontal and vertical components. These components are then conservatively merged

using an algorithm implemented efficiently using graphics hardware.

The vertical components are treated more conservatively than the horizontal components. This is motivated by the fact that many scenes have low vertical complexity. Where many techniques treat $2\frac{1}{2}D$D visibility solutions, they assume the lowest possible vertical complexity[6] this approach may be extended to handle more general models, with low, but non-trivial vertical complexity.

2.5 Analytic Visibility Techniques

In this section, we discuss the most pervasive ideas in analytic visibility. We recommend that the non-expert reader first read our *geometric preliminaries* chapter (Chapter 3).

It is important to note that although there is a strong relation between the analytic visibility techniques presented in this section and from-region visibility culling, the purpose of these algorithms is to generate a dual structure that can answer *generalised* visibility queries rapidly. E.g., polygon-polygon visibility queries, discontinuity meshing, etc. The cost of this is high run-time and space complexity.

2.5.1 Isotopy Classes and Arrangements

In the context of line-spaces, an *isotopy class* refers to a set of lines. Consider a set of blue lines S and two red lines ℓ_1 and ℓ_2. If it is possible to continuously transform ℓ_1 into ℓ_2 without crossing any blue lines (elements of S), then ℓ_1 and ℓ_2 are said to belong to the same isotopy class.

In a 3D scene, an important property of an isotopy class (defined by extending polygon edges to lines), is that all lines within an isotopy class must intersect the same set of polygons.

Pellegrini [Pel93] presents a structure that can be built as a pre-process that then allows for ray shooting in logarithmic worst case time. The pre-process involves the construction of the arrangement of Plücker dual hyperplanes in the zone of the Plücker hypersurface [APS93].

The 5D cells of this subset of this arrangement are bijective to the set of isotopy classes for the scene. To see this, consider two lines corresponding to two points on the Plücker hypersurface, in the same 5D cell. These two points relate to two lines that both pass all the lines (extended from edges) in the scene in the same way. If one line should cross into an adjacent 5D cell, it would lie on the opposite side of one hyperplane. This is equivalent to crossing the line represented by the inverse dual of the crossed hyperplane. Conversely, should a line cross another line in 3D space, the dual behaviour is to cross a hyperplane in the Plücker arrangement.

[6] They assume that any vertical ray passing through the scene will intersect the surface only once.

By constructing the Plücker arrangement, Pellegerini effectively enumerates the isotopy classes. The intersected triangles for each isotopy classes are found and sorted. A binary search may then be used to find a triangle first intersected by a ray in $O(\log n)$ time. The pre-process takes $O(n^{4+\epsilon})$ time, and is infeasible in practice. Pellegrini [Pel97] has published a comprehensive survey on combinatorial and computational complexity of stabbing and ray-shooting problems using line spaces.

Pellegrini is not the first to enumerate these isotopy classes. McKenna and O'Rourke [MO88] use an (implicit) arrangement of quadric surfaces in four dimensions to represent the isotopy classes. We have discussed the work of Pellegrini in more depth, since we too use Plücker coordinates to generate a subset of isotopy classes.

2.5.2 Visibility Events

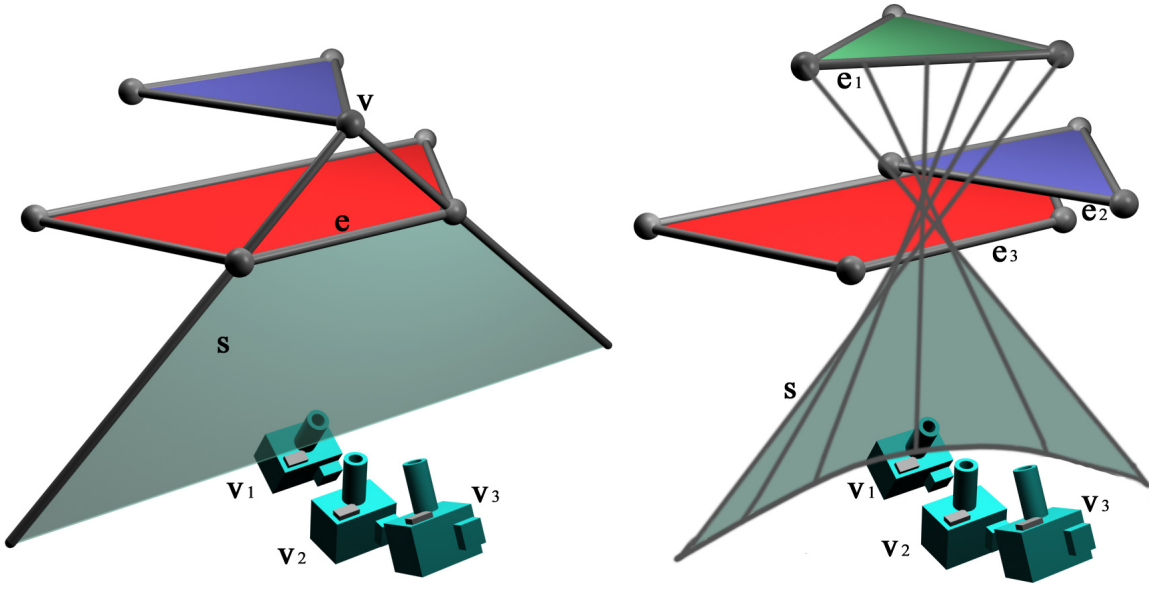
The concept of a *visibility event* [KvD79, PD90] is used by most researchers of analytic visibility. When a camera crosses a particular visibility event, a qualitative change in visibility takes place (e.g., a new vertex, edge or polygon becomes visible or occluded).

It is possible to enumerate all these visibility events. In order to do this, the structure of a visibility event surface needs to be understood. A visibility event surface has an associated event that will occur when crossed. Every visibility event has an associated event surface. It is possible for multiple events to occur when a single surface is crossed.

A visibility event surface is defined by the interactions of edges and vertices within the scene. Without loss of generality, we assume all scene polygons to be convex. A *vertex-edge* (VE) event surface is defined by a vertex of a polygon and the edge of another distinct polygon. The surface is a wedge in the plane of the vertex and the edge. When the event surface is crossed, the polygon containing the vertex becomes visible or invisible. This is depicted in Figure 5a. An *edge-edge-edge*(EEE) event is defined by three edges. Three edges in space define a ruled quadric surface. When this surface is crossed, an edge may become visible or invisible. This is depicted in Figure 5b.

These events are strongly related to the "lines through four lines" problem (see Section 3.5.2). It is noted that there are exactly two lines determined by four lines (in general position). By relaxing the four line constraint to three lines, the set of incident lines becomes a ruled quadric. Should two of these lines intersect (i.e., at a "vertex"), then this surface degenerates into a planar wedge. A representation for the general quadric event surface (also referred to as an event *swath*) can be computed explicitly [Pla92].

Truncating these lines to edges results involve a clipping of the original surfaces. It is important to observe that a vertex is always defined by the intersection of two lines (i.e., a vertex is always



(a) (b)

Figure 5: *VE and EEE events.* (a) A vertex-edge(VE) event. At v_1 the camera cannot see neither vertex v nor the triangle associated with it. At v_2, vertex v becomes visible over edge e (v_2 is on the event swath s). At v_3 vertex v and the triangle associated with it is visible. (b) An edge-edge-edge(EEE) event. At v_1 the camera cannot see the triangle associated with e_1. At v_2 (on the event swatch) a small piece of e_1 becomes visible over the (view apparent) intersection point of e_2 and e_3. At v_3, the triangle associated with e_1 is visible.

the meeting point of two adjacent edges). When considering an EV event, the planar wedge is bounded by the vertices on either sides of the edge. At these boundaries, there is an interaction of four edges/lines (two edges defined by the original VE vertex, and another defined by an edge terminus). Returning the "lines through four lines" problem, we note that the four edges define a single line at each boundary. Such lines are known as *extremal stabbing lines*. Depending on the type of edge/vertex interactions, these lines may be referred to as VV, VEE or EEEE lines.

In theory, the number of VE events is $O(n^2)$, since each vertex and each edge (linear in the number of vertices) can be paired. Similarly, there can be as many as $O(n^3)$ EEE events. Finally, there can be as many as $O(n^4)$ extremal stabbing lines, since they are potentially generated by any combination of four edges. These combinatorial bounds define a worst case of high complexity for any algorithm attempting to enumerate visibility events, however these bounds are pessimistic in practice [DDP97b]. For further discussion and illustration, we recommend the PhD. thesis of Durand [Dur99].

Returning to the Plücker hyperplane arrangement of Pellegrini [Pel93], we relate the concept of visibility events to this arrangement. Three edges define an event surface. These three edges correspond to three lines (embedding the edges). These three lines may be mapped to Plücker hyperplanes. When intersected, these hyperplanes define a 2D surface in $\mathbb{P}^5$. This 2D surface can then be intersected with the Plücker hypersurface to give a 1D *trace* [Tel92b, Tel92a] on the hypersurface. This 1D trace maps back to a set of lines in $\mathbb{R}^3$. This set of lines defines a ruled quadric. The event surface is embedded within this quadric. In order to find the correctly clipped surface, terminal points need to be found for each end of the trace. These points can be found by finding the extremal stabbing lines and mapping them to $\mathbb{P}^5$.

Relating the complexity of arrangements to the number of extremal stabbing lines, we note that the Plücker hyperplane arrangement can be constructed in $O(n^4 \log n)$ time that is only slightly greater than the combinatorial worst case of $O(n^4)$. At present, the visibility complex provides a somewhat better solution, since it supports output sensitive construction.

2.5.3 The Visibility Complex

In Section 2.5.1 we define an isotopy classes to be the set of lines that pass each line in another set of lines in the same way (i.e., they have consistent orientation). The visibility complex [PV93, DDP96, DDP97c, DDP97a, Dur99, DDP02] is a similar structure to that of the Plücker hyperplane arrangement, however the sets that it tries to group are not lines, but *maximal free segments*. These are subsegments of those lines in each isotopy class that *see* the same geometry (i.e., they share the same terminus polygons). In order to account for this segmented view of line space, a discrete "pseudo-dimension" is defined. Effectively, this associates a list of segments with each point in the dual line space. The definition of the visibility complex supports both piecewise linear and smooth objects.

Durand *et al.* [DDP96, DDP96, DDP02] describes the 3D visibility complex in terms of a 4D parameterisation of line space. In particular a parameterisation in terms of spherical angles (for direction) and two other coordinates (translations in the plane through the origin, whose normal is the direction associated with the spherical angles). As noted by Durand, the particular parameterisation is of little import, but is used for illustration purposes.

Durand *et al.* give an output sensitive (hyper)plane sweep algorithm to construct the 3D visibility complex. The algorithm is of order $O((k+n^3)\log n)$ for n, where n is the complexity of the scene, and k is the number[7] of extremal stabbing lines. To date, the visibility complex has not been

[7] k can be of $O(n^4)$, but this is unlikely in practice.

implemented due to its extreme sensitivity to small error, and difficulty of implementation. For practical purposes, a simplification of the visibility complex has been developed. This is known as the *visibility skeleton*.

2.5.4 The Visibility Skeleton

The set of zero and one dimensional faces (vertices and edges) of a polyhedral structure is known as the *skeleton*. The visibility skeleton [DDP97b, Dur99] is the skeleton of the 3D visibility complex.

Each zero dimensional face corresponds to an extremal stabbing line, and each one dimensional face corresponds to a swath of lines which define an event surface. A skeleton is naturally represented as a graph. In terms of visibility events and surfaces, one can observe that each event surface terminates at extremal stabbing lines. Also, many event surfaces may terminate on the same extremal stabbing line. Dually, this is equivalent to many edges (of the skeleton) meeting at a single vertex (of the skeleton).

Durand *et al.* give a *reasonably* robust algorithm that builds the visibility skeleton. The algorithm is somewhat different to the sweep algorithm used to build the complex. The zero and one dimensional faces are built directly, through a combinatorial selection of edges/vertices and ray-shooting. This algorithm has been implemented, and has been used to drive various visibility dependent techniques such as discontinuity meshing [Dur99] and hierarchical radiosity [DDP99, DDP98].

Duguet and Drettakis [DD02] observe that there are, however, certain cases where the construction of the visibility skeleton will fail. Using epsilon geometry [SSG89] Duguet and Drettakis implement a robust version of the visibility skeleton.

Ultimately, the visibility skeleton represents all relevant event surfaces and extremal stabbing lines. However, there is certain topological information lost through the omission of the higher dimensional elements of the complex, that makes is less suited for from-region visibility in practice. We discuss this in context in Section 6.1.3.

2.5.5 The Aspect Graph

The *aspect graph* is a structure developed in the field of computer vision [KvD79, KvD76]. The key idea is to determine all possible aspects/views of an object (or a database of objects). From an image, edge features may be extracted in order to determine a *view*. These views are then matched up with the computed aspects. If a match is found, then the associated object is “recognised”.

The nodes of the aspect graph correspond to aspects. The arcs between the nodes correspond to

visual events that change the aspect. The graph is a dual structure, much akin to the visibility skeleton. One important difference, however, is that the aspect graph is *view centered*. This introduces additional complexity, as events relating to all possible positions (usually restricted to positions outside of the object's convex hull) of an orthographic or perspective camera are also incorporated. The size of the aspect graph can be as large as $O(n^6)$ for the case of an orthographic camera, while for a perspective camera, the aspect graph may have complexity $O(n^9)$. It should be noted, that a change in aspect does not necessarily imply a change in the set of visible polygons. This set is a much smaller subset of the aspect graph.

Gigus *et al.* [GM90, GCS91] detail an algorithm for computing the aspect graph. Plantinga [Pla98] and Plantinga and Dyer [PD90, PD87] present another algorithm for building the aspect graph. Their algorithm uses an auxiliary structure known as the *ASP* that defines the set of lines through a polygon. The ASPs of two polygons can be composed and subtracted to account for occlusion of one by the other. This is a concept used extensively in Chapters 5 and 6.

An extensive discussion of the aspect graph literature (in the context of visibility) can be found in the survey by Durand [Dur99].

2.5.6 Polygon Stabbing and Anti-penumbra Computation

Given a set of polygons S, the general polygon stabbing problem seeks to determine the *existence* of any straight line s that intersects all polygons in S. For the general case it is possible to determine this in $O(n^4\alpha(n))$ time [MO88][8].

By making assumptions about S, it is possible to find lower bounds. For stabbing a set of axially aligned (isothetic) rectangles, a deterministic algorithm of $O(n \log n)$ exists [HT92]. A randomised algorithm with an expected running time of $O(n)$ has also been found [Ame92].

Pellegrini [Pel90] gives an $O(g^2n^2 \log n)$ algorithm for n triangles with of g distinct normals. Teller and Hohmeyer [TH93b] give an algorithm for the case where S consists of oriented polygons. The algorithm runs in $O(n^2)$ expected time.

This algorithms builds a representation of the set of lines stabbing S, by applying constraints in Plücker line space. The solution space is represented as a dual polyhedron. If this polyhedron intersects the Plücker hypersurface, a stabbing line must exist. This stabbing solution is used to find portal sequences for from-region visibility [Tel92a]. The anti-penumbra can also be extracted from a sequence of such polytopes [Tel92b]

[8] Where $\alpha(n)$ is the functional inverse of Ackermann's function.

Further stabbing results may be found in the survey by Pellegrini [Pel97]. We revisit the algorithm of Teller and Hohmeyer in Chapter 5, since it forms an integral part of our solution to the general selective stabbing problem.

2.5.7 Discontinuity Meshing

Discontinuity meshing is a technique for accurately representing the illumination discontinuities on a surface. For area light-sources, discontinuities in the first and second derivatives can exist.

Reversing our light-region analogy, let us consider the view of a light source from the perspective of a point moving along a line embedded in a receiving surface. Where the whole light source is visible, the surface is fully lit. When an object appears to move in front of the light source, the surface falls into penumbra and discontinuity occurs. Depending on the configuration of the occluding geometry and the light source, illumination will begin to change at a constant or linear rate. This determines whether or not the discontinuity is of the first or second order. When the light source appears to disappear behind one or more occluders, the surface point falls into umbra.

The interaction between blockers and a light source are visibility events. When relating this to a receiving surface of a light source, this implies that all discontinuities can be associated with he intersection of the receiving surface and the event surfaces defined by edges of the light source and associated blockers. By partitioning the target surface into a mesh along these intersections, the discontinuity mesh is computed.

In order to determine the associated luminance with each mesh element and the discontinuities associated with each edge of the mesh, the *back projection* of the light source is computed for each mesh element[9]. The back projection is in essence the *aspect* of the light source from a viewpoint on the receiving mesh. The aspects are topologically indistinct for viewpoints within a discontinuity mesh element.

Heckbert [Hec92] introduced discontinuity meshing in the context of radiosity[10], as a means to accurately represent shadow boundaries. Stewart and Ghali [SG93, SG94] solve a slightly different problem to that of Heckbert, but they also treat EEE events. Drettakis and Fiume [DF94] present a similar solution that achieves better performance in practice, but does not have asymptotic bounds that are as tight as those of Stewart and Ghali. Durand *et al.* [DDP99] apply global visibility to hierarchical radiosity to achieve faster and/or more accurate results. Dugeuet and Drettakis [DD02]

[9] The back projection can be maintained incrementally, allowing for efficient implementation.

[10] At the time, other authors had developed discontinuity meshing for point light sources, and for a significantly smaller subset of events for area light sources.

present an algorithm that computes shadow boundaries robustly, using the principles of epsilon geometry [SSG89].

2.5.8 Analytic Visibility in Practice

The high (four) dimensionality of line space in 3D has made the analytic solution to from-region visibility determination a difficult task[11]. Some authors have turned to a simplified version of the problem, namely visibility in 2D. Since the space of lines in 2D is effectively two dimensional, 2D problems are not as combinatorially explosive as their 3D counterparts. See Ghosh [AGS00] for an excellent survey of the extensive literature on in-plane visibility.

The motivation for solving for 2D from-region visibility, is that many real scenes are fundamentally 2D or $2\frac{1}{2}D$ (i.e., height fields) in nature.

Koltun *et al.* [KCCO01] 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 a conservative discretisation through rendering hardware. Independently, Bittner *et al.* [BWW01] also developed an exact visibility solution in the plane, based on similar principles.

Plantinga [Pla93] considers an application of the aspect graph to achieve rapid occlusion culling. Only a subset of interactions are accounted for and only a very limited camera model has been implemented (rotation around one axis).

The Plücker Space Occlusion Tree

Shortly after the publication of our exact algorithm [NBG02], Jiří Bittner published an alternative solution in his doctoral thesis [Bit02]. The approach is similar to that taken by Pu [Pu98], in that a BSP in Plücker space is the primary data structure. Pu empirically evaluates the average case complexity of his approach to be $O(n^{4.31343})$[12]. Pu's algorithm is not attempted on scenes larger than 15 triangles, due to the excessive computational and memory requirements.

Where Pu gives an intractable algorithm to compute a *global* visibility map, Bittner presents various techniques to localise visibility computations, yielding a considerably more practical solution. Visibility is computed from a region, thus constraining the line set to only those exiting a polygonal region. Furthermore, the set of directions exiting such a region is partitioned so that each subdivision (shaft) can be treated separately. This provides better memory management.

[11] With the notable exception of Teller [Tel92a], although even here the scene requires special structure.

[12] Pu believes this to be related to the $O(n^4\alpha(n))$ bound of McKenna and O'Rourke [MO88].

For each shaft, the algorithm proceeds by processing the polygons in a front to back order (or *occlusion sweep*). As each polygon is processed, it is tested for visibility against the BSP as follows: At any one time, the BSP represents the set of lines blocked thus far, and by which polygon each line (or rather, isotopy class) is blocked. A polygon is discarded if it is determined to be invisible (i.e., all lines between the source region and the polygon are already blocked by polygons inserted before it). Should the polygon be visible, then it (or rather, the Plücker polytope representing the set of lines that go through both the source region and the polygon) is merged into the BSP tree.

The merging process requires a 5-dimensional polytope splitting operation. Such algorithms exist (see Section 3.3), however, due to implementation problems, a less efficient algorithm is used[13].

This algorithm performs a local construction of the 3D visibility complex. Only those complex elements representing the maximal free segments with one terminus on the source region are constructed[14].

Since this algorithm enumerates all the isotopy classes of lines going through a region, its performance is tightly coupled with the combinatorial complexity of visibility events in 3D line space. Bittner notes that the size of the occlusion tree is bounded above by $O(n^5)$ and that the worst case running time of the algorithm is $O(n^4 \log n)$.

We revisit and contrast our algorithm to that of Bittners in Section 6.2.4.

2.6 Compression of Visibility Information

All visibility pre-processing systems result in a block of data organised as some form of spatial structure. In order for culling to take place, this structure (or at least part of it) needs to be in memory. Typically, this structure also has to be stored on disk, although it might also have to be transmitted over a network [COZ98].

Visibility data may become large. It is therefore desirable to compress this visibility data. In order to achieve this, both *lossy* and *lossless* techniques have been proposed by Cohen-or *et al.* [COFHZ98], Nadler *et al.* [NFLYCO99], Gotsman *et al.* [GSF99] and Yagel and Ray [YR96]. Panne and Stewart [vdPS99] are the first to focus on this problem and present both a lossy and a lossless compression algorithm.

In the context of visibility compression, the term lossy refers to increasing the conservativity of the visibility solution in order to save space. Panne and Stewart achieve this by using the union of

[13]The H-Representation (see Section 3.2) of the polyhedron is augmented with the splitting plane. Two vertex enumerations (see Section 3.2.2)) are performed: one with this plane positively oriented and one with it negatively oriented.

[14]To be precise, even this may be subdivided due to the partition of directional space

similar visibility sets as an aggregate visibility set. Effectively, the visibility sets of multiple existing cells are collapsed into a single larger one. Panne and Stewart perform this implicitly by using a table representation. The table consists of boolean values relating each cell to each primitive. Yagel and Ray suggest a similar compression algorithm, however the table structure of Panne and Stewart allows for the visibility sets of disjoint cells to be combined where beneficial.

Lossless compression implies that the visibility structure is transformed such that less space is required to store it without the introduction of additional conservativity. To achieve this Panne and Stewart search for coherence in their table. When a cluster of cells have similar visibility, their similarities are grouped together and stored once. Similarly, if a cluster of polygons have the same visibility status, they too are grouped together and stored once.

Note that lossy compression implies the creation of a refined structure that is later collapsed into a coarse structure. As we shall see in Section 4.2.2, unlike Panne and Stewarts bottom-up approach, our top-down approach can obviate this redundancy. Panne and Stewart propose a "top-down, divide-and-conquer" extension to their work that allows for the pre-processing of larger models. Although developed independently, our work (Section 4.2) represents an efficient solution to their proposed problem. We discuss our methods for visibility compression in Appendix B.

Chapter 3

Geometric Preliminaries

In this chapter we cover a collection of mathematical tools that are used to manipulate lines in space efficiently. We also discuss various issues relating to the representation of polyhedra of general dimension. A selection of mathematical topics that are relevant to this dissertation, are presented at the end of this chapter.

3.1 Projective Spaces

We briefly discuss *projective spaces* and motivate their use. We also cover the concept of *projective duality*. Most importantly, we present all the requisite information on *Plücker* coordinate systems, a fundamental tool used throughout this dissertation.

3.1.1 The Classic Projective Plane

Ostensibly, the standard Euclidean spaces are consistent with intuition. However, in a sense they are also incomplete. Consider two lines in $\mathbb{R}^2$. These lines necessarily intersect, with the *exception* of parallel lines. Projective geometries seek a uniform methodology for handling such exceptions. They attempt to construct a space, $\mathbb{P}^2$, where such exceptions do not occur.

Unlike $\mathbb{R}^2$, projective spaces cannot be represented explicitly, but rather implicitly through a projection operation. Points in a d dimensional projective space are typically represented using some form of *model* [Sto91]. In his book, Stolfi presents four models, they are the *straight*, the *spherical*, the *analytic* and the *vector space* models. The differences between these models are purely superficial, and they can be equated through isomorphism.

We choose a natural hybrid of the analytic and vector space models to give an intuitive description of the classic projective plane $\mathbb{P}^2$. A "point" in this space is represented by a homogenous 3-tuple, (x, y, w). 3-tuples usually represent points in $\mathbb{R}^3$, however, in $\mathbb{P}^2$, all 3-tuples refer to an equivalence class, $< x, y, w >$, where any point $(x, y, w) \equiv (\lambda x, \lambda y, \lambda w)$ for any $\lambda \in \mathbb{R}$, where $\lambda \neq 0$. The principal idea is that the points of $\mathbb{P}^2$, are exactly the set of such equivalence classes in $\mathbb{R}^3$. Equivalently, $\mathbb{P}^2$ can be said to be the set of all one dimensional vector spaces of $\mathbb{R}^3$.

To see how the classic projective plane corresponds to the standard plane, $\mathbb{R}^2$, an arbitrary plane in $\mathbb{R}^3$ that does not contain the origin is chosen. Without loss of generality, we choose the plane $w = 1$. We observe that each point on $w = 1$, can trivially be mapped to a unique point in $\mathbb{R}^2$. Namely, a point $(x, y, 1) \in \mathbb{R}^3$ may be mapped to $(x, y) \in \mathbb{R}^2$. This also shows that each point $< x, y, 1 > \in \mathbb{P}^2$ may be mapped to a unique point in $\mathbb{R}^2$ and vice versa.

However, there are points in, $\mathbb{P}^2$ that do not correspond to points in $\mathbb{R}^2$. With respect to our chosen plane $w = 1$, these points correspond to those equivalence classes of the form $< x, y, 0 >$. These points are referred to as *points at infinity* or *ideal points*. We prefer to use the latter terminology, since the former is intuitive only for certain models of $\mathbb{P}^2$.

As a natural extension of points, lines in $\mathbb{P}^2$ correspond to two dimensional vector spaces, or equivalently, planes containing the origin in $\mathbb{R}^3$. Where such a plane intersects $w = 1$, a one-dimensional set is formed. This corresponds to a unique line in $\mathbb{R}^2$. Similarly, every line in $\mathbb{R}^2$ corresponds to a unique plane going through the origin in $\mathbb{R}^3$, and hence to a line in $\mathbb{P}^2$.

There is, however, *one* special line that exists in $\mathbb{P}^2$, but not in $\mathbb{R}^2$, this is the *ideal* line, and with our choice of projection plane, corresponds to the plane $w = 0$. This line is exactly the set of ideal points. This ideal line is the sole difference between $\mathbb{R}^2$ and $\mathbb{P}^2$. We will now consider an example that illustrates how this augmentation of $\mathbb{R}^2$ aids our motivation. Let us choose any two distinct lines in $\mathbb{P}^2$. These lines correspond to planes containing the origin in $\mathbb{R}^3$. The intersection of the two lines in $\mathbb{R}^2$ is isometric to the homogenous point that is the intersection of the corresponding planes in $\mathbb{R}^3$. However, since, *all* distinct planes that go through the origin intersect in a line going through the origin, *all* lines in $\mathbb{P}^2$ intersect at a point in $\mathbb{P}^2$.

Next we give a concrete example showing how the case of parallel lines are handled. Consider two lines, $y = 1$ and $y = 2$ in $\mathbb{R}^2$. These lines are both horizontal and do not intersect. In $\mathbb{P}^2$, these lines correspond to the planes $y - w = 0$ (when $w = 1$, then $y = 1$), and $y - 2w = 0$ (when $w = 1$, then $y = 2$) respectively. The intersection of these planes is the vector space corresponding to $< 1, 0, 0 > \in \mathbb{P}^2$. In general, any two parallel lines will intersect at an ideal point in $\mathbb{P}^2$.

3.1.2 Oriented Projective Geometry

The focus of [Sto91] is on *oriented* projective geometry. Whereas the classic projective plane is an augmentation of $\mathbb{R}^2$ by an ideal hyperplane, the oriented projective plane maintains *two* copies of the plane, one with positive, and one with negative orientation, as well as an ideal hyperplane.

This allows the association of an orientation with geometric structures and is necessary to define properties such as convexity to remove certain ambiguities. Even more fundamentally, the Jordan Curve theorem does not hold in the classic projective plane, but does hold in oriented projective geometries.

3.1.3 d dimensional Projective Spaces

Both the classic and the oriented projective planes may be generalised to d dimensions. The homogeneous coordinates for $\mathbb{P}^d$ are always represnted as points in $\mathbb{R}^{d+1}$. A k dimensional affine subspace in $\mathbb{P}^d$ may be represented as a $k+1$ dimensional vectorspace in $\mathbb{R}^{d+1}$. Also, the projective hyperplane is always an arbitrary, but fixed, hyperplane in $\mathbb{R}^{d+1}$. A d dimensional projective space always contains exactly one ideal hyperplane.

3.1.4 Projective Duality

A *duality mapping* or *duomorphism* is an isomorphism between structures in one space and other structures in the same or another space. Typically, a duality mapping preserves certain *dualistic* properties.

For example, consider a line in $\mathbb{R}^2$. We can define a duality mapping $d(\ell)$ that takes a (non-vertical) *line* ℓ of the form $\ell := \{(x, y) \in \mathbb{R}^2 : y = ax + b\}$ to the point $(a, b) \in \mathbb{R}^2$. Similary we can define $d^{-1}(a, b)$ that takes a point $(a, b) \in \mathbb{R}^2$ to the line defined by $y = ax + b$.

d and d^{-1} have the property that they *preserve incidence*. The duals of all points on a given line all intersect at a single point. The duals of all lines incident at a particular point must lie on the same line in the dual space.

Consider any two distinct points points $(a, b) \in \mathbb{R}^2$ and $(c, d) \in \mathbb{R}^2$. Since these points are distinct they both lie on some unique line. Now let us consider the duals $d^{-1}(a, b) := \{(x, y) \in \mathbb{R}^2 : y = ax + b\}$ and $d^{-1}(c, d) := \{(x, y) \in \mathbb{R}^2 : y = cx + d\}$. These lines are incident where $ax + b = cx + d \Rightarrow x = \frac{d-b}{a-c}$ and $y = a\frac{d-b}{a-c} + b$, thereby giving the point of incidence: $(\frac{d-b}{a-c}, a\frac{d-b}{a-c} + b)$. Now let us consider a third point (e, f) on the line generated by (a, b) and (c, d). Any point on this line may be expressed in terms of (a, b) and (c, d), namely $(e, f) = (\lambda(c -$

$a) + a, \lambda(d - b) + b)$ for some fixed $\lambda \in \mathbb{R}$. Next we consider the dual of (e, f). $d^{-1}(e, f) := \{(x, y) \in \mathbb{R}^2 : y = (\lambda(c-a)+a)x + \lambda(d-b)+b\}$. This and the line $d^{-1}(a, b)$ are incident where $ax+b = (\lambda(c-a)-a)x+\lambda(d-b)+b \Rightarrow \lambda(ax-cx) = \lambda(d-b) \Rightarrow x = \frac{d-b}{a-c}$ and $y = a\frac{d-b}{a-c}+b$, thereby giving the point of incidence: $(\frac{d-b}{a-c}, a\frac{d-b}{a-c}+b)$, that is precisely the same point of incidence as $d^{-1}(a, b)$ and $d^{-1}(c, d)$.

The example of duomorphism d illustrates point-line duality. There are numerous advantages to such duality. Using such a mapping it is possible to prove a geometric theorem in some *primal* (original) space, and then have a dual theorem proved in a *dual* space (an image of the original space under a duality mapping). A formal approach to this is described by Stolfi [Sto91]. This feature translates into the ability to create *dual algorithms*. The principal advantage being that certain theorems and algorithms are more intuitive in the dual space. In terms of implementation, it is possible to use an existing solution to an algorithm to solve the algorithm's dual problem, thereby saving on implementation difficulties.

As an example, let us consider the halfspace intersection problem: Given a set of halfspaces in $\mathbb{R}^2$, find the convex polygon representing their intersection. Each halfspace is represented by an oriented line. The halfspace intersection problem dualises to the convex hull problem. By transforming the oriented lines to points in a dual-space, the convex hull of these points can then be computed using an existing code base. The lines embedding the edges of the convex hull can then be dualised to points in the primal plane. These points are exactly the vertices of the polygon resulting from the halfspace intersection. There is one additional subtlety: the duality mapping is defined in terms of an arbitrary point whose only condition is that it falls in the interior of the halfspace intersection. For details on this algorithm we refer the reader to Edelsbrunner [Ede87] and Preparata and Shamos [PS85].

The duality mapping we have described so far has one serious flaw: it does not handle *all* lines in $\mathbb{R}^2$. Vertical lines are not treated since they cannot be represented by functions of the form $y = mx + h$. To account for this, an alternative representation for lines is imposed. We use the equation form $ax + by + c = 0$. This form can represent *any* line in $\mathbb{R}^2$. We note that since the line is an equation, any non-zero constant multiplication of all a, b and c represents the same line. This is no coincidence. The space of all lines is duomorphic to the projective plane. The most obvious mapping is to that of a homogenous coordinate system.

Consider a mapping $d'(\ell)$ that takes *any* line in $\mathbb{R}^2$ of the form $\ell := \{(x, y) \in \mathbb{R}^2 : ax+by+c = 0\}$ to the point $< a, b, c > \in \mathbb{P}^2$. The same properties are preserved, and no exceptions result from this mapping.

Duality can be generalised to $\mathbb{R}^d$. It is always possible to define a duomorphism between the points in $\mathbb{P}^d$ and the hyperplanes of $\mathbb{R}^{d+1}$.

3.1.5 Plücker space

Plücker space [Plü65] is a special case of a Grassmann coordinate system [Som59, CEG$^+$96, Sto91]. 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:

$$\begin{aligned} \pi_0 &= q_x - p_x & \pi_1 &= q_y - p_y & \pi_2 &= q_z - p_z \\ \pi_3 &= q_z p_y - q_y p_z & \pi_4 &= q_x p_z - q_z p_x & \pi_5 &= q_y p_x - q_x p_y \end{aligned}$$

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[1] 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 6 for an illustration.

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 following set of points:

$$G = \{D_x(x) = 0 : x \in \mathbb{P}^5\} \setminus \{\mathbf{0}\} \tag{3}$$

Since, at least for the purposes of this dissertation, we are predominantly interested in real lines, this surface is used extensively.

[1] Although, this in itself is not an inner product. See Lemma A.1 in Appendix A.

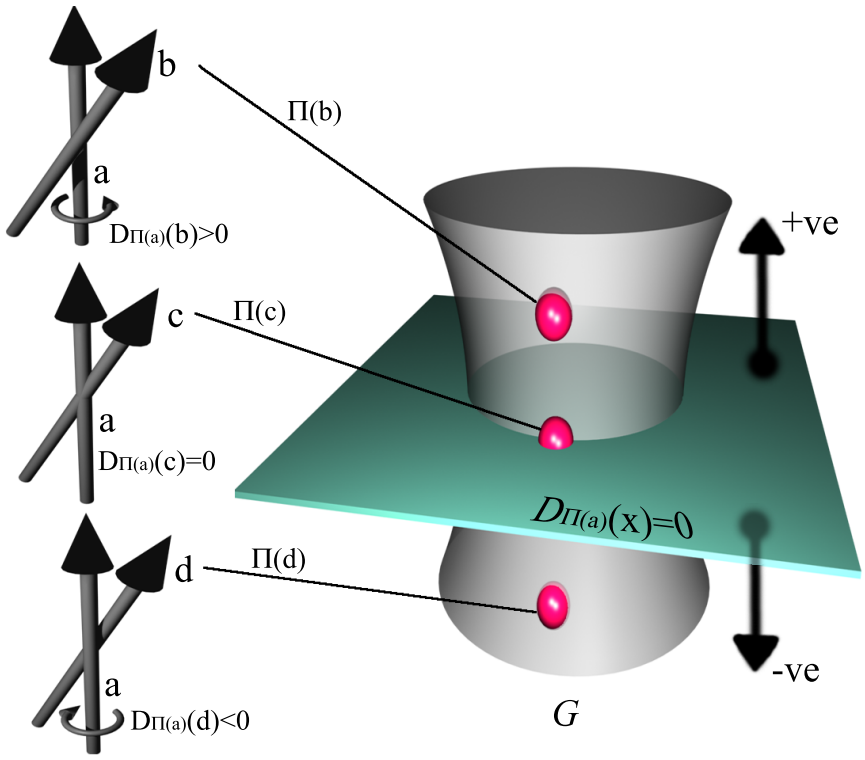


Figure 6: *Line Orientation and Plücker space*. The three diagrams on the left show the three qualitatively different ways for one directed line to pass another. The figure on the right is a visualisation of how these lines relate in the dual Plücker space. The surface G is a visualisation of the Plücker 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 $D_{\Pi(a)}(x) = 0$.

3.2 d-Dimensional Polytope Representation

There are many conflicting definitions for the terms *polytope* and *polyhedron*. To avoid confusion, we define the following:

Definition 3.1 *Given k distinct points $p_1, p_2, \ldots, p_k$ in $\mathbb{R}^d$, the set of points $p = \alpha_1 p_1 + \alpha_2 p_2 + \ldots + \alpha_k p_k, (\alpha_j \in \mathbb{R}, \alpha_1 + \alpha_2 + \ldots + \alpha_k = 1)$ is the* affine set *generated by $p_1, p_2, \ldots, p_k$, and p is an* affine combination *of $p_1, p_2, \ldots, p_k$.*

Definition 3.2 *Given a subset L of $\mathbb{R}^d$, the* affine hull *of L is the smallest affine set containing L.*

Definition 3.3 *Given k points $p_1, p_2, \ldots, p_k$, they are said to be* affinely independent *if the $k-1$ vectors $p_2 - p_1, \ldots, p_k - p_1$ are linearly independent. An affine hull generated from k affinely*

independent points is said to be of dimension $k-1$.

Definition 3.4 *A* polyhedron *is a connected region of space consisting of a piecewise linear boundary. It is said to be of* d *dimensions if its affine hull is of* d *dimensions.*

Definition 3.5 *A* polytope *of* d *dimensions is a convex, bounded,* d *dimensional polyhedron.*

Definition 3.6 *The boundary of a polyhedral set consists of* faces. *Faces are lower dimensional convex polyhedra. A* k-face *is a face of dimension* k*. We further define a* d *dimensional polyhedron to have exactly one* d*-face, which is itself. We also define a* (-1)-face *to be incident on everything.*

Definition 3.7 *A* facet *of a* d *dimensional polyhedron* L *is a* $d-1$*-face of* L.

Polytopes of dimensions 0, 1, 2 and 3 are familiar constructs. They correspond to vertices, segments, bounded polygons and 3 dimensional polytopes. There are two distinct ways to represent polytopes. It is sufficient to represent a polytope as a set of vertices, where the vertices are chosen such that their convex hull is the original polytope. Every polytope may be decomposed into such a set.

Definition 3.8 *The* extreme points *of a polytope* L *are those points belonging to the smallest set of vertices whose convex hull gives* L.

The set of extreme points is always finite. This set is a compact representation of a polytope. Such a representation is possible due to the convexity constraint. This representation is referred to as follows:

Definition 3.9 *The* V-representation *(or Vertex representation) of a polytope is its set of extreme points.*

Another common representation is:

Definition 3.10 *The* H-representation *(or Halfspace representation) of a polytope is a set of halfspaces whose intersection gives the polytope.*

It should be noted that the H-representation can also represent unbounded convex polyhedra. Once again we see an important duality. One representation is through points, while the other is through halfspaces (defined by hyperplanes). In the case of polytopes, it is possible to switch between representations using a transformation algorithm [AF92, FP96, AF96].

3.2.1 The Face Lattice

The face lattice is a directed graph representing the incidence relationship between the faces of a polytope. Each face of the polytope is a node of the lattice. If a face of dimension k is contained within a $k+1$ dimensional face, a directed arc exists from the k dimensional face to the $k+1$ dimensional face (see Figure 7 for an example).

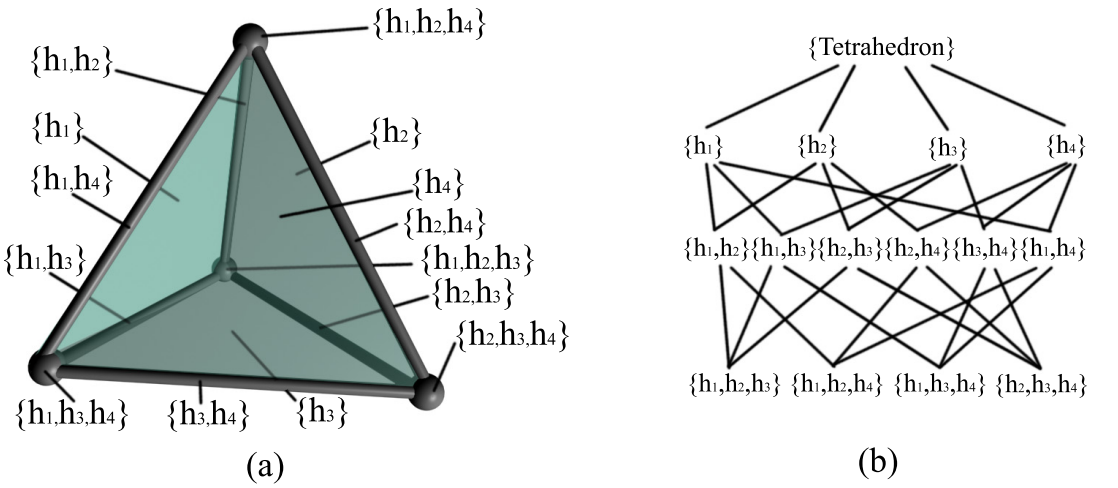


Figure 7: *Face Lattice of a Tetrahedron*. (a) A tetrahedron, with annotated faces. The tetrahedron is defined by four planes, h_1, h_2, h_3 and h_4. Each face is identified, by the subset of planes in which it is embedded. (b) The face lattice of the tetrahedron. The faces are connected by an upwards containment relationship. The bottom row refers to 0 dimensional elements, the row above to 1 dimensional elements, etc. Note how a row of k dimensional elements is defined by the intersection of $3-k$ planes.

Similarly, Face lattices may be defined for general polyhedra and polytope/polyhedral *complexes* (see Section 3.3).

3.2.2 Face Enumeration

For all polytopes, the face lattice is implicit within its vertex information. This is due to the known property of convexity. Given the set of extreme points for any polytope, the face lattice may be recovered. This process is known as *face enumeration* [FR94].

It is possible to define any vertex in a d dimensional space as the intersection of a set of hyperplanes. Algebraically, a set of hyperplanes may be considered as a system of linear equation, where the vertex at the intersection is the solution to such a system. In order to define a unique solution (a vertex) the hyperplanes need to define at least d linearly independent constraints.

Definition 3.11 *A* simple *polytope of d dimensions embedded in a d dimensional space is a d dimensional polytope whose vertices are incident on exactly d hyperplanes (facets) of it's H-representation.*

Given the set of extreme points E of a d dimensional simple polytope P, it is possible to find a set of oriented hyperplanes H whose associated half space intersections results in the convex hull of E (see Section 3.2).

Each extreme vertex is therefore incident on a subset of H, of cardinality d. If we enumerate each hyperplane, we can represent each vertex uniquely by the subset of H on which it is incident. We refer to this subset as the *enumeration set* of the vertex. See Figure 7 for an example.

Through vertex enumeration, we see that it is possible to enumerate *every* face as the intersection of a set of hyperplanes. Indeed, every k dimensional face is the intersection of $d-k$ hyperplanes in H. From this, we observe that in general it is possible to enumerate every k dimensional face as a subset of hyperplanes in H of cardinality $d-k$.

This definition of face enumeration supports certain properties:

1. Given two faces f_1 and f_2 (of dimensionality k), these faces are incident at a face f_3 (of dimensionality $k-1$) , whose enumeration set is the union of enumeration sets f_1 and f_2.

2. The intersection of the enumeration sets of two faces f_1 and f_2 gives the enumeration set of the face of lowest dimensionality containing both f_1 and f_2.

As an example, consider Figure 7. Facets $\{h_3\}$ and $\{h_2\}$ intersect at $\{h_3, h_2\}$. Similarly, edges $\{h_3, h_2\}$ and $\{h_3, h_4\}$ intersect at vertex $\{h_3, h_2, h_4\}$. Should the face resulting from the union not belong to the polytope, then no intersection exists.

To illustrate the second property, consider edges $\{h_3, h_4\}$ and $\{h_2, h_3\}$, these intersect at facet $\{h_3\}$. Consider also, vertex $\{h_2, h_3, h_4\}$ and edge $\{h_1, h_3\}$. These intersect at $\{h_3\}$. Should edge $\{h_3, h_4\}$ be used instead, the intersection is the same edge $\{h_3, h_4\}$. The latter results in a two dimensional face, since the vertex lies on the edge. If the intersection is the empty set, then the largest face containing the specified faces is the polytope itself.

3.3 Splitting a Polytope Complex in d Dimensions

Definition 3.12 *A* polytope complex *of dimension k is a set of k dimensional polytopes that intersect only where they share sub-faces. Additionally, each sub-face of any of these polytopes is considered to be part of the polytope complex.*

Definition 3.13 *A* top level polytope *is a d dimensional polytope of a d dimensional polytope complex.*

A Polytope complex is useful in that it allows for the representation of a bounded non-convex polyhedron, as the union of a set of polytopes. Using the properties of polytopes that we have discussed in earlier sections, we observe that a polytope complex may be used to represent general d dimensional non-convex bounded polyhedra.

The face lattice of a polytope complex may be defined similarly to that of general polytopes. The difference being that polytopes that share boundary faces, will share the associated nodes in the face lattice.

Bajaj and Pascucci [BP96] present an algorithm that *splits* a polytope complex with a hyperplane. Splitting in this context implies that each polytope of the complex that is incident on the hyperplane is split in two, such that the new facets of each new polytope are embedded within the splitting hyperplane. A figure depicting the desired changes in the face lattice is shown in Figure 8.

This operation does not alter the set of points represented by the complex, only it's structure. Such alteration facilitates certain common operations, such as CSG (constructive solid geometry) and clipping (a special case of CSG).

For the readers convenience, we provide the algorithm of Bajaj and Pascucci verbatim in Figure 9.

This algorithm requires that every vertex be classified (as below($\boxed{-}$) or above($\boxed{+}$)) with respect to the splitting hyperplane. A naïve algorithm would require visiting $O(n)$ vertices (where n is the number of vertices). Bajaj and Pascucci observe that a half space range searching algorithm would be more efficient, since clusters of vertices can be classified simultaneously.

Similarly, the algorithm uses the classified vertices to classify each face against the splitting hyperplane. The algorithm assumes that no existing faces are embedded within the hyperplane. Those faces that cross the hyperplane are classified as incident on the hyperplane. Starting with the 1 dimensional faces (edges) of the complex, the incident edges are intersected with the splitting hyperplane to generate new vertices that are classified as being on($\boxed{=}$) the hyperplane.

As the algorithm iterates through the dimensions of the complex, the connectivity of those faces classified as $\boxed{=}$ is established. Those faces that are incident on the splitting hyperplane are all split by the hyperplane. This splitting is performed by adjusting the connectivity of the facets of each face, such that those faces on the negative side of the hyperplane are only connected to faces that are either $\boxminus$ or $\boxed{=}$. Similarly, those faces on the positive side of the hyperplane are adjusted such that they only connect to faces that are $\boxplus$ or $\boxed{=}$.

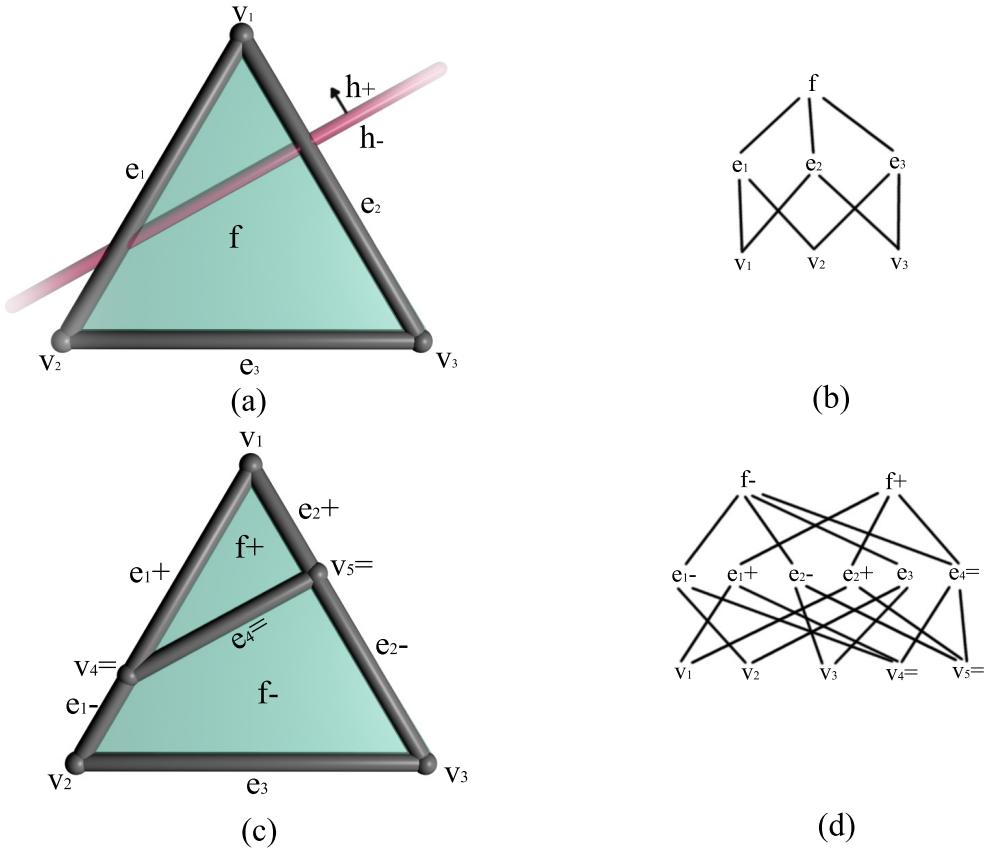


Figure 8: *Example – Triangle Splitting.* (a) A triangle in the plane. A hyperplane (a line in 2D), cuts the triangle in three, one part embedded in the halfplane h^+, one part embedded in the halfplane h^-, and the remaining part is embeded in the actual splitting line. (b) The face lattice of the triangle depicted in (a). (c) The triangle of (a), now split into a quadrilateral ($f-$) and a triangle ($f+$), sharing a new edge ($e_4 =$) and two new vertices ($v_4 =$ and $v_5 =$). (d) The face lattice representing the structure of (c).

To evaluate those factors governing the performance of this algorithm we note the following: the algorithm iterates through the faces of every polytope, for every dimension, starting with faces of dimension 1 and ending with those of dimension d (where d is the dimensionality of the polytope complex). At each face f, all the children ($d-1$ dimensional boundary elements) of f are visited. Ignoring the possible addition of new faces (since this augmentation is typically insignificant), the order of this traversal is $O(pm)$, where p is the total number of faces and m is the average number of children per face.

In Chapter 5, Section 5.3.1 we present a series of improvements on this algorithm that result in output sensitivity to those polytopes that are split.

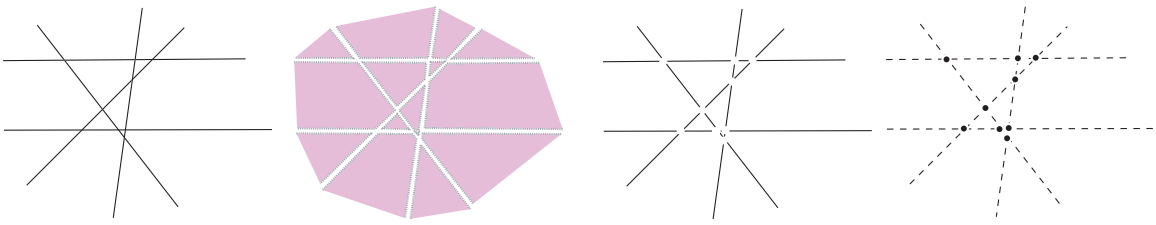
Begin
Step 1 *(primary numerical)* Classify all the vertices of c either $\boxplus$ or $\boxminus$. Then set $k = 1$.
Step 2 *(symbolic computation)* For each $c \in N_k$ do:
– If none of its facets is $\boxplus(\boxminus)$ then classify $c \boxminus (\boxplus)$ and goto Step 3;
– Create a new $(k-1)$-polytope f (classified as $\boxed{=}$) and connect it to each $(k-2)$-polytope in c classified as $\boxed{=}$. Create two polytopes c^+ and c^- connected both (down) to f and (up) to all the $k+1$ polytopes connected with c. Connect each $(k-1)$-polytope in c classified as $\boxplus$ to c^+, and each one classified as $\boxminus$ to c^-. Remove c from N_k.
Step 3 If $k < d$ then $k = k + 1$ and goto Step 2. Else continue to Step 4.
Step 4 *(Secondary numerical)* For each vertex v classified $\boxed{=}$, compute its coordinates by geometrically intersecting the hyperplane h with the edge divided in two parts by v.
End

Figure 9: *Bajaj and Pascucci – Polytope Splitting Algorithm.* The algorithm uses the following terminology: d is the dimension, h is the hyperplane, N_k is the set of all faces of dimension k.

3.4 Arrangements

A set of n hyperplanes in $\mathbb{R}^d$ define an *arrangement*. An arrangement is a partition of the space into cells. In this context, a cell refers to a subset of any dimension. A d-dimensional cell refers to the set of points lying on the same side of the same hyper planes. A $d-k$ dimensional cell is a set of points that lie on the same k hyperplanes.

Consider the case of $d = 2$. In this case, the hyperplanes are lines. In Figure 10 we show an example. Note that the number of 0, 1 and 2 dimensional cells are exactly 9, 23 and 15 respectively.



(a) (b) (c) (d)

Figure 10: *2D Arrangements.* (a) 5 lines (hyperplanes) (b) 15 2D cells (bounded/unbounded polygons) (c) 23 1D cells (segments and half-lines) (d) 9 0D cells (vertices)

Definition 3.14 *The* zone *of a surface in an arrangement is the set of cells incident on the surface.*

In general, the combinatorial complexity of an arrangement of n hyperplanes in a d dimensional space is $O(n^d)$ [EOS86]. Any given d dimensional cell (polyhedron) is bounded by $O(n^{\lfloor d/2 \rfloor})$ cells of any dimension [Ede87]. Edlesbrunner also gives formulae for computing exactly the number of k dimensional cells in a *simple* arrangment (i.e., an arrangment where exactly $d-k$ hyperplanes are incident on any k dimensional cell). These are therefore upper bounds for non-simple arrangements. Furthermore, an algorithm is given for computing hyperplane arrangments in worst case optimal time. We recommend Edelsbrunner [Ede87], if further reading on arrangements is desired.

3.5 Miscellaneous

In this section various advanced topics relevant to this dissertation, are discussed. Namely, the generalisation of the cross product to general dimension and the determination of the set of lines through four lines.

3.5.1 The Generalised Cross Product

The two-dimensional cross product is a familiar construct. In this section we will generalise the concept of the cross product to d dimensions.

In general, the d-dimensional crossproduct is a function of the form $f(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{d-1}) : (\mathbb{R}^d)^{d-1} \rightarrow \mathbb{R}^d$. That is, it takes a set of $d-1$ vectors of dimension d, to another vector of dimension d. For $d = 3$ we get, as expected, a function that takes a pair of 3-vectors, to another 3-vector. A property of the cross product is that the vector produced is orthogonal to each of the input vectors. The magnitude of the resulting vector is twice the $d-1$ dimensional volume of the hyper-parallelepiped defined by the basis vectors in the standard way[2].

The resulting vector may also be interpreted as a basis vector for the space of vectors orthogonal to the input vectors. This implies that the input vectors need to be linearly independent for a valid basis. Linear dependence implies that that the space of orthogonal vectors is of more than one dimension. A general kernel finding technique (such as a Singular Value Decomposition), may be used to compute the basis vectors for the entire set of orthogonal vectors.

The general cross product may be computed as follows: Let $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d$ be the standard basis of a d-dimensional space (i.e., $\vec{b}_i = (\delta_{1i}, \delta_{2i}, \ldots, \delta_{di})$ where δ is the Kronecker Delta). Let input vector j be of the form $(\alpha_{j,1}, \alpha_{j,2}, \ldots, \alpha_{j,d})$. From this, we get the following general formula for the cross product:

[2] Consider a kite in 2D, or the corner of a parallelepiped in 3D.

$$\vec{r} = \begin{vmatrix} b_1 & b_2 & \dots & b_d \\ \alpha_{1,1} & \alpha_{1,2} & \dots & \alpha_{1,d} \\ \alpha_{2,1} & \alpha_{2,2} & \dots & \alpha_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{(d-1),1} & \alpha_{(d-1),2} & \dots & \alpha_{(d-1),d} \end{vmatrix} \tag{4}$$

Where $\vec{r}$ becomes a vector orthogonal to all input vectors. Further reading and explanation can be found in Hanson [Han94].

3.5.2 Determining the Lines Through Four Lines

In this small subsection we briefly present a method that determines the set of lines through four given lines. The solution presented here is a summary of the paper by Teller and Hohmeyer [TH99]. The purpose of this summary is twofold. Firstly, we wish to present certain information that is used later in this dissertation. Secondly, we consider this topic an excellent exercise in developing intuition for Plücker coordinates.

There are many ways to represent lines in 3D implicitly. The most common, is as a set of any two distinct points on the line: a unique line intersects both points (the *join* operation [Sto91]). Another approach that is less frequently used is the representation of a unique line as the intersection of two given planes (the *meet* operation [Sto91]).

It is possible to define a set of lines implicitly through incidence, by using four lines. The set of lines that are incident on or *stab* these four lines may have cardinality 0, 1, 2, or ∞.

In the case of four mutually skew lines, there are exactly two lines incident on the given lines. Degeneracies may occur due to various configurations of parallel or intersecting lines. In this short summary, we only consider the general case (an example is depicted in Figure 11).

Section 3.1.5 shows how a line may be parameterised as a point using Plücker coordinates. Two lines x and y (in $\mathbb{R}^3$) are incident *iff* $D_{\Pi(x)}(y) = 0$. This property can be used to construct the set of lines incident on x: $\{\ell \in G : D_{\Pi(x)}(\ell) = 0\}$. Informally, the set of lines incident on x correspond to those points that are both on the dual hyperplane of $\Pi(x)$, and on the Plücker hypersurface (G).

Therefore, in order to find the set of lines incident on four lines, we need to compute the intersection of the set of lines incident on each one of the given four lines. This can be computed by finding the intersection of the four dual hyperplanes of each line, and then intersecting what remains with the Plücker hypersurface.

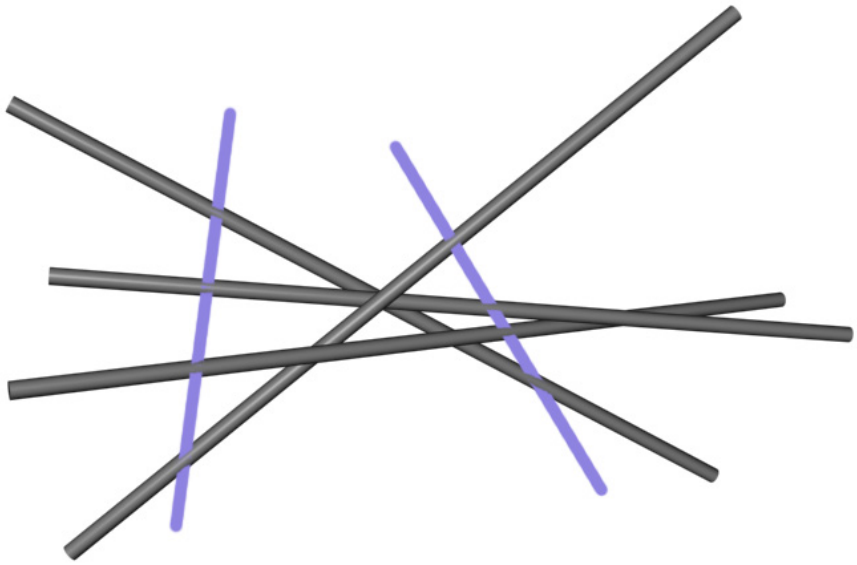


Figure 11: *Stabbing four lines*. Four lines specifying two lines by incidence.

The intersection of $d-1$ hyperplanes in a d dimensional space, will result in a one dimensional space (assuming the hyperplane constraints to be linearly independent). This one dimensional space can then be intersected with the Plücker hypersurface to give two solutions (since the Plücker hypersurface is a quadric). These two solutions correspond to the two real lines incident on the four lines. Mapping these points back to lines in 3-space gives the desired result.

The intersection of k hyperplanes may be computed elegantly, using a null-space computation. Teller and Hohmeyer [TH99] use a singular value decomposition (SVD) to compute the basis vectors of the null space. This affine set may then be intersected with the Plücker hypersurface. One of the basis vectors is fixed in order to account for projectivity.

It should be noted that the addition of another line would over specify the problem, leading to no incident lines. Symmetrically, the subtraction of one line (to three) would result in an under specification. The result would be a two dimensional null space, that when intersected with the Plücker hypersurface gives an infinite set of lines that form a ruled surface in $\mathbb{R}^3$, that is incident on the remaining three lines.

If the dual hyperplanes of the given four lines form linearly dependent constants, then the selection of lines are degenerate in terms of being mutually parallel or collinear.

Chapter 4

Aggressive Visibility Preprocessing

In this chapter we present our approach to rapid visibility computation. The algorithm is *aggressive* (as described in Chapter 1). We also extend the algorithm to creates a 5D subdivision in ray-space. This 5D subdivision is used as a means of accelerating ray-shooting.

Aggressive algorithms are not necessarily suited to all applications. Such an algorithm can be applied effectively when any of the following are met : (a) the perceptual impact of the error is acceptably low, (b) it handle scenes that cannot be solved effectively with a conservative alternative due to excessive overestimation, or (c) the time-frame allowed for preprocessing is less than that of a conservative or exact solution.

The technique presented here solves (b) and (c) effectively. However, (a) is difficult to show since *acceptability* is subjective and usually depends on the nature and context of the application.

We provide evidence quantifying this error in practice. We also present error minimisation heuristics that are exploited by adaptive sampling

We begin with a discussion on computing visibility from a surface in 3D space. Our adaptive technique is presented in this context. We then present the divide-and-conquer approach of our algorithm framework and we show how surface visibility can be integrated hierarchically, in order to greatly accelerate computation. We then discuss our cache management strategy that allows for a logarithmic dependency on the number of cells. Results exploring efficiency and accuracy are then presented.

Finally, we present our scheme for efficiently sampling 5D ray space. This is a natural extension to the visibility preprocessing technique presented here. We present experimental results from a preliminary implementation, demonstrating the utility of this algorithm as a means of accelerating ray-shooting. Since this is still a hardware based extension of our visibility algorithm, error may

result. *Area* sampling, a natural complement of *point* sampling, is introduced to further reduce error during the 5D processing.

4.1 Visibility From a Surface

In this section our novel sampling approach to "from-region" visibility is presented. We describe a sampling method that is based on the hemi-cube [CG85, HA00] and exploits the performance of common graphics rendering hardware. We demonstrate its application in deriving an aggressive visibility set from a rectangular surface in 3D space. Finally, adaptive sub-sampling is introduced as a means to increase performance, while decreasing error.

4.1.1 The Visibility Cube

A *point sample* is defined to be the set of polygons visible from a given point. A visibility cube (strongly related to a radiosity hemi-cube [CG85]) is used to generate such samples (see Figure 12). This is created by treating each of the six sides of a tiny cube enclosing the sample point as independent depth and frame buffers onto which the scene is rendered. Depth buffers are supported by all modern consumer level graphics hardware and ensure that only the pixels of these polygons visible from the point in question are rendered. Each polygon is assigned a distinct 32 bit colour. This allows a given pixel to be mapped back to the polygon responsible for its generation. Any polygon associated with a pixel is considered visible. The set of polygons that are mapped to by at least one pixel from any of the six frame buffers is considered to be the set of polygons visible from the sample point. The visibility cube can be considered a high density sampling over the angular domain, for a fixed spatial position.

The rendering process is not a traditional sampling mechanism, and is, in many ways, different from ray-casting through pixels. We give highlight of these differences in Table 2.

The intended application is for visibility culling in a rasterisation engine. A useful heuristic for obtaining good accuracy for point samples in practice is to set parameters (frame buffer resolution, bit depth of depth buffer and near and far planes) similar to that of the desired output parameters. For maximum accuracy, these factors should be set in accordance with the Nyquist limit.

Sub-sampling the intended rendering resolution is beneficial, however, since it enhances performance by minimising frame buffer reads and reducing the required fill rate, this allows accuracy to be traded for speed. We have found sub-sampling to be necessary when trying to achieve the optimal combination of accuracy and performance. Although accuracy is reduced, sub-sampling results

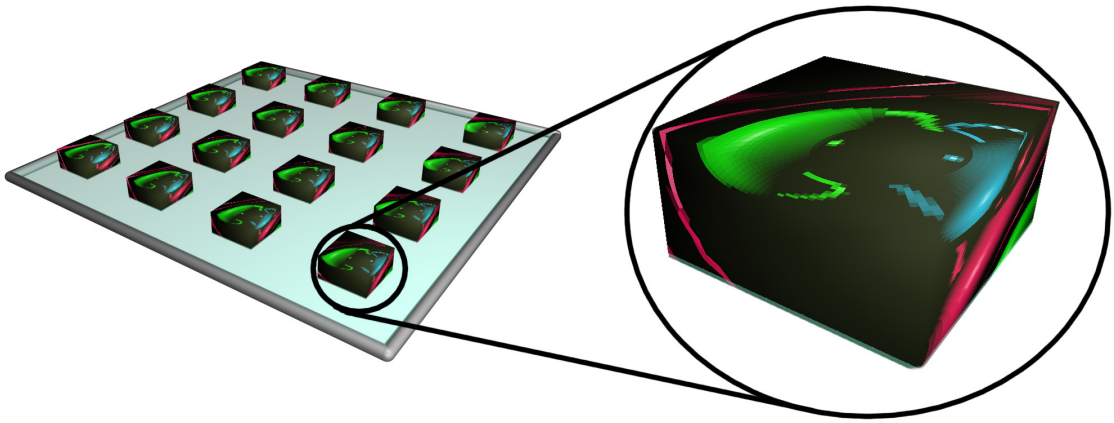


Figure 12: *The Visibility Cube*. A sample of several visibility cubes over a surface. The visible geometry (of several teapots) has been projected onto the cubes.

	Ray Casting	Rasterization
Geometry	Infinitely thin half-line	Sheared frustum. Size depends on sample resolution (pixel size).
Z-Fighting	–	Aliasing in the Z-buffer results in visual interlacing of polygons of similar depth.
Near/Far Clipping	–	Near and far planes may cause near or far geometry to be omitted.
Small Polygons	In the traditional sense of sampling, insufficient sampling results in polygons being omitted.	Small polygons are never "missed", rather only the *nearest* polygon intersecting a *pixel* frustum is selected.

Table 2: *Aliasing: Ray-Casting vs. Rasterisation*. Ray-casting and rasterisation do not produce identical results. We list several differences and catalogue the different types of aliasing artefacts that affect visibility.

only in the occasional omission of small polygons (with little perceptual impact). This is known as *approximate* culling or equivalently *contribution* culling [ASVNB00, BMH98, Zha98].

The concept of a "from-region" visibility set can be defined in terms of point samples. This is simply the union of the visible sets of all possible point samples taken within the rectangular region. Since there are an infinite number of these points, an exact evaluation via point sampling is impossible. Instead, we form the union of a finite subset of point samples.

Insufficient sampling leads to aliasing artefacts that manifest as the exclusion of visible polygons (*false invisibility* error) when rendering.

Performance

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

Firstly, we consider the rendering process. The generation of one side of a visibility cube is similar to that of standard rendering, however several simplifications can be exploited:

1. Mapping is not required (texture maps, bump maps, environment maps, light maps)
2. Smooth shading (Gouraud/Phong) is not required
3. Lighting calculations are not required
4. All geometry in the preprocess is static

This implies that the rendering process can be achieved more efficiently than traditional rendering, often approaching the peak efficiency of the graphics hardware. For further efficiency, our implementation incorporates the following:

1. High performance video/AGP memory is allocated for geometry when possible. Using (on our hardware) 230mb of such memory allows for 8 million triangles to be stored.
2. Triangle stripping enhances performance, and allows more geometry to be inserted into high performance memory.
3. Geometry could be uploaded to video/AGP while rendering using synchronisation extensions to exploit CPU-GPU parallelism. We have not implemented this since we have not found the amount of high performance memory to be a limitation. The large quantities of available high performance memory is a consequence of unused texture memory.

Current hardware (we use an NVidia GeForce4 Ti 4600) claims to be able to transform 136 million vertices per second. Already, top end hardware, such as the ATI Radeon 9700, claims to be able to double or even triple this. In practice, we achieve 17 million triangles per second throughput. Since our samples are taken within the scene bounding box, the slow-down is due mainly to an insufficient fill rate, since nearby triangles consist of many pixels.

Any acceleration technique that can be applied to traditional point rendering can also be applied to visibility cube rendering. We implement frustum culling using bounding spheres. This accelerates our throughput by a factor of 4.2. We optimise the process by noting that six views (partitioning

the full angular domain) need to be computed from the same point. We note that each of these six (infinite) frusta are bounded by four planes from a shared set of six. These six planes are well defined, and are those six that intersect the center and embed any two edges of the visibility cube. We classify the bounding spheres with respect to the six planes once, and use this classification for all six sides.

We consider the utilisation of point-based occlusion techniques as an area of future investigation. It should be noted that during our preprocess, visibility information that has already been computed is exploited (see Section 4.2.2).

The second main issue is that of frame-buffer reading. This is often a bottle neck. Wonka *et al.* [WWS00] claim that this accounts for approximately 54% of their run-time. This is most likely due to the fact that they only render simplified scenes (their 8 million triangle scene is represented by a much smaller building "facade"), and thus may exaggerate the frame buffer read times for more general scenes. Frame buffer reading consists of a considerably smaller part (20%) of our run-times.

The performance of frame buffer reads has not improved at the same rate as triangle rendering. The read bottleneck is due to limitations on bus technology. Older UMA (Unified Memory Architecture) hardware such as the SGI Visual Workstation are on par with current hardware (Intel Pentium 4 with GeForce 2/3/4). Frame buffer reads (RGBA channels) occur at approximately 46 million pixels per second (for 512x512 pixel blocks). Much older UMA hardware such as the SGI O2 only read at 11 million pixels per second.

In Section 4.6 we give several suggestions on how specialised hardware could be engineered in order to enhance visibility cube rendering and frame-buffer processing.

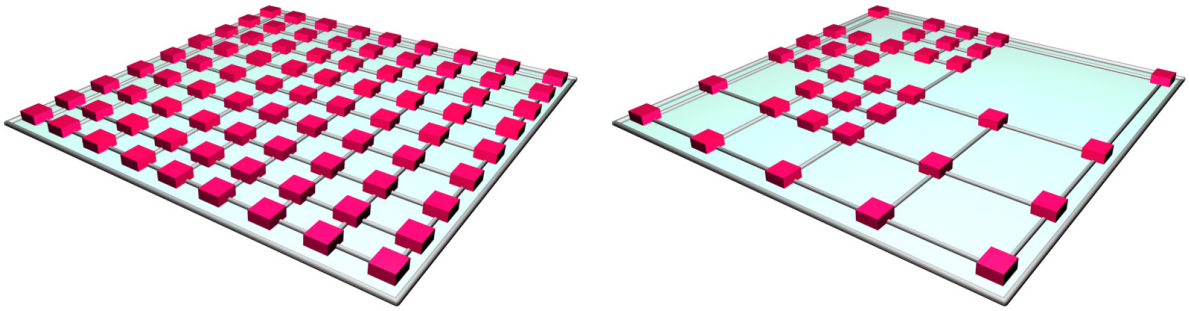
4.1.2 Uniform Sampling

For our purposes, uniform sampling is a naïve solution to the sampling problem (see Figure 13a). With this approach, under-sampling may result in unacceptable error, while over-sampling may lead to prohibitive execution costs. An adaptive sampling method is necessary.

4.1.3 Adaptive Sampling

We assume a rectangular sample domain embedded in 3-space. To begin with, point samples are evaluated at the corners of the rectangle. Then a decision is made whether or not to refine the rectangle into four subregions based on an error minimisation heuristic. The user specifies an error threshold (covered in this section). The 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. It is important to cache shared point samples in order to prevent redundant computation. A typical adaptive subdivision is illustrated in Figure 13b.



(a) (b)

Figure 13: *Uniform vs. Adaptive Sampling.* (a) A uniform distribution of visibility cubes on a 2D surface. (b) A non-uniform distribution of visibility cubes generated by an adaptive subdivision. The adaptive sub-division attempts to minimise both error, and the number of samples required. Effectively, a quad-tree structure is built on the surface.

Basic Error Metric

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

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

$$Err(s_{0..3}) = 1 - min_{i=0}^{3} \left(\frac{|\bigcap_{j=0}^{3} s_j|}{|s_i|} \right) \tag{5}$$

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

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

distribution.

Strict Error Metric

Before we detail our stratified error metric, we would like the reader to consider an alternative. Let us define a new heuristic as follows:

$$\text{diff}(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 = x_2 = x_3 = x_4 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$Err'(s_{0..3}) = 1 - \frac{\sum_{i=1}^{N} \text{diff}(s_0(i), s_1(i), s_2(i), s_3(i))}{N} \quad (7)$$

We assume that the pixels of each visibility cube are enumerated consistently from 1 to N. We define $s_j(i)$ as pixel i of sample j. Where Err returns 0 *iff* every sample sees the same polygons, Err' return 0 *iff* each sample generates the same visibility cube. This adds an extra constraint: it is not sufficient for a polygon to simply be seen by all samples, it must be visible in exactly the same directions/pixels.

The heuristic defines a measure of image based similarity. It is even possible for each sample to see exactly the same set of polygons, but for Err' to still return 1. It should be clear that if four samples see the same polygons, at the same pixels (i.e., they see the *same images*), then it is highly improbable that further refinement is necessary.

This measure is infeasible in practice, since 0 error will only occur when samples are very near each other. It is also likely that the error heuristic will increase excessively as the distance between sample points grows. In the next section we present a stratified metric that provides a compromise between Err and Err'.

Stratified Error Metric

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

$$Err''(s_{0..3}) = max_{k=0}^{d} \left(1 - min_{i=0}^{3} \left(\frac{|\bigcap_{j=0}^{3} s_{jk}|}{|s_{ik}|} \right) \right) \quad (8)$$

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

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

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

Treating and Exploiting Manifold Meshes

Another approach that improves the error heuristic is the exploitation of specific scene properties. For instance, many scenes consist of manifold objects with interiors that do not represent valid view-points. In this case, each of the four point samples can be classified as interior or exterior. Equation 8 is applied to each case independently. We denote the set of exterior and interior samples as E and I, respectively. Some simple properties are: $|E| + |I| = 4$ and $E \cap I = \emptyset$. Two thresholds t_e and t_i are defined. We subdivide *iff* any of the following conditions hold:

- $|E| = 1$ Single exterior point does not provide sufficient information for a final decision.
- $|E| > 1$ and $Err''(E) > t_e$. This is the difference between samples, considered only at *valid* camera positions.
- $|I| > 1$ and $Err''(I) > t_i$. If the interior difference 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 visible set. A point in the same half-space as the normal of a visible polygon is considered exterior. Caveat: in practice, discretisation errors typically cause a few pixels from backfacing polygons to be visible along the silhouette of an object. To counter this, the polygon that contributes the most pixels is chosen as the half-space classifier. This classifier selection can be efficiently integrated into the processing of the visibility cube buffers.

4.2 Algorithm Framework

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.

4.2.1 Visibility From a Volumetric Region

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

$$V(P^+) = V(R) \cup I(P^+) \tag{9}$$

$V(R)$ represents the visibility from the rectangle R, and can be evaluated with the method discussed in Section 4.1.3. The set $I(P^+)$ is simply those polygons that intersect P^+, and can be computed with a simple polygon-cuboid intersection algorithm. Similarly, $V(P^-)$ may be expressed as:

$$V(P^-) = V(R) \cup I(P^-) \tag{10}$$

In general, the visibility set, $V(C)$, of a cell, C, is the union of those polygons that intersect C and those polygons visible from the surface of C.

4.2.2 Hierarchical Subdivision

A hierarchy of cells is generated in a top down fashion by starting at the scene bounding box and 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 subdivision process of a cell is illustrated in Figure 14. Our method for maintaining, reusing and distributing cells is also discussed in this section.

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

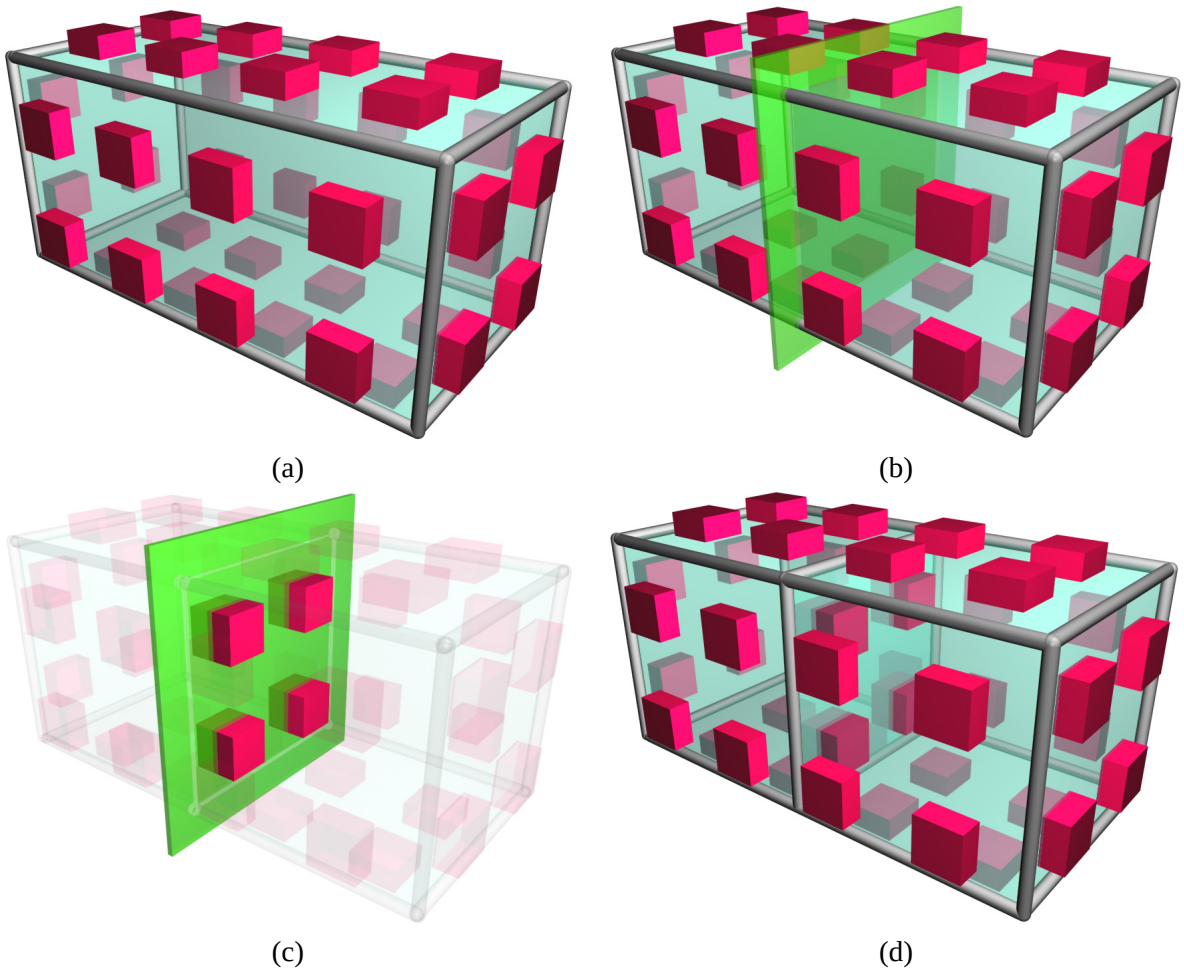


Figure 14: *Hierarchical Subdivision*. (a) A typical cell within the hierarchy. (b) The chosen splitting plane within the cell. (c) New samples are generated on the sub-division. (d) The original cell is now partitioned. Note, that when a cell is subdivided, only those samples on the partition plane need to be evaluated. The samples shown in (a) need to be cached. Should both cells terminate subdivision after (d), then the samples shown in (c) would be obsolete and can then be deleted from the cache.

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

When the subdivision of a cell ceases, all samples on the surface of the cell are aggregated. The

union of the set of visible polygons for all samples is computed and stored with the cell.

An advantage of top-down hierarchical subdivision is that smaller sub-cells that do not contribute significantly to culling, are never evaluated in contrast to the bottom-up subdivision of van de Panne and Stewart [vdPS99].

A second advantage is that previously computed information in the upper levels of the hierarchy may be exploited to accelerate the evaluation of the lower levels.

Superset Simplification

In theory, the set of polygons visible from a cell, C, is a superset (allowing for potential omissions due to aggressive sampling), of those visible from any view-point, q, within C: $V(q) \subseteq V(C)$ $\forall q \in C$. This allows the process of splitting a view cell to be optimised. When generating samples on the splitting rectangle of a cell, only polygons known to be visible ($V(C)$), need be rendered. This implies that the cost of building a point sample decreases as its depth in the binary hierarchy increases.

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

Cache Management

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

Typical implementations result in a set of cells being generated on a uniform grid. Each grid element can then be expanded into a hierarchy. Cohen-Or *et al.* [COFHZ98, NFLYCO99] use a fixed two level hierarchy. Durand *et al.* [DDTP00] also use an initial grid.

Our approach, however, has an interesting property that allows us to evaluate a parent cell, and then split the parent (if so desired) into two child cells, at the *same* cost as generating the child cells initially. This requires the ability to partition an existing rectangular region, while being able to disseminate the correct visibility information to the partitions, without (significant) additional computation. Since we use a sample based approach, all the samples belonging to the parent region

are distributed to their associated partitions. In order to keep the cost low, we *cache* all samples until they are no longer required.

The process proceeds as follows (illustrated in Figure 14): First, we assume that the visibility (surface samples) from some cell C has been computed (Figure 14a). Second, if subdivision is indicated (depending on subdivision criteria), a splitting plane is chosen (by a heuristic) that splits the cell into C^- and C^+ (Figure 14b). Third, the required samples on the shared boundary of C^- and C^+ are computed (subject to the visibility set $V(C)$) (Figure 14c). Fourth, the cells in the negative and positive half-spaces (as defined by the splitting plane) are propagated to C^- and C^+, respectively (Figure 14d). Finally, cell C is deleted.

Removing Redundant Samples

Samples are considered redundant if they cannot possibly be used again. During sub-division of the cell hierarchy, parent cells are split into child cells. At this point, cached samples are fetched and distributed to the correct cells. When a cell has terminated its sub-division process, its aggregate visibility is computed as the union of the point sample visibility.

In order to save memory resources, it is necessary to compute which of the samples associated with the current (now finalised) cell are redundant, and that will be used again. Any given sample can be shared among a maximum of eight cells, and a minimum of two. In the former case, the sample must exist as a shared corner of eight cells, in the latter case, the sample must be interior to a shared side of two adjacent cells. It is also possible for a sample cell to lie along the edges of a cell.

We associate a counter with each sample. When subdivision terminates for a cell, the counter of each sample on the cell is updated. Samples interior to the cell wall are incremented by four, samples interior to a cell edge are incremented by two, and samples on a cell corner are incremented by one. When the counter reaches eight (and eventually, the counter of *every* sample will reach eight), the sample is deleted, since it must fall interior to a set of cells that will no longer be subdivided.

For most samples, the initial counter values are set to zero. Samples interior to a side of the initial scene bounding box are set to four, those on an edge are set to six, and on a corner are set to seven.

The depth first traversal that builds the hierarchical subdivision thus allows for the early removal of samples.

4.2.3 Algorithm Analysis

The adaptive nature of the algorithm makes it difficult to obtain a useful upper bound. So, in order to obtain a useful complexity estimate, we make several simplifying[2] assumptions. First, we note that the core operation is generating samples on surfaces. We let ω be the surface area of the splitting surface at the first level of the hierarchy (i.e., the area of the surface that splits the bounding box in two). We let k be the number of samples on this surface. Our first assumption is that the sampling *rate* of $\frac{k}{\omega}$ samples per unit area applies globally. Secondly, we assume that the bounding box is a cube. Thirdly, we make the assumption that the computed spatial hierarchy is balanced, and is of depth d. From this, we determine the number of cells, c, to be 2^d.

Since the volume is a cube, the total new sampling surface area at any level in the hierarchy is equal to ω. This implies that the total sampling surface area is $d\omega$. kd samples are therefore required to sample all the required area. This can also be expressed as $k \log c$ samples.

In terms of triangles rendered, we select a scene size n. Assuming exactly $6n$ polygons are rendered at each point (ignoring frustum culling), we deduce that $6nk \log c$ polygons are rendered in total.

If we account for superset simplification, however (Section 4.2.2), the situation is markedly different. The current upper bound $O(nk \log c)$ is only tight if every polygon is visible from every cell. This is an unrealistic case, and is certainly not one to which any occlusion algorithm should be applied.

To account for superset simplification we define a rate of decay α that is assumed to be constant. This is the factor by which the size of the visibility set decays per level, on a parent to child basis. For example, if $\alpha = 0.6$, the size of the visible set of some cell is 0.6 times the size of the visible set of its parent. In practice, α remains fairly constant, however it can grow towards 1 in the deeper levels of the hierarchy. It becomes 1 when the child cell sees exactly the same as its parent. This is clearly a condition for terminating subdivision, although in practice the rapid changes in visibility exhibited by most real scenes prevents most cells ever getting close to 1. As the depth complexity of the scene increases, the average value of α tends towards 0.5.

Using this rate of decay, the actual number of polygons per sample rendered at depth q can be expressed as $n\alpha^{q-1}$. The total number of polygons rendered P can be refined to:

$$P = kn + kn\alpha + kn\alpha^2 + \ldots + kn\alpha^{d-1} \quad (11)$$

$$\Rightarrow P = kn(\alpha + \alpha^2 + \ldots + \alpha^{d-1}) \quad (12)$$

[2] Simplified, but typically true in practice.

Asymptotically, this implies that P is of $O(kn \log d)$ that becomes $O(kn \log(\log c))$. The series converges if $\alpha < 1$, and this is witnessed in practice. Each additional level, requires a similar number of samples to the previous level. However, each sample can exploit the visibility information from the parent, thus becoming less expensive to compute. Similarly, the addition of an individual cell within the new level (by subdivision), is less expensive than a cell in the previous level, since: a) the splitting planes are, on average, half the surface area and b) the cost of rendering the samples on this reduced surface is decreased by the decay. It should be noted that asymptotically, the cost of a sample tends to become "free". This is not quite true, since frame buffer read times always add a (non-zero) constant per sample. For all practical cases, though, it is unlikely that frame reads will dominate (currently reads comprise 20% of the preprocess). Should this occur (due to an unlikely advance in rendering performance), the algorithm will run in $O(kd)$ time.

The final consideration is that of k. In most cases it is likely that k is some function of n. However, it is *also* a function of the geometric scene distribution and general configuration, the error threshold and the subdivision heuristic. The way k relates to n is thus dependent on the nature of the scene. We have found that in practice, k is considerably smaller than n, usually around four orders of magnitude smaller.

4.3 Results

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

4.3.1 Performance

We begin by showing that the algorithm can be used to preprocess scenes[3] that cannot be processed effectively by existing solutions. We use a *large* forest scene consisting of 5 million polygons (see Figure 15). Nearly 200 trees, each highly detailed, consisting of 25 000 polygons each. A "small" forest scene, consisting of 2 million polygons is also tested. This scene consists of 80 trees, also of 25 000 polygons each (see Figure 16). Note: internally, we make no use or assumption of instanced geometry.

[3] Videos of the aggressive algorithm are available at: http://www.cs.uct.ac.za/s̃nirenst/Vis

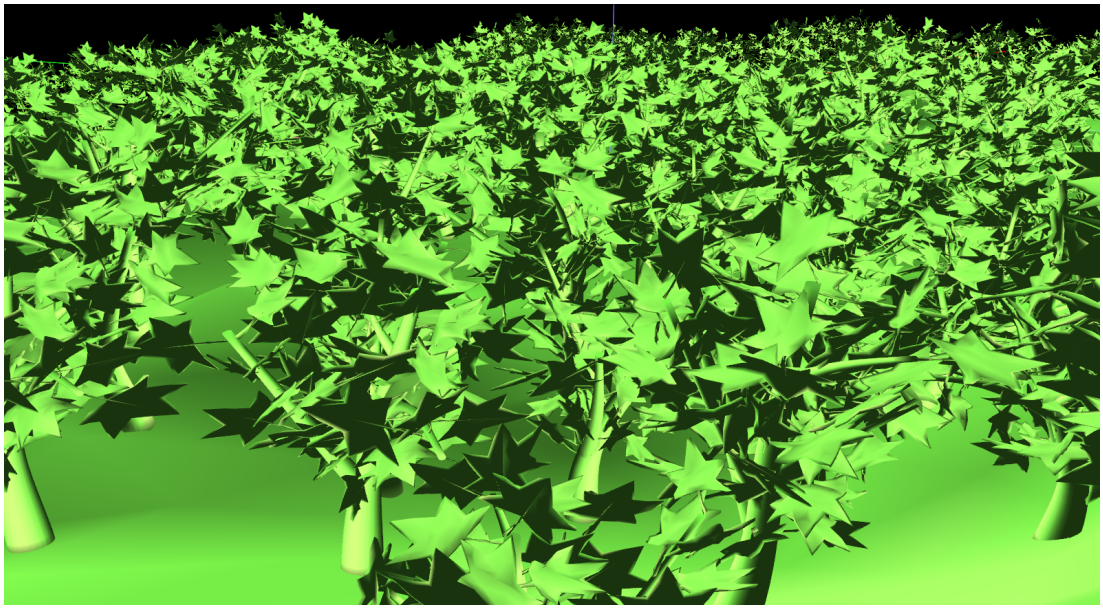


Figure 15: *Test Scene – Forest Model.* A very complex forest model consisting of 5 million polygons. Each tree consists of 25 thousand polygons. The image shows the output of our algorithm. 11.9% of the scene is rendered from the region containing the view point.

For reference, we pre-process the forest scene used by Durand [Dur99]. This scene consists of approximately 1450 trees[4], at 1000 polygons each (see Figure 17). The scene is far more densely occluded than our large forest scene, implying that more culling can be performed.

We also test a large, complex *town* scene comprising 1.35 million triangles (see Figure 18a). This scene is realistic, and consists of a mix of simple and detailed objects (see Figure 18b).

First, we consider the visible set size. Being an aggressive algorithm, it is to be expected that the visible subset is less than or equal to the results of an exact solution. Durand *et al.* [DDTP00] cull 75% of their scene, resulting in a visible set of 25%. In comparison, we are able to cull 99.21% from the model (see Figure 19). Where the extended projection algorithm took 17 seconds per cell, our technique takes 2.57 seconds per cell.

Second, we consider the effectiveness of the algorithm when applied to our large forest scene. 91.32% of the scene is culled on average (see Figure 20a). This allows for an acceleration of 11.5 times that of naïve rendering. We have no knowledge of any existing alternative algorithm capable of preprocessing this scene to a significant degree. Indeed, although the exact algorithm presented in

[4] And several flamingoes.

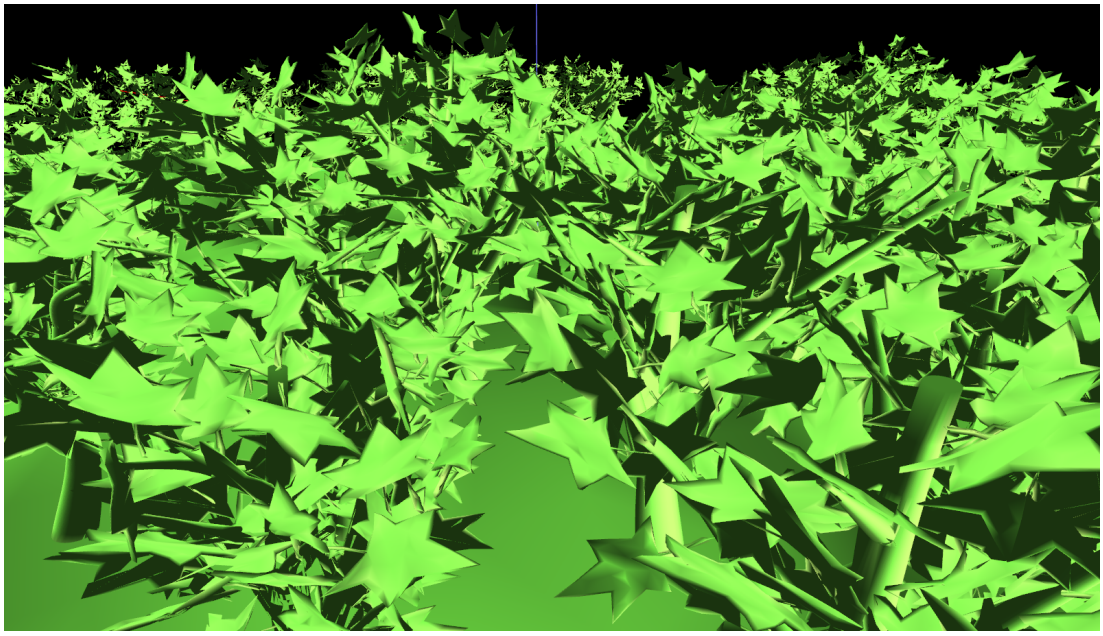


Figure 16: *Test Scene – Forest Model.* A complex forest model consisting of 2 million polygons. Each tree consists of 25 thousand polygons. The image shows the output of our algorithm. 11% of the scene is rendered from the region containing the view point.

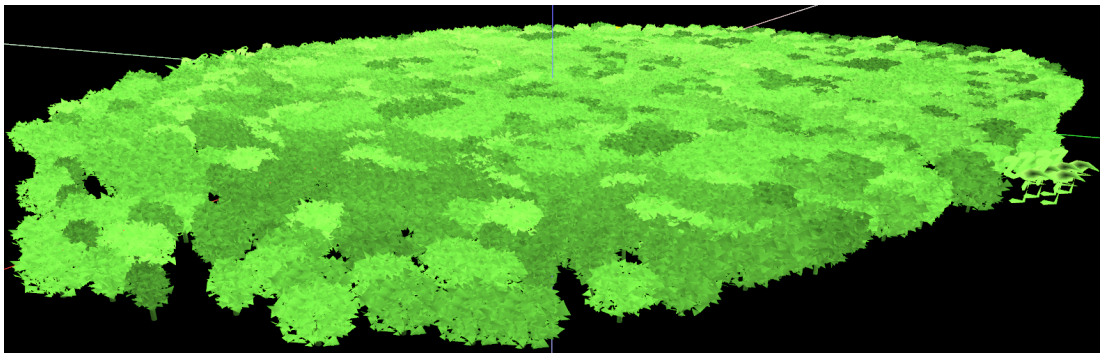
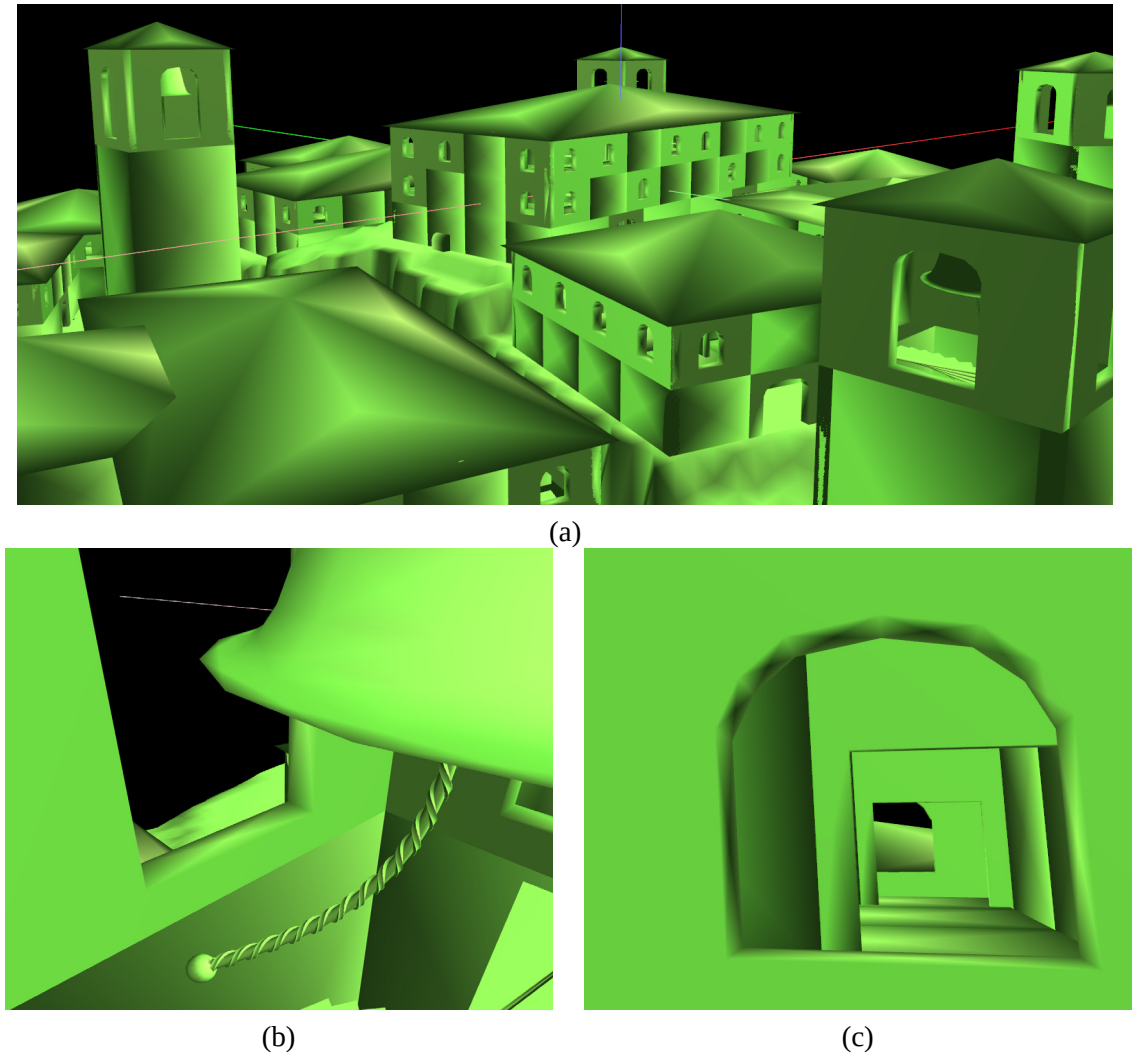


Figure 17: *Test Scene – Durand's Forest.* The forest scene used by Durand *et al.* [DDTP00]. This flat forest consists of 1450 trees of 1000 polygons each.

Chapter 6 is capable of processing it, without the possibility of error, the time required is excessive for a single workstation. The large forest model is processed at a rate of 9.23 seconds per cell for a total time of 1 hour and 19 minutes.



(a)

(b) (c)

Figure 18: *Test Scene – Town Model.* (a) A complex town model consisting of 1.45 million polygons. The detail objects within this scene consist of up to 40 thousand polygons each. The image shows the output of our algorithm. 1.4% of the scene is rendered from the region containing the view point. (b) A highly detailed object (rope) within the scene. (c) The buildings are honeycombed, allowing sight lines right through them.

The town scene is processed at 2.5 seconds per cell for a total time of 21 minutes and 20 seconds. An average of 1.45% of the scene was determined to be visible (see Figure 20c).

Using the error threshold, we see that it is possible to trade quality for performance. Using the 2 million polygon forest scene we see that a 32% performance increase was gained by increasing the threshold. This tradeoff was less effective for the town scene. We discuss the reasons for this

Exp.	Scene	Size	Threshold	Time/Cell	No. Cells	Visible Set
1	Forest	5m	0.2	9.23s	512	8.68%
2	Forest	2m	0	5.48s	512	14.84%
3	Forest	2m	0.999	4.17s	512	13.63%
4	Durand *et al.* [DDTP00]	1.45m	0.99	2.57	400	0.79%
5	Town	1.35m	0	2.5s	512	1.45%
6	Town	1.35m	0.999	2.46s	512	1.35%

Table 3: *Aggressive Algorithm/Preprocess – Performance Results*. The *Experiment* column is the experiment reference number. Each experiment number corresponds to one preprocess and analysis. They may be used to cross-reference the error results in Table 4. The *Scene* column indicates the type of scene. The *Size* column gives the size of the scene (in triangles (m = millions)). The *Threshold* column indicates the error threshold used in the preprocess (see Section 4.1.3). The *Time/Cell* column gives the time taken per cell. The *Number of Cells column* shows the number of cells into which the bounding box was subdivided ($512 = 8 \times 8 \times 8$ and $400 = 20 \times 20$). The *Visible Set* column provides the percentage of visible geometry averaged through all cells. The scenes were all sampled using 512×512 pixel item buffers.

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

Table 4: *Aggressive Algorithm/Preprocess – Error Results*. The *Experiment* column is the experiment reference number. Each experiment number corresponds to one preprocess and analysis. They may be used to cross-reference the performance results in Table 3. The *Average Error* column gives the average image error for a sampled camera path (details in Section 4.3.2). The values in parentheses are the absolute number of pixels corresponding to the percentage (a 512×512 pixel view is used for our tests). The *Maximum Error* column is the image error for the frame with the largest error. The *CR Count* column gives the minimum number of connected regions (CR) into which the erroneous pixels may be partitioned. The CRs are averaged over the frames in our test path. The *Average Maximum CR* column is the average size of the *largest* CR in each frame. The *Total Maximum CR* column is the size of the maximum CR over all frames of the walk through. The scenes were all sampled using 512×512 pixel item buffers.

in Section 4.3.2. We also discuss the quality implications of the tradeoff in the next section. Given the exponential shape of this sensitivity, end user application (i.e., a person wishing to preprocess a model) should use a logarithmic scale to manipulate the threshold scale.

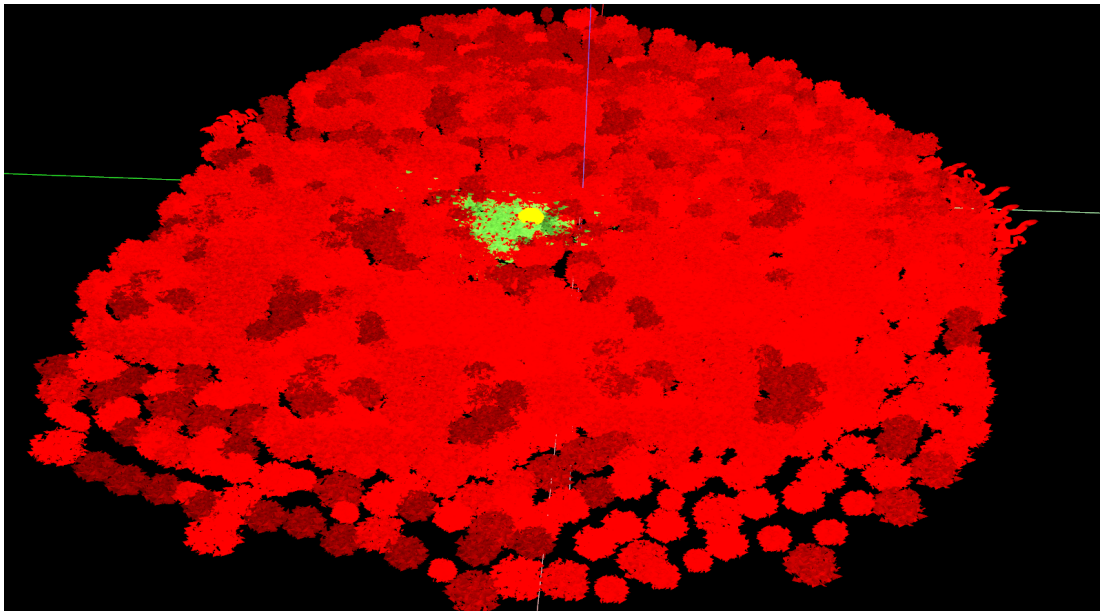


Figure 19: *Test Scene – Aggressive Culling of Durand's Forest.* The forest scene used by Durand *et al.* [DDTP00]. From the cell of the given view point (yellow sphere), 0.8% of the scene is visible. Visible polygons are shown in green, while invisible polygons are drawn in red.

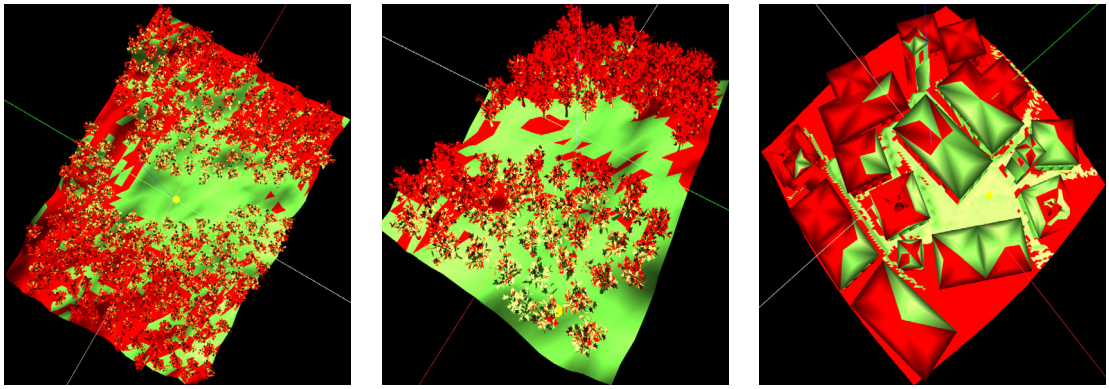
In order to ascertain scalability with respect to the number of cells, we have executed the preprocess for our 2 million polygon forest scene using a varying number of cells. The results are shown in Figure 21.

This demonstrates the logarithmic dependence of time on the number of cells (see Section 4.2.3). There is a horizontal asymptote, that is offset by a positive non-zero value from the origin. This limit is approached when the *rate of decay* (see Section 4.2.3), converges to 1.

4.3.2 Error

We present error results for the various models in Table 4. The error results are found by replaying a walkthrough of several hundred frames. For each frame, the visible geometry is rendered in green (flat shaded and unlit). Similarly, the "invisible" geometry is rendered in red. We count those pixels that are red, and consider this to be the number of erroneous pixels or *error* in the frame. See Figure 22 for an example.

We were unable to obtain error results for Durand's scene using this evaluation method, since there is no reasonable path through the scene. Without such a path, geometry gets clipped to the



(a) (b) (c)

Figure 20: *Aggressively Culled Scenes – Large Forest, Small Forest and Town.* Sample output from our aggressive algorithm. Green polygons are visible and red polygons are invisible. The view point used is that of the yellow sphere. (a) 4 million polygons are culled (21.6% of the scene is visible) from the given view point. (b) 1.9 million polygons are culled (10.3% of the scene is visible) from the given view point. (c) 1.1 million polygons are culled (2.5% of the scene is visible) from the given view point.

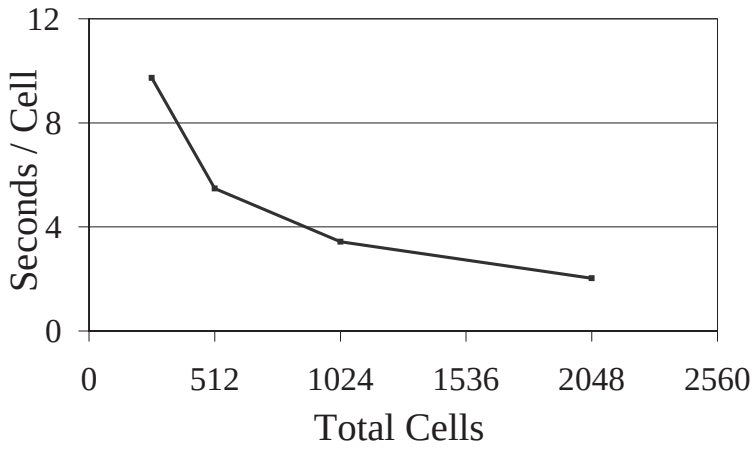


Figure 21: *Scalability by Cell.* The average time taken per cell (vertical) is plotted against the number of cells by which the bounding box is subdivided. As evidenced, there is exponential decay in the time taken per cell. The model used is our 2 million polygon forest model.

near-clipping plane, allowing invisible geometry to show through. When comparing our results to those of the exact algorithm developed in Chapter 6, we find that the aggressive result is 10.2% smaller than the exact result. We do not use this method for comparison in the other scenes, since it

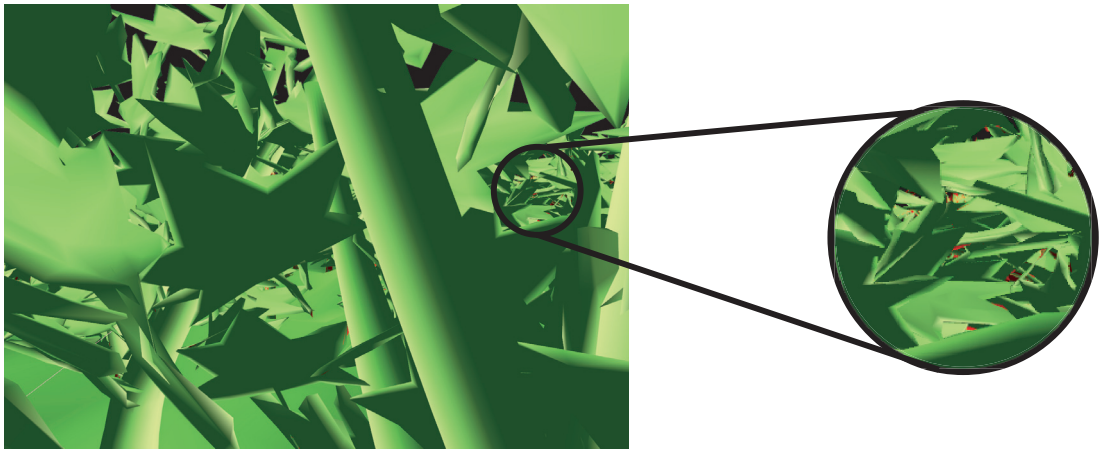


Figure 22: *Error Measure*. A 1600×1200 screen shot from a walk through. Green polygons are those determined to be visible by our aggressive algorithm, while the red polygons are determined to be invisible. The appearance of red pixels mark erroneous rendering. Shading and lighting are used for this image, although our automated error evaluator uses constant colouring.

has little bearing on the actual image error.

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

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

The other experiments show how our technique fared for other scenes, using small and large

thresholds. The town scene resulted in lower average and maximum error, while the forest scene resulted in more fragmented errors.

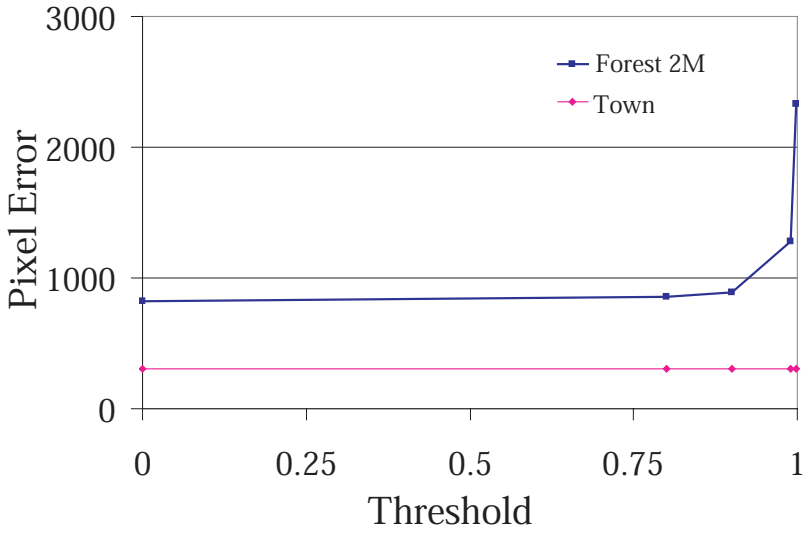


Figure 23: *Error vs. Threshold*. The average pixel error for a walk through as a function of threshold.

We show how error is affected by error thresholding. The forest scene is particularly sensitive to this threshold, as depicted in the graph of Figure 23, while the town scene is not. The reason for this disparity, is that the forest allows for gradual changes in visibility as a view sample moves through a region. In contrast, the town scene can have a very large change in visibility from one nearby view sample to another (e.g., if an adjacent view sample crosses a wall). This large change in visibility will force any reasonable threshold to continue subdividing, thus resulting in the consistent, low image error shown in the figure.

Similarly, we show how the threshold relates to the average size of the connected regions in Figure 24, and the maximum size of the connected regions in Figure 25. Despite the fact that the average error is lower, the town scene results in a larger average connected region error, since when an error does occur, it tends to be larger, because the town scene contains larger triangles than the forest. The maximum connected region error is larger for the forest scene, although this is because several viewpoints of the test path contain clusters of sub-pixel size polygons. Such regions are usually perceived as noise and do not impact significantly on image quality.

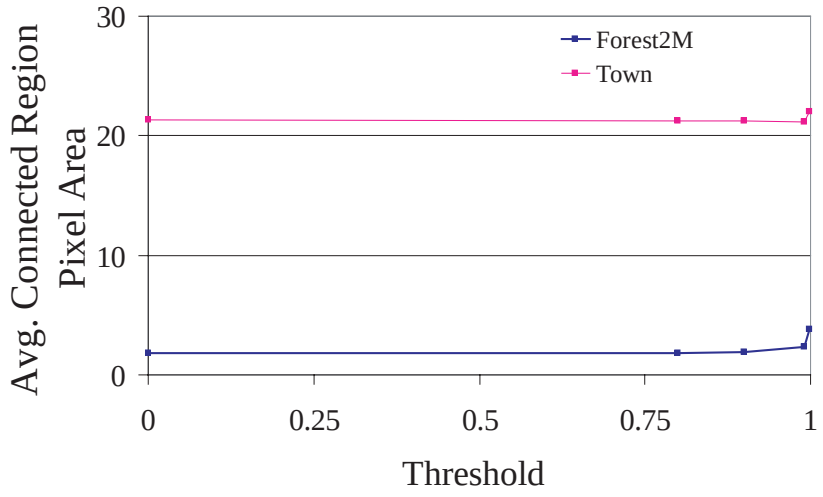


Figure 24: *Average Connected Region vs. Threshold*. The average size of the connected error regions (in pixels) for a walk through as a function of threshold.

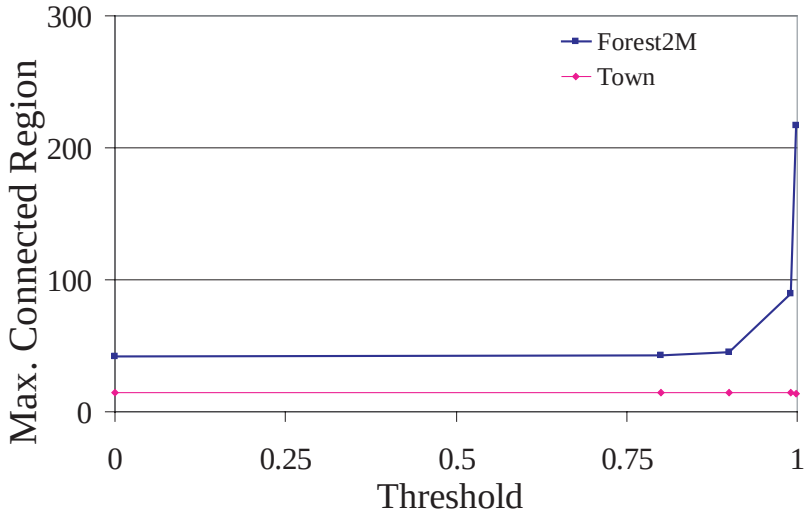


Figure 25: *Maximum Connected Region vs. Threshold*.
The largest connected error region (in pixels) for a walk through as a function of threshold.

4.4 Preprocessing Ray Space

In this section we [5] present an extension to the aggressive visibility algorithm presented earlier in this chapter. This section covers the work of Sharpe *et al.* [SHN+03] in context.

Ray shooting is a simple yet computationally expensive task that is fundamental to many algorithms in computer graphics and computational geometry [Aga92, AS93]. The "ray-shooting problem" is to find the nearest intersection of a ray with a set of geometric objects, with respect to the ray origin. Ray shooting is used to calculate form factors for radiosity, to create photon maps, and is, of course, the central component of both classic and Monte-Carlo ray tracing. In this section we present a new approach to accelerating ray shooting.

A naïve approach requires that every ray is tested against every geometric primitive in the scene. In order to reduce the number of ray-primitive intersection tests required, researchers have developed a range of acceleration schemes. The most popular schemes use spatial subdivisions, such as binary space partition trees or uniform grids. We refer the interested reader to the thesis of Vlastimil Havran for a critical survey [Hav00]. The goal of such schemes, is to reject trivially those parts of the scene that cannot be intersected first by a given ray.

With the same objective in mind, we propose an approach based on Arvo and Kirk's *ray classification* [AK87] technique. The original algorithm subdivides the scene using a 5D partition of ray space. The algorithm *classifies* the query ray into a 5D cell and returns a candidate set of primitives associated with this cell. Our key contributions are:

1. An algorithm that *fully* pre-processes the scene[6], effectively pre-computing all candidate sets. This allows for more efficient ray shooting after a once-off pre-process.

2. An upper bound on the candidate set size at run-time. This may be used to obtain upper time bounds for run-time rendering.

3. A new technique for computing the candidate sets. Graphics hardware is exploited to accelerate the pre-process. Adaptive sampling is used to minimise error.

4. We present a method that accounts for occlusion within the candidate sets. This increases the performance of the ray shooting by trivially rejecting *all* hidden primitives, while simultaneously decreasing the memory requirements (to feasible levels) for a full pre-process.

[5] The implementation of the proposed algorithm formed an Honours (4th year) project. The project team consisted of Adrian Sharpe and Matthew Hampton. The project was proposed and supervised by Shaun Nirenstein. Additional supervision was performed by James Gain and Edwin Blake.

[6] As opposed to the lazy evaluation used by Arvo and Kirk.

In terms of application, this algorithm may be used to preview ray-traced scenes rapidly. The aggressive nature of the algorithm may make the resulting images unsuitable for production release, although as a (possibly interactive) preview, this technique presents an excellent compromise between quality and performance.

4.4.1 Brief Background

Since Rubin and Whitted [RW80], there has been much research devoted to ray-shooting acceleration schemes. Most methods use 3D spatial subdivision in conjunction with a ray traversal algorithm. Each cell or voxel contains a list of the primitives that are fully or partially contained within it. The ray traversal algorithm traverses the cells along a ray in order. The list of the nearest cell is tested first, so that if an intersection point is found within the boundary of the cell, no further cells need be tested. However, if no intersection is found, or the intersection point is outside the cell's boundary, the primitive list within the next cell must be tested.

These schemes perform reasonably well in the average case (since a ray is more likely to first intersect the near primitives that are tested first). However, a ray may still be tested against a number of cells (and their contained primitives) before an intersection is found.

Arvo and Kirk [AK87] describe a subdivision of 5D ray space, that eliminates this costly traversal of cells. A 5D *cell* is defined by a parallelepiped of ray origins in 3D (that we will call the *origin box*) and a range of ray directions. It has a natural manifestation in 3D as a *beam*. If a beam does not intersect a primitive then no ray in the 5D cell can either. This property is used to find a relatively small *candidate set* for each cell.

A ray need only be tested against the candidate set of the cell that contains it. Classifying a ray requires at most one tree traversal (this is reduced further in practice by using a caching scheme). In the context of Arvo and Kirk [AK87], the purpose of the 5D subdivision is to function as a caching mechanism. The candidate sets are evaluated lazily, the premise being that the cost of the candidate set computation would be amortised over successive "cache hits" (i.e., successive rays with the same classification).

This method is very memory intensive due to the deep level of subdivision required before candidate sets are sufficiently small [SD94]. This is further exacerbated by the highly conservative nature of these candidate sets[7]. The implementation presented by Arvo and Kirk is conservative. Firstly, occluded primitives are not culled from the candidate sets see Figure 26), and secondly, the

[7] In the context of offline ray-tracing, this is the most cost-effective solution.

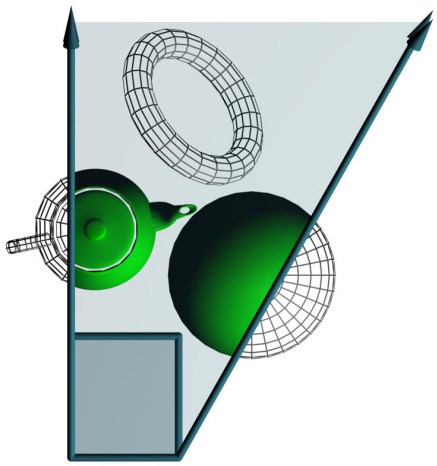


Figure 26: *Candidate Set Beam.* A beam that intersects many polygons, but few are possibly visible from the origin box due to occlusion. The polygons in the candidate set are solid, while polygons that are occluded or are outside the beam are shown in wire frame.

beam used is a conservative overestimate of the optimal beam. This results in primitives outside of the beam, being included as candidates.

Arvo and Kirk compensate for their conservative approach by *truncating* the candidate sets. Primitives that are further than a specified distance from the origin box are omitted. The beam is capped by a truncation plane and if a ray does not intersect any primitives in the candidate set, it is projected onto the truncation plane and reclassified. This can be effective in many cases, but the initial advantage of a single tree traversal and candidate set per ray is lost. It would be better to evaluate which primitives are occluded and remove them.

Our technique is similar to that of Gotsman and Sudarsky [GSF99], however, their subdivision scheme is optimised for visibility culling, rather than ray shooting[8]. Furthermore, their sampling does not exploit graphics hardware (potentially leading to greater compromises in either time or accuracy), nor does it effectively use the information of previously cast rays to minimise error.

4.4.2 5D Pre-processing

Our solution to the ray shooting problem is an adaptation of the aggressive visibility presented earlier in this chapter, that allows us to generate candidate sets for a 5D subdivision similar, to that of Arvo and Kirk. Using an aggressive technique results in an optimum candidate set being chosen

[8]Gotsman and Sudarsky subdivide directional space into only 8 components. Also, this subdivision only occurs around the azimuth.

for each 5D cell (with minimal false exclusion). All visibility determination is performed offline as a pre-process. The algorithm is summarised in Figure 27.

```
Generate initial set of cells V
Set primitive budget to threshold
for each c ∈ V do
   Subdivide( c )
next c

procedure Subdivide( c )
   Compute Candidate Set of c
   if ElementsIn(c) ≥ threshold then
      {c_-, c_+} ← Split(c)
      Subdivide( c_- )
      Subdivide( c_+ )
   end if
end procedure

procedure {c_-, c_+} ← Split(c)
   Choose 5D splitting hyperplane for c such that:
      ElementsIn(c_-) + ElementsIn(c_+) is minimised
      Return {c_-, c_+}
end procedure
```

Figure 27: *Ray-space Subdivision Algorithm*. The subdivision algorithm begins by generating an initial set of cells. These cells are then recursively subdivided until a threshold has been reached. The splitting routine attempts to maximise the separation of the two subcells.

The Initial Classification

Only those cells whose origins lie within the 3D bounding volume of the scene need be evaluated. For those rays with origins lying outside, but still intersecting the bounding volume, the origin can be moved to the point of intersection and reclassified.

We begin by constructing six cells. The origin box of these cells is the scene bounding box. To specify the angular bounds we note that the direction of any ray can be associated with its *dominant axis*, denoted $+X$, $-X$, $+Y$, $-Y$, $+Z$ or $-Z$. Using this association, Arvo and Kirk [AK87] define an isomorphism between the sphere of directions, and the surface of an axially aligned cube. Initially, it is convenient to partition the direction space by the six dominant axes (discussed in Section 4.4.2). This leads to the six initial cells.

Adaptive Subdivision

A spatial subdivision of the 5D ray space is used to accelerate ray shooting. At each level of the subdivision the size of the candidate set is reduced. At run-time, this reduces the computation cost of finding the first ray-primitive intersection by reducing the total number of required intersection tests.

Assuming a balanced hierarchy (roughly true in practice), the cost of ray intersection becomes $O(\log n + c)$, where n is the number of cells, and c is the user set candidate set maximum. The logarithmic traversal time, assumes that the tree has been balanced as a post-process. This is hugely beneficial, considering that in practice, $\log n$ is very small, and c may be chosen to be arbitrarily small. No other practical techniques limit the number of intersection tests prior to finding the first ray-primitive intersection.

Separation and Effectiveness

Separation refers to how the primitives are split and shared between the child candidate sets. It is used to measure the effectiveness of a subdivision. An effective subdivision will share very few primitives between the child cell and thus have high separation. The effectiveness of a subdivision is important for deciding how to subdivide a cell, and whether a given subdivision is beneficial.

Subdivision strategies

In this section we outline the heuristics that we used to guide our subdivision process.

- **Naïve subdivision** A subdivision strategy in which each dimension is divided in turn at each level, up to a specified maximum depth, is the simplest to implement and test. It produces a complete k-D tree, where every branch has the same height.
- **Maximum Candidate Set Size** The naïve subdivision strategy is improved by changing the terminating criteria so that subdivision stops early when the candidate set size drops below a specified user maximum. A maximum depth limit is also provided to stop subdivision from continuing indefinitely. Assuming the maximum depth is never reached (true for a vast majority of cells), this will bound candidate set size. This produces a k-D tree that is roughly balanced in practice.
- **Most Separated Dimension** This strategy involves looking at the set of all possible subdivisions and choosing the one that gives the best separation. An optimal solution for this is

extremely expensive. However, aggressive visibility sampling can also be used to perform this process. A limited sampling of the five possible splitting hyperplanes is used to approximate the separation (a similar approach is used by Gotsman *et al.* [GSF99]). The subdivision with the most separation is then chosen and fully sampled. This produces an axis aligned binary space partition tree.

Optimisations

For a given scene the cost of the preprocess is largely related to the number of samples taken. We employ a number of simple optimisations to decrease this overhead:

- The candidate set for a child cell is always a subset of the candidate set of its parent. Therefore, to determine visibility for a child cell, only the primitives visible in the parent cell need to be rendered while sampling. Fewer primitives results in faster rendering of visibility samples at deeper levels of the subdivision (see also Section 4.2.2).
- For any cell, the list of pertinent samples necessary for determining visibility is stored with the cell. These samples are reused in child cells, if possible. A sample can only be reused if the current subdivision does not cross its bounds (see Section 4.4.2 and 4.4.2). Reuse can be increased by storing more that just a visibility list with each sample, for example, by breaking the rendering into a grid of lists. The reuse of samples drastically reduces the sampling cost, since only the splitting surface needs to be sampled for any subdivision (see also Section 4.2.2).
- Caches are also implemented to facilitate the reuse of samples across subdivision branches. This allows samples to be reused on neighbouring cells that do not share a parent (that are in separate branches of the tree). All requests for samples are directed through the cache. A sample is created if it does not already exist. Samples are eliminated from the cache when they are no longer needed (see also Section 4.2.2).

Constructing Candidate Sets

The candidate set for a 5D cell is the union of the primitives that intersect the origin box and the primitives visible by rays that originate from the surface of the origin box, with direction bounded by the angular range of the cell.

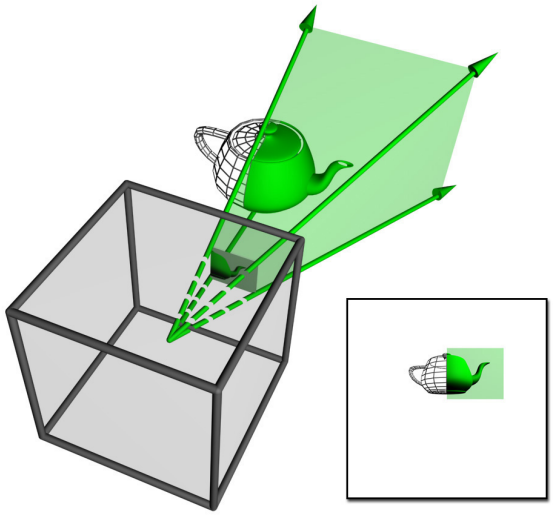


Figure 28: *5D Point Sample.* A visibility cube with point-sample frustum restricted to the subset of angles required (uv bounds $(0, 0)$ to $(0.6, 0.5)$). The inset shows a typical rendering for a full face of a direction cube. The shaded region is all that need be rendered for a restricted point-sample.

In order to determine what is visible from a surface we sample the set of rays that can originate from that surface. This is done using two complementary methods: *point-samples* and *area-samples*. For a point-sample, we sample many rays originating from a single point on the surface (see Section 4.1.1). Conversely, for an *area-sample*, all the rays sampled are parallel, but have a differing origins (see Chrysanthou *et al.* [CCOL98]). Both can be created using graphics hardware. The details are as follows:

Point-samples

Within the context of ray space, the desired point-sample is a restricted visibility cube. The initial six way subdivision of ray space means that only one face of the visibility cube need be rendered for any sample. Furthermore, the two angular bounds restrict the portion of the face that has valid samples. To ensure that the entire rendering is useful, the point-sample is created by rendering the scene using a perspective projection, where the frustum reflects the angular bounds of the 5D cell. This is illustrated in Figure 28.

In our implementation, each point-sample reveals what is visible from a point over the full range of the angular bounds of a 5D cell. If this set of visible primitives is manipulated as a single set of items, one cannot determine in which direction a particular primitive in the set is visible, but merely

that it is visible in some direction from the point-sample origin. This has efficiency implications, since a subdivision of one of the angular dimensions requires that all existing point-samples must be discarded. One cannot reuse the sample in this case because the direction information has not been preserved.

The reuse of samples greatly accelerates the performance of this technique, especially for large scenes. Having to discard a sample is an expensive operation. We reduce the effect of this by breaking up a point-sample into a number of sets, each representing an angular region of the rendering. This is done by splitting the rendering into a grid of separate samples on the visibility cube. When an angular subdivision takes place the relevant lists are copied into a new sample. Although this reduces the problem to a certain degree, it does not remove it. The only way to do so, is to store the entire rendering for each point-sample. Even with image compression, the memory requirements for this would be excessive.

Point-sampling results in a large number of directional samples for a small number of origins. Improving the coverage of sampled origins, involves the evaluation of more point-samples. An adaptive sampling of the surface is used to achieve this. The four corners of the surface are sampled and then further samples are taken recursively in the manner of a quad-tree. Termination is governed by an error minimising heuristic.

We use a heuristic similar to that described in Section 4.1.3. Although this heuristic was developed for full directional visibility cubes for determining from-region visibility, we use it with almost no adaptation for our point-samples[9]. This heuristic uses an image based similarity measure between adjacent samples. The user defines the error threshold. When the number of common items is above the threshold then subdivision stops.

The angular restriction to the visibility cube introduces a number of sampling problems, illustrated in Figure 29. Sampling can leave significant portions of the scene un-sampled. Primitives in these portions will be categorised falsely as invisible if the terminating condition is satisfied for the outer samples. This problem arises because all of the rays sampled by the point-samples are clustered around a few key origins. The problem is further exacerbated when the angular bounds are small, since the resulting gaps become larger. It is particularly significant since the un-sampled regions are near the origin box surface.

In order to improve the coverage of the ray samples and to avoid the sampling gaps that are created by point-sampling, an alternative sampling methodology, termed *area-sampling* is also used.

[9]The contribution that each polygon makes to the error measure is scaled by the number of sample rays that intersect it.

Area-samples

Rather than sampling a range of ray directions from a single origin, an area-sample shoots parallel rays from a range of origins across the surface. This is achieved through a generalisation of the restricted visibility cube used for point-samples. Point-sampling uses perspective projection, with the centre of projection on the surface and the front clipping plane very near to the surface. The area-samples use an orthographic projection in the specified direction with the sampling surface itself as the front clipping plane.

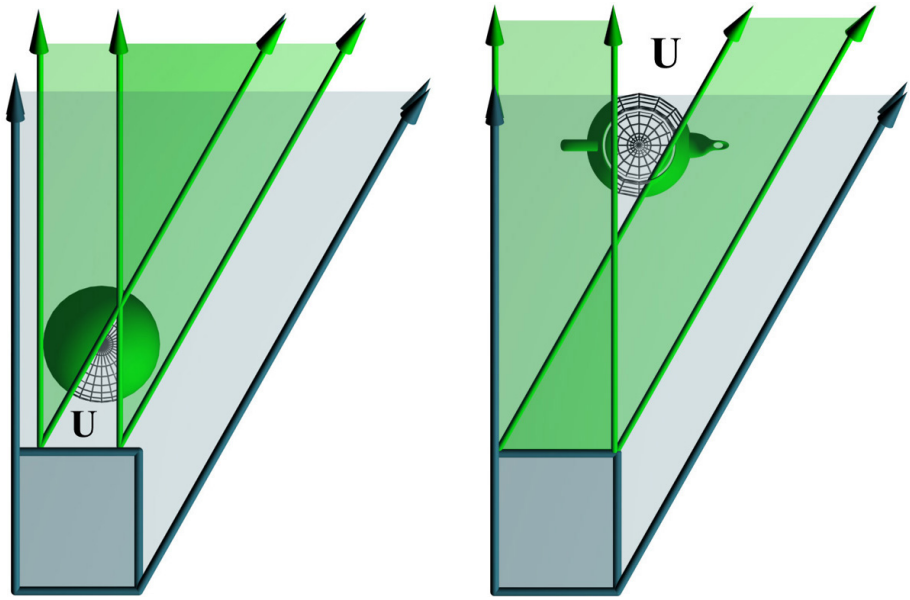
We adaptively sample visibility from the surface starting with four samples at the four angular extents. However, the original termination heuristic leads to unnecessary oversampling on the four surfaces of the origin box that are orthogonal to the dominant direction. This is because the orthographic projection includes a shearing operation used to achieve the desired ray direction. This shearing is particularly significant for area-samples taken from these surfaces and can stretch insignificant polygons such that they contribute too much to the error measure. Scaling the polygons contribution by the shearing factor produces a better weighted sampling.

Area-samples and point-samples have complementary properties. Where point-samples cannot be reused after splitting an angular dimension, area-samples cannot be reused after splitting the x, y or z dimension. The reason is the same: the only information kept with a sample is the list of visible primitives.

There are also sampling problems with area-samples (see Figure 29). Where point-samples have un-sampled regions close to the origin box surface, area-samples leave more remote regions un-sampled. It is also evident that point-sample gaps are larger in scene volume, when the sampling surface is large, whereas area-sample gaps are larger when the sampling surface is small.

Because of their complementary nature, the best solution uses a combination of point-samples and area-samples. The nature of the sampling problems of each sample type suggests that more area-samples should be used in the early stages of subdivision when the sampled cell surfaces are large. More point-samples should be used deeper in the hierarchy, when the cell surfaces are smaller.

Uniform sampling of origin box surfaces is required for efficiency. Since child cell samples are based on their parent cell's visibility, it does not prove useful to adaptively sample a surface to further depth than the parent did. No extra information can be recovered. Similarly, it is not useful to render the item buffers at the same or higher resolution for the child cells. Instead, the maximum adaptive sampling depth and the sampling resolution can be reduced as the subdivision takes place. This ensures that no time is spent rendering more samples than are absolutely necessary for visibility determination.



(a) (b)

Figure 29: *Un-sampled Space.* Sampling problems with both point-samples (a) and area-samples (b). The polygons in the un-sampled regions (U) will be classified as invisible (rendered in wire frame).

4.4.3 Preliminary Results

The techniques presented in this section have been fully implemented. A number of scenes were processed using a dual Pentium 4 1.7Ghz with 1.2GB of RAM and an NVidia G-Force 4 Ti 4600 graphics card. We have evaluated the performance of the preprocessor, the accuracy of the visibility information and the degree of ray shooting acceleration.

Performance

The length of the preprocess depends mainly on the total number of samples that are rendered. This number is directly related to the user specified *maximum cell depth* and *maximum sample depth* parameters. We use both the Maximum Candidate Set Size (A) and the Most Separated Dimension (B) strategies (see Section 4.4.2). Table 5 and 6 give the preprocessing parameters and results, respectively, for a number of trials using three different scenes.

Both subdivision strategies effectively reduce the average and maximum number of triangles for

	Scene	Size	Cell Depth	Sample Depth	Sub. Strategy
1	roomsbig	45k	10	6	A
2	roomsbig	45k	15	6	A
3	rooms	5k	10	6	B
4	roomsbig	45k	15	6	B
5	hilltop	485k	12	6	B

Table 5: *5D Preprocessing input parameters.* A is the Maximum Candidate Set Size subdivision strategy while B is the Most Separated Dimension strategy (see Section 4.4.2).

	Time	Cells	Tri/Cell		Cell Depth		Sample Depth		File Size
	h:mm		Ave.	Max.	Ave.	Max.	Ave.	Max.	(MB)
1	0:25	5263	625	4944	9.9	10	4.7	6	29
2	1:51	82192	318	2312	14.5	15	5.0	6	143
3	4:37	1514	402	1122	9.0	10	5.6	6	2.4
4	27:25	12172	695	2348	13.0	15	5.1	6	29.8
5	38:43	17972	535	14267	11.9	12	5	6	93.0

Table 6: *5D Ray-space Pre-processing results.*

each candidate set. Trial 5 successfully processed a large scene and reduced the average number of triangles per cell to 0.11% of the original scene size. The maximum number of triangles for a cell was 2.94% of the scene size.

Subdivision strategy B took significantly longer because of the extra processing involved in choosing the splitting hyperplane. However, this increased pre-processing time is balanced by a reduction in pre-processed data file size. The maximum file size was 93Mb, only 2.5 times larger than the scene file. This is compared to Strategy A, whose maximum file size was 50 times bigger than the scene file. We have yet to integrate an effective compression scheme [vdPS99].

Accuracy

The accuracy of the preprocessor is evaluated by comparing images produced using the pre-processed visibility files against images produced using a reference renderer. For our evaluation the reference renderer used a bounding volume hierarchy (BVH) [Hav00] to find the closest intersection. The reference ray tracer produces an exact image. We calculate the number of incorrect intersections using our 5D visibility information. Table 7 shows the accuracy statistics of the trials.

Trial 1 and 2 produce acceptable rates of error. Trial 5 had a higher average error rate of 3.65%, but this is acceptable considering the size of the scene and the depth of the pre-process. The maximum error rate was 82.19%, a substantial proportion of the image. This image, and most of the

	Min.	Max.	Ave.	St. Dev.
1	0.00%	1.79%	0.03%	0.17%
2	0.00%	9.58%	0.22%	0.98%
5	0.00%	82.19%	3.65%	10.78%

Table 7: *Error rate statistics for ray traced images.*

		5D		BVH	
	Size	Ave. Int/Ray	× Speedup	Ave. Int/Ray	× Speedup
1	45k	594.94	75.51	40.46	1112.21
2	45k	126.20	356.57	40.46	1112.21
4	45k	200.59	224.33	40.46	1112.21
5	485k	290.34	1670.45	61.96	7827.63

Table 8:
Ray shooting acceleration statistics.

others with error rates above the average, had view-points near a falsely excluded polygon. This polygon counted for the majority of the error. It should be noted that this type of error is very infrequent. The maximum error for this scene is nearly 7.3 standard deviations from the mean.

Ray Shooting Acceleration

The success of the technique for accelerating ray tracing depends directly on its ability to reduce the number of intersection tests required. Table 8 highlights the performance of the technique. The × *Speedup* column is relative to the naïve approach (intersecting every ray with every primitive). The speedup using our technique is also compared to the speedup using a bounding volume hierarchy (BVH) column.

For all scenes the number of intersections per ray was vastly reduced compared to a naïve approach to ray shooting. 5D pre-processing successfully reduced the size of the candidate set to a manageable level.

The traversal of the 5D tree requires one comparison per node, thus with a maximum overhead of 15 floating point comparisons, the candidate size can be reduced to as little as 0.1% of the total scene size and consequently reduce the number of required intersections to as little as 0.03% of the total scene size. These are promising results, considering that a hybrid of this technique with a BSP tree or bounding volume hierarchy offers the potential for substantial further reductions.

4.4.4 Conclusion (5D Subdivision)

We have presented a novel approach to accelerate ray shooting. An offline 5D subdivision pre-process reduces the candidate set for a ray to a manageable size based on both visibility and occlusion. The results (in Table 6) reflect success in bounding both maximum and average ray intersections per ray.

Indeed, we effectively limit the number of intersections per ray to at most 2.5% of the scene size (but on average, less than 1%). This is at the cost of a small error: on average, less than 4% of rays return the incorrect primitive on very large scenes. For the smaller scenes, the average error rate is closer to 0.3% (Table 7). Furthermore, our technique completes in practical time, even for large scenes.

4.5 Parallelising Hardware Accelerated Visibility

We have investigated the parallelisation of the visibility algorithm presented in this chapter on a cluster of machines. The key idea is to distribute the pre-process onto multiple machines in order to benefit from the graphics hardware on each machine.

Parallelising this algorithm presents a challenge. Effectively utilising the sample cache in such an environment is non-trivial. We have considered various models: a supervisor-worker architecture, a distributed supervisor architecture, and a brute-force sweep architecture (most suited to a cluster of low-end machines). Each have their own benefits and costs.

For a full account, see Oberholster *et al.* [OLBN01a] and Oberholster and Lewis [OLBN01b].

4.6 Hardware Extensions

There are two groups of hardware extensions that we propose to accelerate our visibility algorithm. The first group focuses on improving *rendering performance*, while the second group focuses on solving the frame-buffer read bottleneck.

4.6.1 Accelerating Rendering

For the purposes of our algorithm, we list the following features that are required for maximum performance:

- A 24 or 32 bit Z-buffer

- A 24 or 32 bit frame-buffer
- The fastest rasterisation possible
- Occlusion/contribution testing should also be included
- Fast hardware vertex transformations
- Support for storing geometry on the video card (preferably) or AGP memory

Contemporary graphics hardware has many feature (programmable pipelines, anti-aliasing, etc.) that are not required by our technique. All aspects relating to improved image quality can be stripped from the hardware. This can save on die space, heat maintenance and cost.

Unlike standard rendering, the visibility algorithm is roughly predictive. It is possible to build a queue of camera positions and directions that will need to be rendered. This implies that multiple graphics pipelines can be used to parallelise the algorithm. Given the simplicity of the required hardware, it may be possible to integrate these pipelines on a single graphics card, where each graphics processing unit (GPU) shares the same geometric data for efficiency.

4.6.2 Accelerating Buffer Feedback

The maximum rate that pixels can be read from a graphics card on standard PC hardware is approximately 46 million pixels per second. This limitation is due to the current bus architecture standards, not the graphics hardware.

The bus architecture required to achieve accelerated item buffer reads is already in a prototyping phase. Intel Corporation have developed a technology called *PCI Express* [Int03]. This architecture will replace the Advanced Graphics Port (AGP). The first iteration of the new PCI architecture is faster than the AGP $8\times$ standard. More importantly, however, is that unlike its AGP counterpart, this high bandwidth is bidirectional.

4.7 Conclusion

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

A kd-tree is constructed in a top down fashion. Adaptive sampling is performed on the surfaces of the kd-tree nodes to obtain the set of visible geometry. This sampling exploits graphics hardware

to enhance performance. An efficient scheme is used to share samples between cells and to minimise the memory resident lifespan of these samples. We have developed an efficient heuristic to guide the sampling process with the goal of minimising error.

Results show this technique to be significantly faster than existing techniques. Also, we show that this technique can be used to preprocess even the most complex of scenes effectively. Image error is minimal.

We extend this technique to a 5D sampling of ray-space. A 5D kd-tree is built in a top-down fashion, attempting to limit the number of triangles in each node. At run-time this information can be used to accelerate the shooting of rays for general scenes.

Preliminary results show the 5D version of the algorithm to be more prone to errors than its 3D counterpart. In most cases though, the error is small. The number of ray-triangle intersections required per ray is between 1% and 2.5% of those required by a naïve algorithm.

Chapter 5

The Selective Stabbing Problem

A difficult problem that we have encountered in our research is that of determining *exactly* whether or not one polygon can be seen from some viewpoint on another polygon. In the next Chapter we show how that such a solution can be used as a tool to solve for the exact set of polygons visible from a region.

A natural generalisation of the stabbing problem [TH93b] is a problem we term the *selective stabbing* problem. In Chapter 6 we show how this generalisation may be cast as an exact polygon to polygon visibility *query*.

The general problem asks one, or both, of the following questions:

1. Given two sets S and M of convex polygons. Does there *exist* a line that stabs *all* polygons in S and misses *all* polygons in M.

2. Given two sets S and M of convex polygons. Compute a representation of the set of lines that stab *all* polygons in S and miss *all* polygons in M.

In this chapter we present an efficient solution to the *selective stabbing* problem. The general problem and solution can be expressed outside the context of visibility. We discuss it in such a manner. In the paper by Nirenstein *et al.* [NBG02], the exact visibility algorithm is presented without the generalisation of the selective stabbing algorithm. The work in this chapter supersedes the theory component of this paper.

The term *stab* refers to incidence and *miss* refers to non-incidence. An answer to Question 2 implicitly answers Question 1, in that such a line must exist if and only if the set of lines requiring representation is non-empty. We refer to lines in the solution set of Question 2 as *stab-miss* lines. Figure 30 illustrates the principle of selective stabbing.

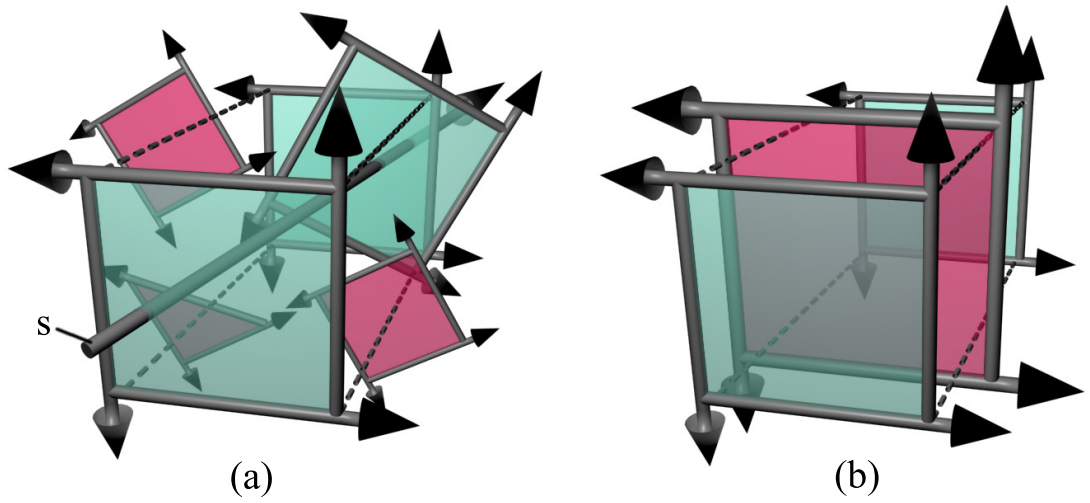


Figure 30: *Selective Stabbing*. The cyan polygons correspond to those elements in the S ("must hit") set, while the magenta polygons correspond to those in the M ("must miss") set. The problem is to show the existence of a line that stabs *all* elements of S, and misses *all* elements of M. (a) shows a case where (by example), a *stab-miss* line (s) must exist. (b) shows a case where clearly no *stab-miss* line can exist.

The classical stabbing problem can be cast as the special case where $M = \emptyset$. Teller and Hohmeyer [TH93b] present a solution for stabbing oriented convex polygons. We too make the assumption that all the polygons in S are oriented, thereby admitting stabbing lines oriented in only one direction. Blocker polygons may be used to block stabbing lines in one *or* both directions. To achieve the latter, each blocker polygon can be treated as two blockers that are identical, except for their orientations.

Our solution consists of two major phases. The first being the construction of the set of stabbing lines for the set S. These stabbing lines are represented as a polytope in Plücker space (see Section 3.1.5).

In the second phase, the set of stabbing lines is "trimmed" by the set of blockers M. Using the Plücker space polytope representation, we show how constructive solid geometry may be used to achieve this trimming.

Finally, if the whole polygon is trimmed away, then the stab-miss line set is empty. If the whole polytope is not trimmed away, then the remaining structure is intersected with the Plücker hypersurface. If no intersection exists, then again the stab-miss line set is empty. If an intersection does exist, then the intersection points can be mapped to the (non-empty) stab-miss set.

5.1 Phase 1: Constructing the Set of Stabbing Lines

The existence of a stabbing line through a single (non-degenerate) polygon is clearly the affirmative. Similarly, a stabbing line through two (oriented) polygons must exist, unless the angle between the normals of the two polygons is obtuse. For the case of $S = \emptyset$, the result is true or false depending on the problem definition.

The computation of the *set* of stabbing lines in the case where $|S| < 3$ is a special case. For completeness, in this section we present a general solution for $|S| \geq 3$ similar to that of Teller and Hohmeyer [TH93b]. We also give a brief discussion of the reasons for the cases $|S| < 3$ creating such difficulties.

We are most interested in $|S| = 2$, since in Chapter 6 we shall see that visibility queries always involve selective stabbing where $|S| = 2$. We therefore provide a rigorous exposition of the problems encountered in this case and show how they can be solved efficiently. We present a specialised algorithm for the computation of a line-space representation for this case that is simpler, and more efficient than for the general case. In order to achieve this, we derive and exploit several novel mathematical results for the two-polygon case.

The notation used in this chapter adheres to that of Chapter 3.

5.1.1 Computing the Set of Stabbing Lines for $|S| > 2$

Ideally, we would like a generic representation of the set of stabbing lines for a set S. This set should be 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 set. The intersection of this polyhedron with the Plücker hypersurface gives the desired set of points. This polyhedron may be constructed as follows:

For a line s to stab a set S, it is necessary and sufficient that all edges from the polygons of S pass by a directed version of s with the same relative orientation. This is illustrated in Figure 31. 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:

$$D_\pi(\Pi(s)) \geq 0, \forall \pi \in \{\Pi(e_1), \Pi(e_2), \ldots, \Pi(e_n)\} \tag{13}$$

We call each one of these constraints, an *edge constraint* of the set of polygons S.

The projective space $\mathbb{P}^5$ is an augmentation of $\mathbb{R}^5$ by two additional hyperplanes at positive and negative infinity (see Section 3.1). These additional hyperplanes are useful as an elegant means to

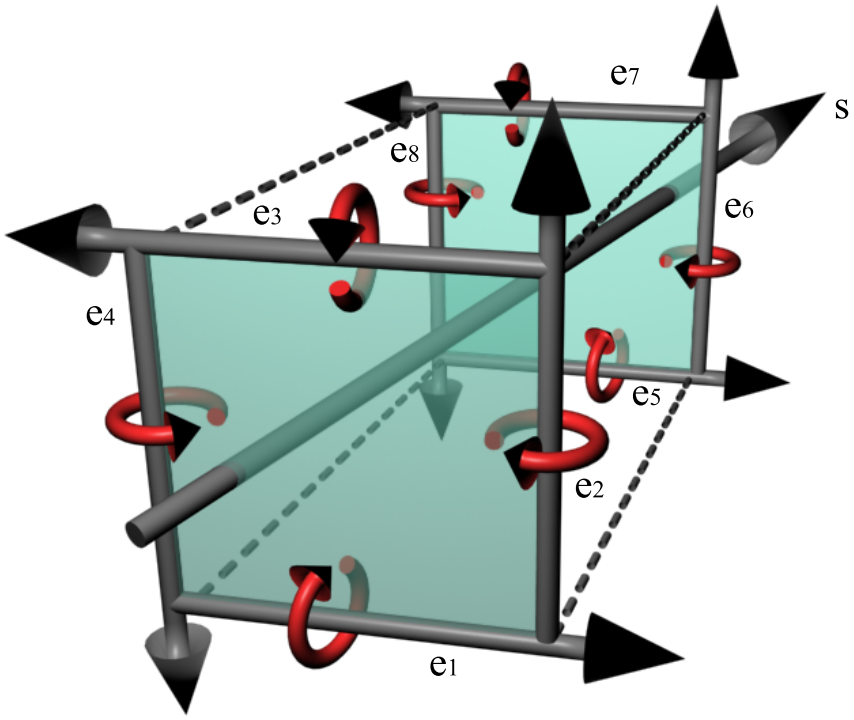


Figure 31: *Stabbing constraints*. Directed lines e_1 to e_8 correspond to the edges of two quadrilaterals (the set S). Note that the stabbing line s passes all e_1 to e_8 to the left. This condition is both necessary and sufficient for stabbing.

handle uniformly those cases that 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 a 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 from occurring (i.e., to ensure that all stabbing lines can still be represented). This process is equivalent to projecting down to $\mathbb{R}^5$. This approach was also taken by Teller and Hohmeyer [TH93b].

After this projection, the function $D_\pi(x) : \mathbb{P}^5 \rightarrow \mathbb{R}$ becomes $D'_\pi(x) : \mathbb{R}^5 \rightarrow \mathbb{R}$ where $D'_\pi(x) = \pi_0 x_3 + \pi_1 x_4 + \pi_2 x_5 + \pi_3 + \pi_4 x_1 + \pi_5 x_2$. In this case, the first element has been selected as the normalisation component, and thus $x_0 = 1$ and may be omitted. The six tuple that was $x = (1, x_1, x_2, x_3, x_4, x_5)$ becomes the tuple $x = (x_1, x_2, x_3, x_4, x_5)$ in $\mathbb{R}^5$. Where the old form of the equation $D_\pi(x) = 0$ $(x \in \mathbb{R}^6)$ defined a hyperplane through the origin in $\mathbb{R}^6$, $D'_\pi(x) = 0$

($x \in \mathbb{R}^5$) defines a less restricted hyperplane in $\mathbb{R}^5$. Using D' rather than D, the solution space for the constraints of a stabbing line s given in Equation 13 may be used to define a volume in $\mathbb{R}^5$. This volume completely represents the set of stabbing lines for the set S. This construction and mapping is depicted in Figure 32a and Figure 32c.

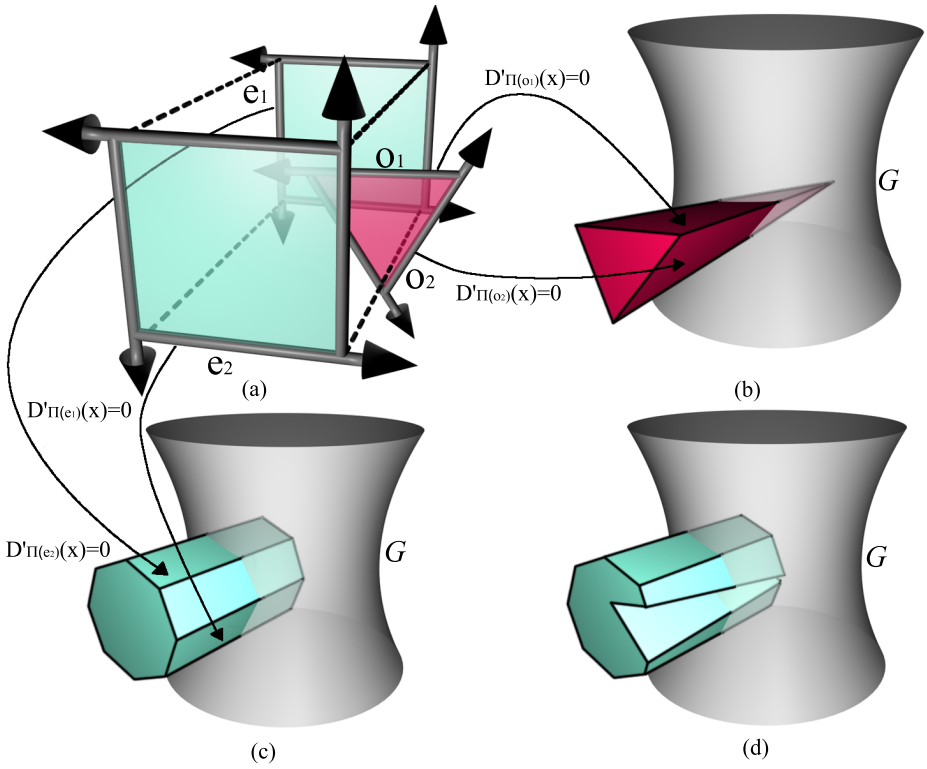


Figure 32: *Plücker-complex Construction*. (a) A typical setup between two polygons (the quadrilaterals), and one occluder (the triangle). (b) A visualisation of the mapping of the edges of the triangle occluder to a volume in $\mathbb{R}^5$. Again, G is a visualisation of the Plücker hypersurface. (c) 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. The remainder fully represents the set of lines between the quadrilaterals that *miss* the occluding triangle.

Definition 5.1 *The* stabbing polyhedron *of a set of convex polygons S, is the set of points (in Plücker coordinates) satisfying the set of constraints in Equation 13, where $e_1, e_2, \ldots, e_n$ are the edges of the polygons in S.*

There are various techniques that may be used to extract a polyhedral representation from several half-space intersections. We used a version of the double description method [AF92] to extract the

extreme points. With this and the hyperplane information, we construct the full face lattice/graph of the polyhedron using a combinatorial face enumeration algorithm similar to that presented by Fukuda and Rosta [FR94].

The technique presented in this section is valid for any cardinality of S. What separates the cases $|S| < 3$, is that the half-space intersections do not form a polytope, rather they define a convex, necessarily *unbounded* polyhedron. We discuss this next.

5.1.2 Unboundedness in the Set of Lines Through One Polygon

The case where $|S| = 1$ results in an unbounded stabbing polyhedron in $\mathbb{R}^5$. This can be shown easily: consider any (non-degenerate) polygon p. Now consider a stabbing line ℓ of p, intersecting p at a point q. ℓ can be swivelled at q with two directions of freedom and still remain a stabber of p. This implies that those components of $\Pi(\ell)$ corresponding to direction (the first two/three components depending on the nature of the projection to $\mathbb{R}^5$) can be chosen arbitrarily. The constraints imposed by p are therefore unable to bound $\Pi(\ell)$ in line-space, thereby demonstrating the unboundedness of the stabbing polyhedron of S.

5.2 The Set of Lines Though Two Polygons

In this section we investigate the structure of the Plücker polyhedron of a set S, where $|S| = 2$. We have developed a method that obtains a bounded representation of this line set in worst case optimal time.

5.2.1 Proof Outline

We begin by proving a lemma characterising those lines that are *extremal* stabbing lines. These lines correspond to the intersection points of the boundary edges of the stabbing polyhedron of S with the Plücker hypersurface.

These intersection points can be found by computing the dual of the (now characterised) extremal stabbing lines. In order to compute the complete stabbing polyhedron, it is sufficient to find the extreme points of the polytope.

We prove that no such points exist by showing that each boundary edge of the stabbing polyhedron is in fact an infinite line. This is achieved by proving that all the polytope boundary edges are parallel. This is true for all pairs of polygons, however, for simplicity we only show this for

a special case. The special case suffices, since all configurations can be transformed to this case (McKenna and O'Rourke [MO88] use a similar argument).

In order to prove parallelism, we show how the direction vectors of these boundary lines (in $\mathbb{R}^5$) can be computed explicitly. We then show that these directions are the same.

Next, we show that any point within the polytope may be expressed using the Minkowski-Weyl theorem[1]. Using this representation, we characterise those points that lie on the Plücker hypersurface. This leads to a bound on the intersection between the stabbing polyhedron and the Plücker hypersurface. This bound may then be used to compute a Plücker polytope that has the same intersection with the Plücker hypersurface as its unbounded superset.

Using these results, we give an $O(mn)$ worst case optimal algorithm. m and n are the number of vertices in the two respective input polygons. This enumerates the extreme rays of the stabbing polyhedron and then caps it.

5.2.2 Details

Firstly, we assert that the general construction (as in Section 5.1.1) can indeed be unbounded (this is evidenced for any pair of convex polygons that lie within distinct planes). This is counter-intuitive, since the argument used for $|S| = 1$, in Section 5.1.2, does not hold. Given a set consisting of two *distinct* polygons S where $|S| = 2$, it appears to be impossible to choose any of the components of the Plücker mapping of a stabbing line arbitrarily, without violating one of the edge constraints. This might, at first sight, hint at a consistently bounded representation for the set of stabbing lines of S. However, solving for the solution space of the edge constraints of S always gives an unbounded polyhedron.

We begin with a characterisation of the set of extremal stabbing lines (see Section 2.5.2) generated by two polygons.

Lemma 5.1 *Given two convex polygons A and B, where A consists of n edges, and B of m edges, (1) there are exactly $n \times m$ extremal stabbing lines between A and B, (2) these stabbing lines correspond to every vertex-vertex combination between A and B.*

Proof: A stabbing line is extremal *iff* it is constrained (incident) on four edges. Because A and B are convex polygons, they can each contribute at most two edges. Since the edges of A are coplanar, the intersection point of the two contributing edges must be a vertex of A. Similarly, the intersection point on B must be a vertex of B.

[1] I.e., as a convex combination of extreme vertices and rays.

We note that there are n vertices in A and m vertices in B. We also note that there are $n \times m$ possible combinations of A to B vertex-vertex pairs. Every such combination of vertices (equivalent to a combination of four edges) gives an extremal stabbing line, and every extremal stabbing line must be generated by a pair of vertices from A to B.

□

In Section 3.5.2, we see that four lines in general position generate exactly two lines by incidence. Let us consider an extremal stabbing line of polygons A and B. This line is determined by four edges, and is simply the line through the vertex defined by the intersection of one edge pair of A, and the vertex defined by the intersection of another edge pair of B.

We note in passing, that by extending these edges to lines, we find a second line is incident on the four lines, namely the line k, where the plane of A intersects the plane of B. This line is incident on every line in the plane of A and every line of the plane of B (projectivity handles the incidence of parallel lines arising from parallel edges).

Selective stabbing is by definition invariant under invertible affine transformation[2]. The existence of a stab-miss line s, for a stab set S and miss set M is equivalent to that of the stab set $T(S)$ and miss set $T(M)$, where T is comprised of only a rotation and a translation. Similarly, given the stab-miss set P for stab set $T(S)$ and miss set $T(M)$, the stab-miss set for S and M, is simply the pre-image $T^{-1}(P)$.

We continue our proof for the case where exactly one of the two polygons of the stab set S is embedded in the $x = 0$ plane. We note that *any* scenario can be transformed into this case using a consistent translation and rotation of the geometry. It is therefore sufficient to prove consistent unboundedness for this case.

Theorem 5.1 *Given two distinct lines ℓ_1 and ℓ_2 in the primary plane $x = 0$ (in $\mathbb{R}^3$) and two distinct lines ℓ_3 and ℓ_4 in a different plane B (also in $\mathbb{R}^3$), the direction $\vec{r}$ of line $p + \lambda\vec{r}$ in $\mathbb{R}^5$ defined by $p + \lambda\vec{r} = \{x \in \mathbb{R}^5 : D'_{\ell_1}(x) = 0$ and $D'_{\ell_2}(x) = 0$ and $D'_{\ell_3}(x) = 0$ and $D'_{\ell_4}(x) = 0\}$ is a function only of the plane B.*

Proof: Denote the intersection point of ℓ_1 and ℓ_2 as a, and denote the intersection point of ℓ_3 and ℓ_4 as d. Let b, c, e and f be points on ℓ_1, ℓ_2, ℓ_3 and ℓ_4 respectively, such that $a \neq b$, $a \neq c$, $e \neq d$ and $f \neq d$.

$p + \lambda\vec{r}$ must lie within each of the four hyperplanes of it's definition. This implies that the vector

[2] Since collinearity is preserved.

$\vec{r}$ must be orthogonal to the normals of the four defining hyperplanes. Using the generalised cross product (see Section 3.5.1) and the plucker dual hyperplanes, we can express $\vec{r}$ as follows:

$$\vec{r} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} & \vec{l} & \vec{m} \\ 0 & 0 & 0 & b_y - a_y & b_z - a_z \\ 0 & 0 & 0 & a_y - c_y & a_z - c_z \\ d_z e_x - d_x e_z & d_x e_y - d_y e_x & e_x - d_x & e_y - d_y & e_z - d_z \\ f_z d_x - f_x d_z & f_x d_y - f_y d_x & d_x - f_x & d_y - f_y & d_z - f_z \end{vmatrix} \tag{14}$$

Where rows $n = 2, 3, 4$ and 5 correspond to the coefficients of $D'_{\Pi(\ell_{n-1})}(x) = 0$. The constant is ignored, since it is only the hyperplane normals that are required to define $\vec{r}$. Vectors $\vec{i}$, $\vec{j}$, $\vec{k}$, $\vec{l}$ and $\vec{m}$ correspond to the standard 5D Euclidean basis. The zeros in the top left of the determinant matrix are due to the assumption that ℓ_1 and ℓ_2 are in the plane $x = 0$. This results in $a_x = b_x = c_x = 0$.

Two levels of cofactor expansion gives the following for $\vec{r}$:

$$\begin{pmatrix} (d_x e_y - d_y e_x)(d_x - f_x) \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} - (f_x d_y - f_y d_x)(e_x - d_x) \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} \\ (d_z e_x - d_x e_z)(d_x - f_x) \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} - (f_z d_x - f_x d_z)(e_x - d_x) \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} \\ (d_z e_x - d_x e_z)(f_x d_y - f_y d_x) \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} - (f_z d_x - f_x d_z)(d_x e_y - d_y e_x) \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} \\ (d_z e_x - d_x e_z)(f_x d_y - f_y d_x) \begin{vmatrix} 0 & b_z - a_z \\ 0 & a_z - c_z \end{vmatrix} - (f_z d_x - f_x d_z)(d_x e_y - d_y e_x) \begin{vmatrix} 0 & b_z - a_z \\ 0 & a_z - c_z \end{vmatrix} \\ (d_z e_x - d_x e_z)(f_x d_y - f_y d_x) \begin{vmatrix} 0 & b_y - a_y \\ 0 & a_y - c_y \end{vmatrix} - (f_z d_x - f_x d_x)(d_x e_y - d_y e_x) \begin{vmatrix} 0 & b_y - a_y \\ 0 & a_y - c_y \end{vmatrix} \end{pmatrix} \tag{15}$$

By factoring and expanding, we obtain the following:

$$\vec{r} = d_x \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} \begin{pmatrix} f_x d_y + f_y e_x - f_y d_x - e_y f_x - d_y e_x + d_x e_y \\ e_z f_x - d_x e_z - f_x d_z - e_x f_z + d_z e_x + f_z d_x \\ d_x e_y f_z - d_x e_z f_y + d_y e_z f_x - d_y e_x f_z + d_z f_y e_x - d_z e_y f_x \\ 0 \\ 0 \end{pmatrix} \tag{16}$$

We digress to consider the structure of plane B. This plane can be identified by a unit normal $\vec{n}$, where $\vec{n} = (n_x, n_y, n_z)$ and a constant q in the standard way. Since point d lies on ℓ_3 and ℓ_4 and e lies on ℓ_3 and f on ℓ_4, these three points must lie in plane B. $\vec{n}$ may therefore be expressed in terms of these points:

$$\vec{n} \quad = \quad (n_x, n_y, n_z) \tag{17}$$

$$\vec{n} = \gamma((e-d)\times(f-d)) \tag{18}$$

$$\Rightarrow \vec{n} = \gamma \begin{pmatrix} (e_y-d_y)(f_z-d_z)-(e_z-d_z)(f_y-d_y) \\ (e_z-d_z)(f_x-d_x)-(e_x-d_x)(f_z-d_z) \\ (e_x-d_x)(f_y-d_y)-(e_y-d_y)(f_x-d_x) \end{pmatrix} \tag{19}$$

$$\Rightarrow \vec{n} = \gamma \begin{pmatrix} e_yf_z-e_yd_z-d_yf_z-e_zf_y+e_zd_y+d_zf_y \\ e_zf_x-d_xe_z-f_xd_z-e_xf_z+d_ze_x+f_zd_x \\ f_xd_y+f_ye_x-f_yd_x-e_yf_x-d_ye_x+d_xe_y \end{pmatrix} \tag{20}$$

Where γ is a normalisation constant. Similarly, it is possible to express q in terms of these points, by substituting in any of d, e or f into the plane equation of B. We choose point d:

$$q = n_xd_x+n_yd_y+n_zd_z \tag{21}$$

$$\Rightarrow q = \gamma(d_xe_yf_z-d_xe_zf_y+d_ye_zf_x-d_ye_xf_z+d_zf_ye_x-d_ze_yf_x) \tag{22}$$

Returning to our derivation of $\vec{r}$ (Equation 16), through substitution we can further simplify the expression to:

$$\vec{r} = \frac{d_x}{\gamma}\begin{vmatrix} b_y-a_y & b_z-a_z \\ a_y-c_y & a_z-c_z \end{vmatrix} \begin{pmatrix} n_z \\ n_y \\ q \\ 0 \\ 0 \end{pmatrix} \tag{23}$$

The direction of $\vec{r}$ is clearly dependent only on plane B since we fixed plane A. The scalar $\begin{vmatrix} b_y-a_y & b_z-a_z \\ a_y-c_y & a_z-c_z \end{vmatrix}$ is the x component of the cross product between two distinct vectors in the direction of lines ℓ_1 and ℓ_2 respectively. Since these lines are embedded in $x=0$ and distinct, the normal must be non-zero. The special case where $d_x = 0$ can cause problems. This is addressed in Section 5.2.6.

□

5.2.3 Implications

We consider each extremal stabbing line. As shown in Lemma 5.1, each extremal line generated by two polygons A and B originates from a vertex of A and terminates on a vertex of B. Each such extremal line is associated with a vertex pair and each vertex is associated with two edges. Since these are the two edges that meet at the vertex, each vertex pair is associated with four edges. Extending these edges gives four lines, two in the plane of A, and two in the plane of B.

As shown, the entire scene can be transformed by an affine function T such that polygon A lies within $x = 0$. Using Theorem 5.1, we see that the direction of each extremal line in Plücker space is the same, since this is a function of the plane of $T(B)$, and all edges of $T(B)$ lie within this plane. Given that the extremal rays are all parallel, we deduce that the volume defined by the dual Plücker half-space intersections is unbounded. Furthermore, we can deduce that the structure of the stabbing polyhedron is that of a 4D polyhedron, extruded in direction $\vec{r}$.

5.2.4 Constructing the Stabbing Polyhedron in Worst Case Optimal $O(nm)$ Time

In this section we present an efficient algorithm for constructing the stabbing polyhedron between two polygons. By *construction*, we refer to the explicit determination of the skeleton (0 and 1 dimensional elements) of the polyhedron.

General algorithms that enumerate vertices from a half-space description may also be used. Teller [Tel92a, TH93b] use the fact that the half-space intersection test is the dual problem of the convex hull problem, and solve the dual problem. Vertex enumeration may also be performed directly using the *reverse search* algorithm [AF92, AF96] or the *double description* method [FP96]. Our method uses the special structure of the stabbing polyhedron to perform enumeration more efficiently.

Additional enumeration of the higher dimensional faces may be performed to construct the full face lattice if desired (see Section 3.2.2). We use the full face lattice in Phase 2.

Given two polygons A and B, transform them using the affine transform T that rotates and translates A onto the $x = 0$ plane. Let V be the set of lines, where each line corresponds to a $T(A)$ to $T(B)$ vertex-vertex pair. If the number of vertices/edges in A and B are n and m respectively, then the cardinality of V is exactly nm (best and worst case).

Each line in V can then be mapped to Plücker coordinates. The points where the edges of the stabbing polyhedron intersect the Plücker hypersurface, is exactly the set $\Pi(V)$[3]. The direction $\vec{r}$ can be computed from any vertex-vertex pair within V (Theorem 5.1). Finally, the skeleton of the Plücker polytope is the set of lines $e_1, e_2, \ldots e_{nm}$, where $e_i = p_i + \lambda\vec{r}$, and p_i is the i-th point in $\Pi(V)$. Since this is a constant time operation on the elements of a set with cardinality nm, the algorithm has the worst case optimal complexity of $O(nm)$.

This result has the potential to accelerate the technique of Bittner [Bit02]. For each region, Bittner needs to construct a stabbing polyhedron for every occluder inserted into, or tested against, his occlusion tree. His current system uses an implementation of the general reverse search method.

[3]Since all lines in V are extremal stabbing lines.

Our experiments using Komei Fukuda's implementation of the double description method [FP96], shows the general approach to take approximately 4ms to enumerate for a quadrilateral and a triangle as input. Our direct construction is at least three orders of magnitude faster.

5.2.5 Capping the Stabbing Polyhedron

In order to cap the polyhedron we choose two capping planes (one for each open side of the polyhedron) with normal vector $\hat{r}$. What remains is to find two constants h^+ and h^- for the hyperplane constraints:

$$\hat{r} \cdot (x_1, x_2, x_3, x_4, x_5)^T \leq h^+ \tag{24}$$

$$\hat{r} \cdot (x_1, x_2, x_3, x_4, x_5)^T \geq h^- \tag{25}$$

such that these two constants result in hyperplane bounds for all those points that are incident on *both* the stabbing polyhedron and the Plücker hypersurface.

For convenience we define the coefficients of the normalised vector $\hat{r}$ as follows:

$$\hat{r} = \frac{1}{||\vec{r}||} \begin{pmatrix} n_z \\ n_y \\ q \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ 0 \\ 0 \end{pmatrix} \tag{26}$$

Let us consider a point a in polygon A, and a point b on polygon B. It is possible to represent any point on A by a convex combination of the n vertices of A:

$$a = \sum_{i=1}^{n} \alpha_i a_i \qquad \sum_{i=1}^{n} \alpha_i = 1, \alpha_i \geq 0 \tag{27}$$

Similarly, it is possible to represent any point on B by a convex combination of the m vertices of B:

$$b = \sum_{i=1}^{m} \beta_i b_i \qquad \sum_{i=1}^{n} \beta_i = 1, \beta_i \geq 0 \tag{28}$$

This implies that the line $\overline{ab}$ has Plücker coordinates of the form:

$$\Pi(a,b) = \begin{pmatrix} \sum_{i=1}^{m} \beta_i b_{x_i} - \sum_{i=1}^{n} \alpha_i a_{x_i} \\ \sum_{i=1}^{m} \beta_i b_{y_i} - \sum_{i=1}^{n} \alpha_i a_{y_i} \\ \sum_{i=1}^{m} \beta_i b_{z_i} - \sum_{i=1}^{n} \alpha_i a_{z_i} \\ \sum_{i=1}^{m} \beta_i b_{z_i} \sum_{i=1}^{n} \alpha_i a_{y_i} - \sum_{i=1}^{m} \beta_i b_{y_i} \sum_{i=1}^{n} \alpha_i a_{z_i} \\ \sum_{i=1}^{m} \beta_i b_{x_i} \sum_{i=1}^{n} \alpha_i a_{z_i} - \sum_{i=1}^{m} \beta_i b_{z_i} \sum_{i=1}^{n} \alpha_i a_{x_i} \\ \sum_{i=1}^{m} \beta_i b_{y_i} \sum_{i=1}^{n} \alpha_i a_{x_i} - \sum_{i=1}^{m} \beta_i b_{x_i} \sum_{i=1}^{n} \alpha_i a_{y_i} \end{pmatrix} \tag{29}$$

Normalising this expression, and mapping the result from $\mathbb{P}^5$ to $\mathbb{R}^5$ gives [4]:

$$\Pi'(a,b) = \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} \begin{pmatrix} \sum_{i=1}^{m}\beta_i b_{y_i} - \sum_{i=1}^{n}\alpha_i a_{y_i} \\ \sum_{i=1}^{m}\beta_i b_{z_i} - \sum_{i=1}^{n}\alpha_i a_{z_i} \\ \sum_{i=1}^{m}\beta_i b_{z_i}\sum_{i=1}^{n}\alpha_i a_{y_i} - \sum_{i=1}^{m}\beta_i b_{y_i}\sum_{i=1}^{n}\alpha_i a_{z_i} \\ \sum_{i=1}^{m}\beta_i b_{x_i}\sum_{i=1}^{n}\alpha_i a_{z_i} \\ -\sum_{i=1}^{m}\beta_i b_{x_i}\sum_{i=1}^{n}\alpha_i a_{y_i} \end{pmatrix} \tag{30}$$

In order to find an h^+ and h^- that will bound the Plücker dual of this arbitrary stabbing line, we substitute $\Pi'(a,b)$ into Equation 24:

$$\Pi(a,b)\cdot\hat{r} = \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} \begin{pmatrix} \sum_{i=1}^{m}\beta_i b_{y_i} - \sum_{i=1}^{n}\alpha_i a_{y_i} \\ \sum_{i=1}^{m}\beta_i b_{z_i} - \sum_{i=1}^{n}\alpha_i a_{z_i} \\ \sum_{i=1}^{m}\beta_i b_{z_i}\sum_{i=1}^{n}\alpha_i a_{y_i} - \sum_{i=1}^{m}\beta_i b_{y_i}\sum_{i=1}^{n}\alpha_i a_{z_i} \\ \sum_{i=1}^{m}\beta_i b_{x_i}\sum_{i=1}^{n}\alpha_i a_{z_i} \\ -\sum_{i=1}^{m}\beta_i b_{x_i}\sum_{i=1}^{n}\alpha_i a_{y_i} \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ 0 \\ 0 \end{pmatrix} \tag{31}$$

$$= \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} r_1 \left(\sum_{i=1}^{m}\beta_i b_{y_i} - \sum_{i=1}^{n}\alpha_i a_{y_i}\right) + \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} r_2 \left(\sum_{i=1}^{m}\beta_i b_{z_i} - \sum_{i=1}^{n}\alpha_i a_{z_i}\right) + \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} r_3 \left(\sum_{i=1}^{m}\beta_i b_{z_i}\sum_{i=1}^{n}\alpha_i a_{y_i} - \sum_{i=1}^{m}\beta_i b_{y_i}\sum_{i=1}^{n}\alpha_i a_{z_i}\right) \tag{32}$$

$$\leq \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} |r_1| \left(\sum_{i=1}^{m}\beta_i |b_{y_i}| + \sum_{i=1}^{n}\alpha_i |a_{y_i}|\right) + \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} |r_2| \left(\sum_{i=1}^{m}\beta_i |b_{z_i}| + \sum_{i=1}^{n}\alpha_i |a_{z_i}|\right) + \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} |r_3| \left(\sum_{i=1}^{m}\beta_i |b_{z_i}|\sum_{i=1}^{n}\alpha_i |a_{y_i}| + \sum_{i=1}^{m}\beta_i |b_{y_i}|\sum_{i=1}^{n}\alpha_i |a_{z_i}|\right) \tag{33}$$

$$< \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} |r_1| \left(\sum_{i=1}^{m}|b_{y_i}| + \sum_{i=1}^{n}|a_{y_i}|\right) + \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} |r_2| \left(\sum_{i=1}^{m}|b_{z_i}| + \sum_{i=1}^{n}|a_{z_i}|\right) + \frac{1}{\sum_{i=1}^{m}\beta_i b_{x_i}} |r_3| \left(\sum_{i=1}^{m}|b_{z_i}|\sum_{i=1}^{n}|a_{y_i}| + \sum_{i=1}^{m}|b_{y_i}|\sum_{i=1}^{n}|a_{z_i}|\right) \tag{34}$$

$$< \frac{1}{min_{i=1}^{m} b_{x_i}} |r_1| \left(\sum_{i=1}^{m}|b_{y_i}| + \sum_{i=1}^{n}|a_{y_i}|\right) +$$

[4] We note that $\sum_{i=1}^{n}\alpha_i a_{x_i} = 0$, since all points of A are assumed to exist in the $x = 0$ plane.

$$\frac{1}{min_{i=1}^{m} b_{x_i}} |r_2| \left(\sum_{i=1}^{m} |b_{z_i}| + \sum_{i=1}^{n} |a_{z_i}| \right) +$$
$$\frac{1}{min_{i=1}^{m} b_{x_i}} |r_3| \left(\sum_{i=1}^{m} |b_{z_i}| \sum_{i=1}^{n} |a_{y_i}| + \sum_{i=1}^{m} |b_{y_i}| \sum_{i=1}^{n} |a_{z_i}| \right) \quad (35)$$

Similarly, it can be shown that:

$$\begin{aligned} \Pi(a,b) \cdot \hat{r} \quad > \quad & -\frac{1}{min_{i=1}^{m} b_{x_i}} |r_1| \left(\sum_{i=1}^{m} |b_{y_i}| + \sum_{i=1}^{n} |a_{y_i}| \right) - \\ & \frac{1}{min_{i=1}^{m} b_{x_i}} |r_2| \left(\sum_{i=1}^{m} |b_{z_i}| + \sum_{i=1}^{n} |a_{z_i}| \right) - \\ & \frac{1}{min_{i=1}^{m} b_{x_i}} |r_3| \left(\sum_{i=1}^{m} |b_{z_i}| \sum_{i=1}^{n} |a_{y_i}| + \sum_{i=1}^{m} |b_{y_i}| \sum_{i=1}^{n} |a_{z_i}| \right) \end{aligned} \quad (36)$$

Inequalities 35 and 36 are independent of scalars α and β. This implies that they are not functions of the line $\overline{ab}$, but rather are functions only of the vertices of A and B. Since a and b are arbitrarily chosen points of A and B (respectively), this implies that they bound the Plücker duals of all stabbing lines through A and B. Thus giving acceptable values for h^+ (Equation 35) and h^- (Equation 36).

These bounds are not necessarily optimal. We consider the determination of tighter bounds (or the proof that those bounds presented here are optimal) to be an area of future research. The implication of loose bounds on these caps is that the bounding spheres used for accelerating CSG (see Section 5.3.1) become more conservative, resulting in less trivial rejection.

Since the polytope exists in $\mathbb{R}^5$ and not $\mathbb{P}^5$, the coefficients of the capping planes are not subject to geometric distortion. Distortion due to projectivity occurs at the stage where $\mathbb{P}^5$ is projected down to $\mathbb{R}^5$. The projective plane that is chosen determines the size of the capped polytope in $\mathbb{R}^5$.

With our current selection of a projective plane[5] a singularity does occur, but this can be treated (see Section 5.2.6). Using an alternative plane removes this problem, however, the formal proof becomes more complex[6]. In particular, since the two polygons are disjoint convex sets, there must exist a separating plane. It is this plane that should be transformed to be parallel to the YZ plane. The advantage is that no stabbing line can become parallel to this plane.

5.2.6 The Polytope-Plane Intersection Case

In Section 5.1.1, we show that the set of stabbing lines for a set S can be represented by a set of constraints using a dual mapping of edges to half-spaces in Plücker coordinates. These constraints

[5] We orient the plane of one polygon to be parallel to the YZ plane.

[6] We intend this as future work.

(Equation 13) are depicted in Figure 32.

There is one case where these constraints may not fully constrain the set of lines to *only* those stabbing S. This is the case where the *plane* of at least one polygon intersects another polygon in the stabbing set. Experiments have shown that this case occurs frequently in practice.

For illustration purposes, we consider the case with two polygons. Let A and B be two convex polygons, where the plane embedding B intersects A. Denote the set of points of this intersection as the set I (see Figure 33). The only lines incident on I that should be part of the set of stabbing lines, are those that stab B. Decomposing the set of constraints we obtain the following:
Constraints of A:

$$D_\alpha(\Pi(s)) \geq 0, \forall \alpha \in \{\Pi(a_1), \Pi(a_2), \ldots, \Pi(a_n)\} \tag{37}$$

Constraints of B:

$$D_\beta(\Pi(s)) \geq 0, \forall \beta \in \{\Pi(b_1), \Pi(b_2), \ldots, \Pi(b_m)\} \tag{38}$$

Where $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ are the edges of A and B respectively. We note that since constraints of B are defined using a greater or equal to relation, then *all* lines incident on I and embedded in the plane B satisfy the constraints of B. Furthermore, any one of these lines are incident on every edge to line extension of the edges of B. Indeed, each one of these lines ℓ satisfies: $D_\beta(\Pi(\ell)) = 0, \forall \beta \in \{\Pi(b_1), \Pi(b_2), \ldots, \Pi(b_m)\}$.

One way to prevent this problem is to clip the polygon A against the plane of B at a distance of ϵ, where ϵ is very small. This should work in practice, since the only error introduced is the possible omission of those polygons stabbed only from the ϵ strip of A. Once again, this ϵ strip is a very small area.

A better alternative, however, is to further constrain the set of lines. We introduce two additional constraints (defined by two lines c_1 and c_2) that limit the lines on the plane of B to those lines incident on B.

Lines c_1 and c_2 are defined as follows: Let p_1 and p_2 be the endpoints of the line segment I (intersection of the plane of B with A). Let a_{p_1} and a_{p_2} be the edges of A incident on p_1 and p_2, respectively. Define the plane P to have the normal vector $p_2 - p_1$, and to go through p_1. Next, the vertices of B are sorted by their signed Hausdorff distance from P. The vertex with the smallest distance v_1 is associated with p_1, and the vertex with the largest distance v_2 is associated with p_2. These two pairs of vertices define two lines, namely the *supporting* lines of I and B. Finally, c_1 is defined to be the line with the same direction as a_{p_1}, but translated so that it is incident on v_1. Similarly, c_2 is defined to be the line with the same direction as a_{p_2}, but translated so that it is incident on v_2.

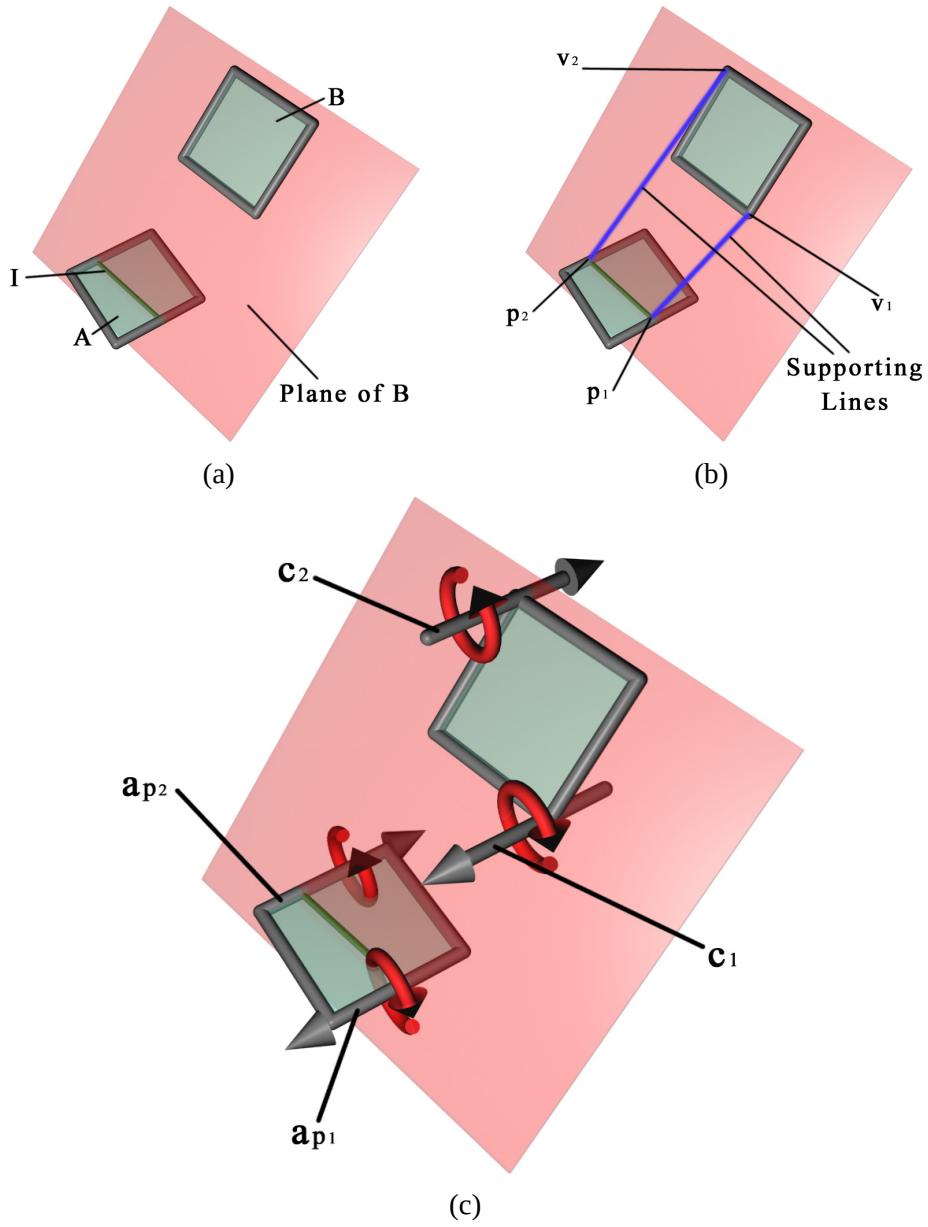


Figure 33: *Plane-Polygon Intersection Case*. (a) A two polygon scene, where the plane of polygon B intersects polygon A at line segment I. All directed lines in the plane of B that intersect I in the correct direction through A will satisfy the edge constraints of these two polygons. This *includes* non-stabbers (b) The supporting lines of I and polygon B. (c) The addition of two more constraints, defined by c_1 and c_2 will constrain the set of lines in the plane of B to only those that intersect B.

Adding the following constraints allows for an exact solution to this case:

$$D_\pi(\Pi(s)) \geq 0, \forall \pi \in \{\Pi(c_1), \Pi(c_2)\} \tag{39}$$

The general half-space intersection routine must be used in this case.

5.3 Phase 2: Incorporating Misses

In Phase 1, the set of lines stabbing the set of polygons S are computed and represented as a polytope in $\mathbb{R}^5$ using a Plücker parameterisation. In Phase 2, polyhedral set subtraction operations are used to trim this polytope in order to remove those points corresponding to lines *blocked* by a second set of polygons M.

5.3.1 CSG in Plücker Space

To remove those lines blocked by an element of M from the polyhedron, we have to find a more explicit representation for the set of lines blocked. As for the stabbing 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 necessarily unbounded. We depict the mapping of an occluder to $\mathbb{R}^5$ in Figure 32b, and the subtraction of this volume from the stabbing polytope in Figure 32d.

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 volume into a set, or *complex* (see Section 3.3), of polytopes that have no faces crossing any hyperplane of $\mathcal{O}$. The polytopes 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 would therefore provide the remaining set of lines (those unblocked by the occluder).

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 polytopes, within the complex, intersects the Plücker hypersurface.

Splitting a Complex by a Hyper-plane

We develop an algorithm similar to the multi-dimensional polytope splitting algorithm of Bajaj and Pascucci [BP96]. The original algorithm (discussed in detail in Section 3.3) maintains the whole polytope-complex as a single face graph structure. Their algorithm has to traverse each face of each dimension. For each of these faces, its children ($k-1$ dimensional boundary elements) need to be visited. This results in an $O(pm)$ time algorithm for p faces and an average of m children per face. Bajaj and Pascucci do suggest the use of a half-space range reporting algorithm to classify the vertices more efficiently. We present an alternative that allows *all* faces to be classified efficiently.

The basis of our improvement is that for each top level polytope of the complex, we construct two *notationally disjoint*[7] top level polytopes when splitting, one on either side of the hyperplane. This introduces redundancy into the representation, since shared faces are duplicated, but we gain the performance advantage of rapidly isolating those polytopes that are split, thereby reducing the set of polytopes traversed to those incident on the hyperplane. That is, the traversal is sensitive to the number of polytopes in the *zone* of the hyperplane.

To query whether a polytope intersects a hyperplane, we use a 5D bounding sphere for a conservative test. If this conservative test indicates a potential intersection, an accurate vertex-hyperplane sidedness test is 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.

Figure 34 gives pseudo-code for our algorithm (cf original in Figure 9). There are several important differences between our algorithm and the original: In the first step the algorithm begins with a conservative computation of those polytopes incident on the splitting hyperplane. In the second step, the algorithm computes exactly which polytopes are split by the given hyperplane.

By the third step, all the primary numerical calculations have been performed. In the third step all the split faces are traversed. The splitting occurs in a similar manner to the original algorithm. Unlike the original algorithm, we have moved the secondary numerical calculation into the splitting algorithm, so that the vertices arising from hyperplane-edge splits are evaluated when they are determined. The advantage of this is that all the required information (i.e., the children vertices of the split edge) are readily available. The original algorithm requires auxiliary data be to refer to the two vertices of each split edge.

Another modification, required for a notationally disjoint complex, is the maintenance of a list of those faces synthesised from the splitting operation. These are exactly those faces marked as

[7] Disconnected face lattices.

Step 1 *(Conservative intersection test)*
  $S \leftarrow \emptyset$
  for each $c \in C$ **do**
    if boundingSphere(c) intersects H **then**
      $S \leftarrow S \cup c$
    endif
  next c
Step 2 *(Accurate intersection test)*
  for each $s \in S$ **do**
    Classify all the vertices of s either $\boxed{+}$ or $\boxed{-}$.
    if allVertices(s) = $\boxed{-}$ **or** allVertices(s) = $\boxed{+}$ **then**
      $S \leftarrow S \setminus s$
    endif
  next s
Step 3 *(Split zone of H)*
  for each polytope $s \in S$ **do**
    $E \leftarrow \emptyset$
    for $k = 1$ to d **do**
      for each $c \in s_k$ do:
        if not allFacets(c) = $\boxed{-}$ **and not** allFacets(c) = $\boxed{+}$ **then**
          - Create a new $(k-1)$-polytope f (classified as $\boxed{=}$) and connect it to each $(k-2)$-polytope in c classified as $\boxed{=}$.
          - If $k = 1$, then compute the vertex associated with f by intersecting H with c.
          - Create two polytopes c^+ and c^- connected both (down) to f and (up) to all the $k+1$ polytopes connected with c.
          - Connect each $(k-1)$-polytope in c classified as $\boxed{+}$ to c^+, and each one classified as $\boxed{-}$ to c^-.
          - Remove c from s_k.
          - $E \leftarrow E \cup f$.
        endif
      next c
    next k
    - $E' \leftarrow E$
    - removePositiveConnectivity(E)
    - removeNegativeConnectivity(E')
    - $s^+ \leftarrow$ postiveFaces(s)
    - $s^- \leftarrow$ detachPositiveFaces(s)
    - $s^+ \leftarrow s^+ \cup E'$
    - $C \leftarrow C \setminus s$
    - $C \leftarrow C \cup s^+$
    - $C \leftarrow C \cup s^-$
  next s

Figure 34: *Improved Polytope Splitting Algorithm.* The algorithm uses the following terminology: d is the dimension, H is the hyperplane, s_k is the set of all faces of polytope s of dimension k. C is the polytope complex. Final optimisations are given in Figure 40.

$\boxed{=}$. One can visualise this set E, as the face lattice of the polytope resulting from a cross section through the complex by the given hyperplane.

Using E (computed per polytope), it is possible to sunder each polytope into two disjoint face lattices. Firstly, the faces marked positive are removed and inserted in a new face lattice. Secondly, a cloned version of the E set is attached to the new face lattice. In the representation of Bajaj and Pascucci, the set E is shared. The redundancy introduced by our approach is exactly the cost of storing those faces in the E set twice. It should be noted though, that the connectivity information is *not* duplicated, as the set E is attached only to the negative complex, while the set E' is attached only to the positive complex. We illustrate this principle in Figure 35. Note that when we refer to face lattice "unions" we mean the union of both the set of faces and the reconstitution of the connectivity.

Removing a Polytope from a Complex

Bajaj and Pascucci do not describe an algorithm that will allow a polytope to be removed from the complex. Using the original representation of the complex, we have derived a simple recursive polytope subtraction algorithm. See Figure 36. A similar algorithm for our symbolically disjoint representation is presented in Figure 37. The difference, is that when using the disjoint approach, connectivity with other polytopes does not have be checked before deletion.

5.3.2 Optimising the CSG process

In order to achieve the most efficient CSG, we attempt to minimise the number of split operations performed on the complex. We begin by considering a naïve algorithm for solving the selective stabbing problem. We use the notion of a hyperplane arrangement in line space as a way of deriving our algorithm from first principles. This has two purposes: Firstly, we show how optimisations can be applied successively, until our algorithm is found. Using some of the concepts resulting from the arrangement derivation below, an algorithm that is more efficient than the one presented in Section 5.3.1 is found. Secondly, the consideration of our algorithm in this context makes the qualities that differentiate it from global visibility algorithms [DDP96, Pel93] explicit.

By inserting the hyperplanes of the stabbing polytope, along with the hyperplanes defined by the Plücker duals of *all* occluder edges into a 5D arrangement, all isotopy classes are enumerated. What remains, is to inspect each isotopy class (5D faces of the arrangement).

If one such class corresponds to a set of stab-miss lines, then the result of Question 1 (see Page

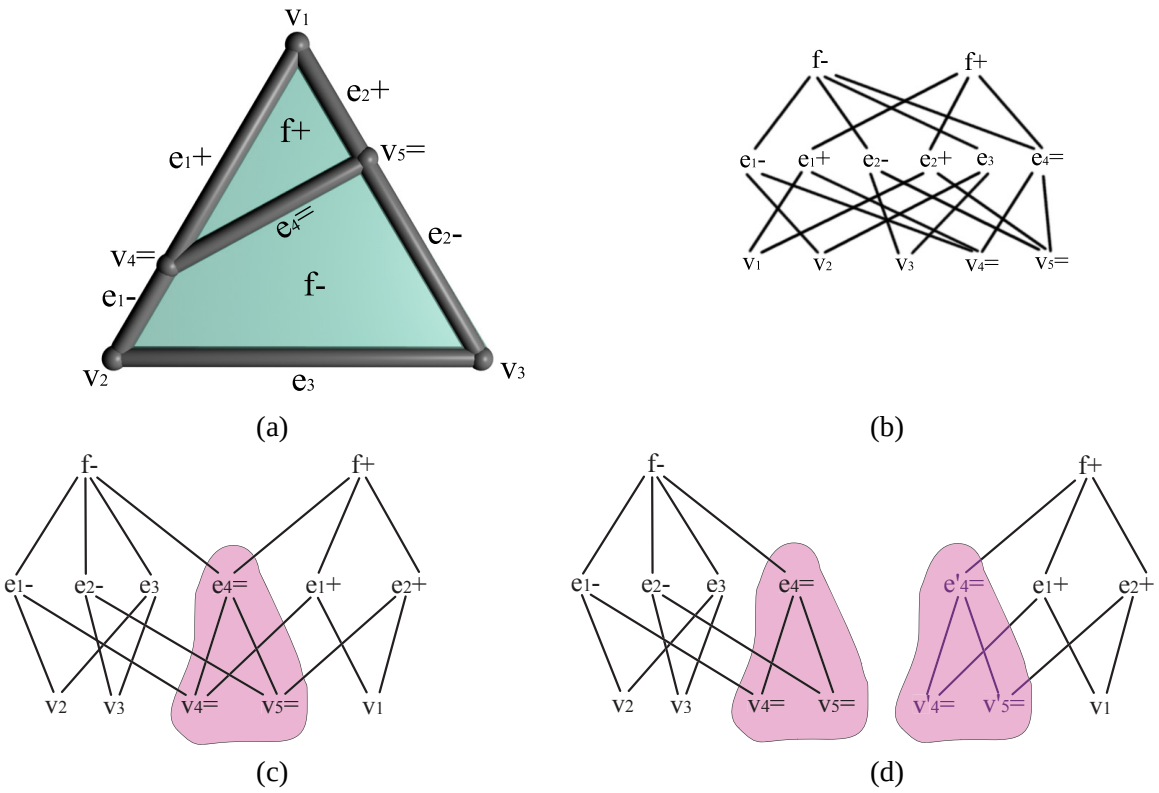


Figure 35: *Modified triangle splitting.* (a) A triangle split by a hyperplane (see also Figure 8). (b) The face lattice of the split triangle depicted in (a). (c) A reordering of (b) to show the shared faces E (highlighted set). (d) The separating of E to generate a disjoint face lattice. Note that the face representations ($e_4^=$, $v_4^=$, $v_5^=$) are duplicated, but that the connectivity (the six edges crossing the boundary of E in (c)) remains the same (the six (3+3) edges leaving the two regions in (d)).

```
procedure RemovePolytope( P )
    for each child k of P do
        if NumParents(k) = 1 then
            RemovePolytope( k )
    next k
    delete( P )
```

Figure 36: *Original Approach – Polytope Removal.* An algorithm for removing a polytope from the original polytope complex representation of Bajaj and Pascucci.

```
procedure RemovePolytope( P )
    for each child k of P do
        RemovePolytope( k )
    next k
    delete( P )
```

Figure 37: *Modified Approach– Polytope Removal.* An algorithm for removing a polytope using our representation.

91) is true, otherwise it is false. Question 2 (see chapter start) may be answered by reporting a list of all isotopy classes corresponding to stab-miss lines[8]. Care must be taken not to include those 5D faces of the arrangement that do not correspond to isotopy classes of *real* lines (i.e., those faces of the arrangement that do not intersect the Plücker hypersurface).

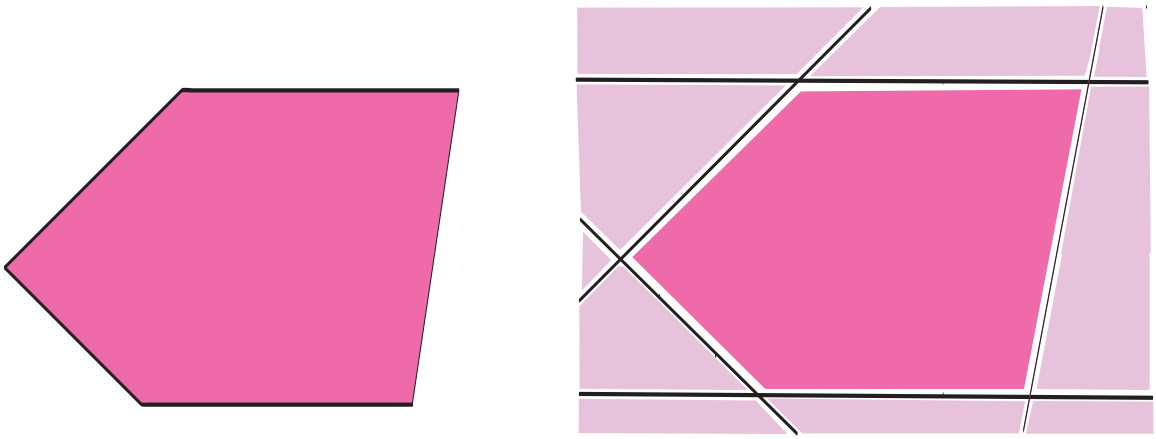
As it stands, this algorithm runs in $O(n^5)$ time, since this is the complexity for constructing a 5D arrangement [EOS86, Ede87]. By constructing only those parts of the arrangement in the zone of the Plücker hypersurface, this time may be reduced to $O(n^4 \log n)$ [APS93].

The first optimisation is to compute only those isotopy classes that correspond to stabbing lines of S. In a direct construction of an arrangement, determining this subset would be prohibitively expensive. However, since the set of such isotopy classes can be computed explicitly (i.e., they have the same intersection with the Plücker hypersurface as the stabbing polytope of Section 5.1), we begin a priori with the single isotopy class of lines that stab all of S (to be refined later by the inclusion of M).

If we consider the polygon in Figure 38a to be a visualisation of the 5D stabbing polytope, we see how it can be embedded into an arrangement of its defining hyperplanes (Figure 38b).

In general, we do not have to compute the whole subset of the arrangement that falls within the stabbing polytope. It is more efficient to construct and inspect progressively, in a hierarchical fashion: let the total number of edges in S and M be n. Let H be an ordered set of the dual hyperplanes of the n edges in both S and M respectively. Let $\mathcal{A}^k$ refer to the arrangement constructed from the insertion of the first k elements of H. By this definition, $\mathcal{A}^{|S|}$ will then partition line space into those elements that stab S and those that do not (see Figure 38b). The sequence $\mathcal{A}^0, \mathcal{A}^1, \ldots, \mathcal{A}^n$ defines a spatial hierarchy: for any $0 < i < n$, the arrangement $\mathcal{A}^{i+1}$ is always a refinement of the arrangement $\mathcal{A}^i$. A cell $c \in \mathcal{A}^i$ is a parent of a cell $d \in \mathcal{A}^{i+1}$ *iff* $d \subseteq c$.

[8]The reader should note that if a given isotopy class contains one stab-miss line, then *all* lines within the class will also be stab-miss lines.



(a) (b)

Figure 38: *Polytope in an Arrangement*. (a) A polytope. (b) An arrangement embedding the polytope of (a). Recall that lines are hyperplanes in 2D.

Should a cell $c \in A^k$ correspond to an isotopy class (defined only by the first k edges) that is blocked/occluded, then all descendants of c in the hierarchy will also be blocked. Further information could be gained by continued refinement, although this information (e.g. knowledge of additional occluders) is unnecessary for our application. In terms of implementation, the cells that are determined to be blocked are removed or rather *subtracted* from the complex, and are thus no longer represented explicitly.

To illustrate this last concept, we depict a stabbing polytope that has had an occluder subtracted from it (in Figure 39a). Figure 39b shows the complex of this object, and Figure 39c shows the arrangement in which it is embedded. Let us call the arrangement of Figure 38b, $\mathcal{A}^5$. Then the arrangement of Figure 39c is $\mathcal{A}^7$. It is unnecessary to compute the blue polytopes in Figure 39e, since they correspond to an already blocked (subtracted) polytope in Figure 39c.

By removing all "blocked cells" progressively, large branches of the hierarchy are pruned (i.e., refined isotopy classes). In most cases this provides greatly improved performance. The problem of choosing a good ordering, in order so as to maximise performance is discussed in Chapter 6.

Further optimisation is still possible. The number of isotopy classes are sensitive to the number of lines (one line is extended from each edge). Accounting for the boundedness of the edges, allows us to collapse many isotopy classes into a considerably coarser, yet sufficiently refined superset.

Simply put, it is unnecessary to refine an isotopy class according to whether subsets of it lie on

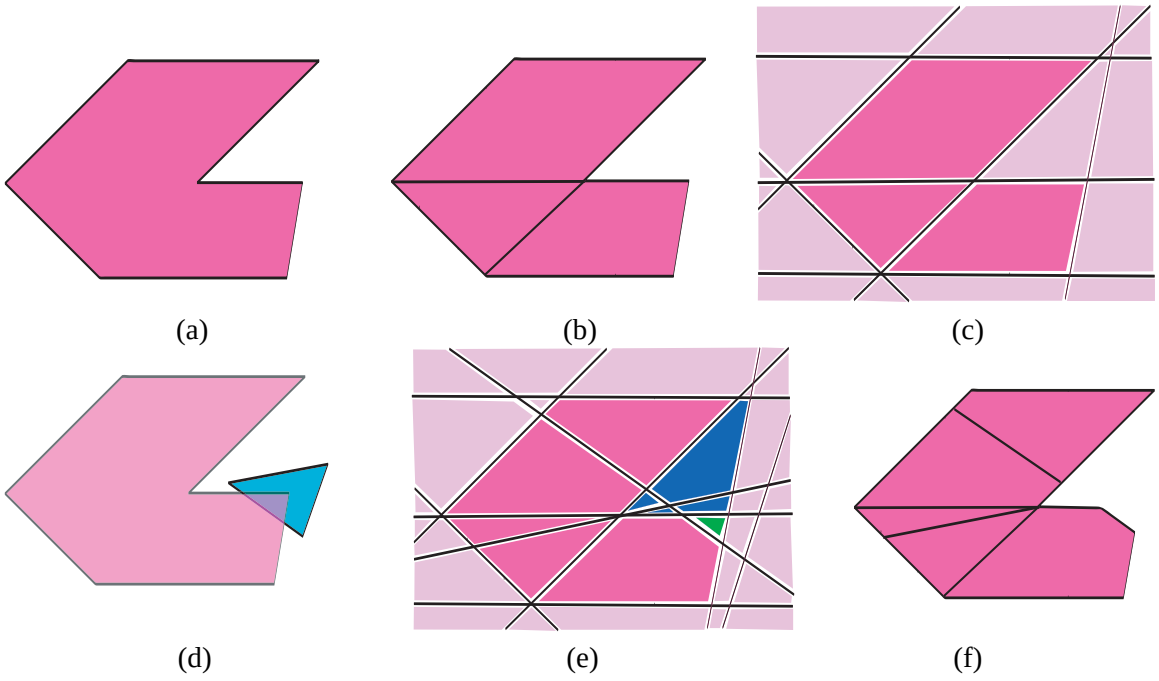


Figure 39: *Polyhedral Complex in an Arrangement*. (a) A polyhedron. (b) A polyhedral complex representing the polyhedron in (a) as the union of convex polytopes. (c) An arrangement embedding the polyhedron and complex of (a) and (b), respectively. (d) An object is to be subtracted. (e) Complex after the hyperplanes of the object in (d) have been used for further refinement. The blue polytopes correspond to isotopy classes that are occluded/blocked, but fall within a region *previously* determined to be blocked. The green polytope refines the stabbing polytope to define a newly occluded isotopy class. (f) The resulting polytope complex (i.e., the subset of the arrangement that is yet unblocked).

one side or another of some edge of x, if it can be shown that the set of lines it represents cannot be blocked by x. This is illustrated in Figure 41.

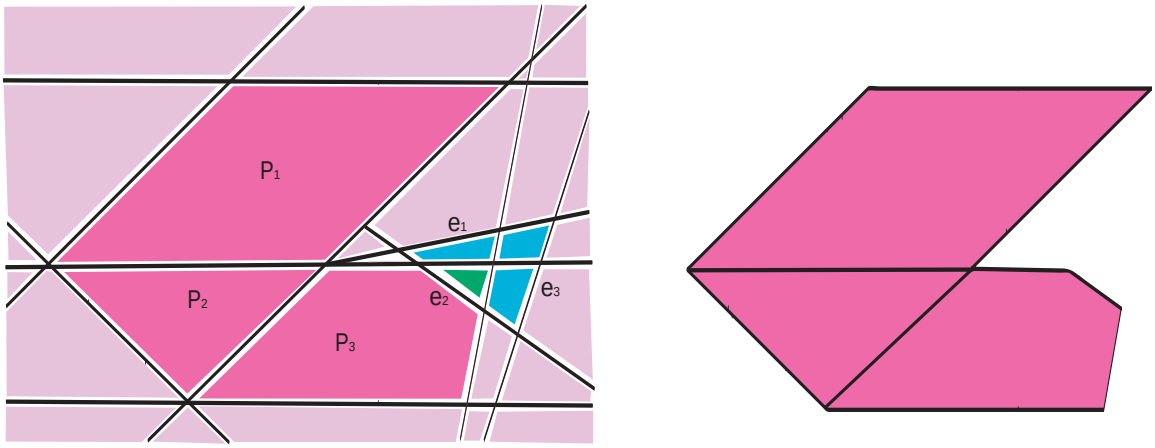
To achieve this, our occluder subtraction routine will split *only* those parts of the polytope complex that lie on the boundary of the polyhedron to be subtracted O. When splitting the complex, this is a simple matter of removing those polytopes that lie completely in the "outside" half-space of any of the hyperplanes defining O. We give a modified Step 1 and 2 in Figure 40. The key concept is that this splitting algorithm is "aware" of the subtraction being performed, thus preventing unnecessary refinement within the complex.

In summary, the algorithm begins with the construction of the stabbing polytope. Next, blocked polyhedra are subtracted from the complex using our augmented polytope complex splitting algorithm. These splits are invoked on a given polytope of the complex *iff* the polytope lies on the

```
Step 1' (Conservative intersection test)
    L ← all hyperplanes belonging to occluder polyhedron
    L ← L \ H
    S ← ∅
    for each c ∈ C do
        if boundingSphere( c ) intersects H then
            for each l ∈ L do
                splitC ← true
                if boundingSphere( c ) lies outside l then
                    splitC ← false
                    breakFromLoop( l )
                endif
            next l
            if splitC then
                S ← S ∪ c
            endif
        endif
    next c
Step 2' (Accurate intersection test)
    for each s ∈ S do
        Classify all the vertices of s either [+] or [−] with respect to H.
        if allVertices( s ) = [−] or allVertices( s ) = [+] then
            S ← S \ s
        else
            for each l ∈ L do
                Classify all the vertices of s either [+] or [−] with respect to l.
                    if allVertices( s ) = [−] then
                        S ← S \ s
                        breakFromLoop( l )
                    endif
            next l
        endif
    next s
```

Figure 40: *Subtraction aware step 1 and 2.* This is a modification of the algorithm presented in Figure 34. This algorithm is aware that the split operation is part of a subtraction operation, and therefore avoids extraneous splitting. This includes all optimisations.

boundary of the polyhedron to be subtracted. At any time, the only data that is persistent in memory is the face lattices and vertices of the set of polytopes corresponding to unblocked lines space.



(a) (b)

Figure 41: *Optimised Splitting.* (a) The same subtraction is being performed, as per Figure 39d. However, P_1 is not split by e_2, since P_1 lies entirely on the "outside" half-space of e_1 (the inside half-spaces of e_1, e_2 and e_3 intersect in the blue triangle). Similarly, P_2 lies wholly in the outside half-space of e_2 and therefore P_2 is not split by e_1. (b) A polyhedral complex representing the polyhedron in (a). Note that the structure of the complex is much simpler than that in Figure 39f.

5.4 Conclusion

In this chapter, we have presented a solution to the selective stabbing algorithm. The solution algorithm executes in two phases. First a stabbing polytope is built that represents all lines in the stabbing set. Such solutions do exist, however, the case where the stabbing set consists of exactly two polygons requires special treatment. We discover that the form of the polyhedron is that of an extruded 4D polytope. We prove a theorem showing how the axis of extrusion can be derived explicitly. This is then used to build the stabbing polytope efficiently, without the usage of a less efficient general vertex enumeration algorithm. The general form of the required bounds for a pair of planes capping the polyhedron are also computed.

The second phase of the algorithm describes our 5D polyhedral constructive solid geometry algorithm. This algorithm incrementally trims away the unoccluded line space from the stabbing polytope. Should the whole stabbing polytope be trimmed away, no stab-miss lines can exist. Otherwise, the remaining parts of the polytope can map directly to all the existing stab-miss lines. We augment an existing polytope complex splitting algorithm to achieve this. The new version is sensitive to those polytopes in the zone of they hyperplane. We also show how a subtraction operation can be implemented efficiently using such a splitting algorithm.

Chapter 6

Exact Visibility Pre-Processing

> "How many trees make up the forest? How many houses a city?...as
> the German proverb goes, one cannot see the forest for the trees.
> Forest and city are two things essentially deep, and depth is
> fatally condemned to become a surface if it wants to be visible."
> – Jose Ortega y Gasset, "The Forest", Meditations on Quixote, pg. 59.

This chapter details our exact from-region visibility algorithm. This consists of two parts. The first part is a *visibility query* engine, which queries *exactly* whether or not one polygon can see another within a 3D scene. The query algorithm can be cast as a specialised selective stabbing problem. We detail this in Section 6.1.

Our visibility query algorithm is, in essence, a direct construction of a localised subset of the visibility complex [DDP96]. 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.

We use the visibility query as a means of experimentally verifying the performance of the selective stabbing algorithm of Chapter 5. In particular, we present and evaluate heuristics for choosing an efficient order of occluder/blocker subtraction.

The second part is the global framework. The goal of our framework is to utilise the visibility query effectively, in order to compute from-region visibility efficiently. This is detailed in Section 6.2. Our general approach is to prevent the unnecessary recomputation of visibility results. We generate and use *virtual occluders* to disregard line sets already determined to be occluded. This results in an *output sensitive* algorithm. We also use a two tiered spatial hierarchy for clustering

occlusion queries. The computations used in the parent nodes (bounding boxes) are reused when evaluating the children (primitives).

Another objective of the framework is to perform *only* those computations necessary to determine exact from-region visibility. Integrating the exact query algorithm into a framework where the visibility of a majority of polygons can be determined trivially, allows processing time to be spent only on those queries that require an exact algorithm to be evaluated accurately.

Results are given which demonstrate the effectiveness of our approach. This includes an empirical analysis of the scalability of the whole system and the visibility query in context.

6.1 The Visibility Query

In this section we describe our exact visibility engine, based on an implementation of the selective stabbing solution presented in Chapter 5. We begin by casting the visibility problem as a selective stabbing problem. We then investigate the most important performance factor: choosing a good order for occluder subtraction. Finally, we discuss the visibility algorithm in the context of other analytic visibility techniques.

6.1.1 Casting Visibility as Selective Stabbing

Visibility can be defined formally in terms of line segments. Namely:

Definition 6.1 *A set of points Y is said to be* visible *from a set of points X if there exists a closed line segment s whose two end points lie in X and Y respectively. Additionally, s cannot intersect another set M, where M is the set of points representing all occluding geometry.*

Definition 6.2 *We refer to a segment s with the properties described in Definition 6.1 as a* sight segment.

We wish to cast the visibility problem as a selective stabbing problem, where the stab set S is comprised of two query polygons, and the miss set M, consists of all occluding geometry.

A selective stabbing scenario cast this way, is sufficient to prove or disprove visibility, if and only if, the supporting line of any potential sight segment only intersects M if the sight segment does. Indeed, if this condition is met, it is clear that there is a bijection between the set of sight segments and the set of stab-miss lines. Examples illustrating this principle may be found in Figure 42.

It is always possible to derive a set M' from M which guarantees that the above condition is met. The approach which involves the smallest change from M, is to *clip* M to the union of two

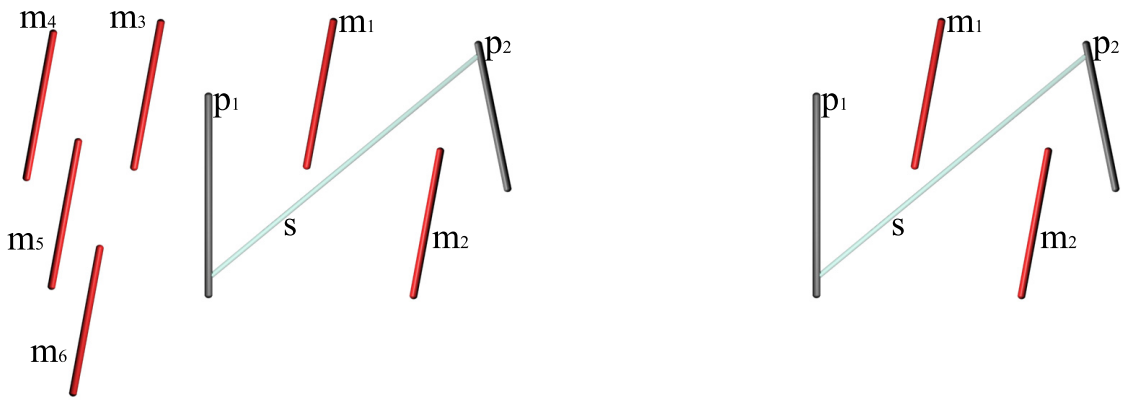


Figure 42: *Casting to Visibility – Problem*. The above figures show a plan view of a typical query scenario. (a) A scene where at least one sight line (s) does exist between the query polygons (p_1 and p_2). m_1 to m_6 are occluders. Note that as a selective stabbing problem, *no* stab-miss line exists. (b) A similar scene to that of (a). However, every sight line between p_1 and p_2 is supported by a stab-miss line.

anti-penumbra: one cast by the first stabbing polygon through the second, and the other cast by the second stabbing polygon through the first. This is illustrated in Figure 43a.

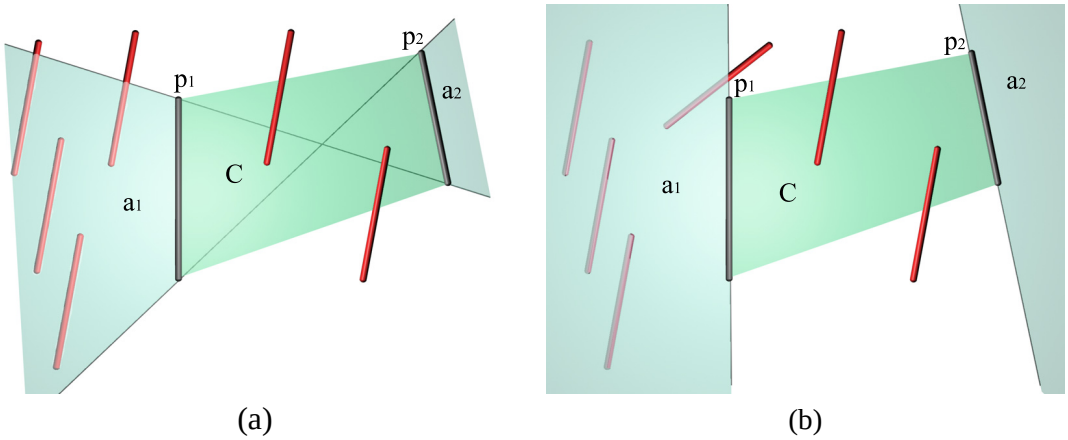


Figure 43: *Casting to Visibility – Solution*. The above figures show a plan view of a typical query scenario. (a) C depicts the convex hull of the query polygons. a_1 represents the anti-penumbra cast through p_1 by p_2. Similarly, a_2 represents the anti-penumbra cast through p_2 by p_1. All occluder geometry within the regions a_1 and a_2 must be removed in order to ensure sight segment to stab-miss line bijectivity. (b) A faster conservative approach. All geometry on the one side of the planes embedding p_1 and p_2 are clipped (all geometry in regions a_1 and a_2). The geometry clipped is on the side which does *not* contain the convex hull C.

There are, however, more conservative techniques which also allow for this condition to be met. Let C be the convex hull of S. It is sufficient to clip all polygons against two planes: the first plane being the plane embedding the first polygon of S, the second being the plane embedding the second polygon of S. The parts of M discarded, are those on the opposite sides of the plane from C. This is illustrated in Figure 43b. In general though, it is advantageous to remove those parts of M which do not intersect C, since they cannot affect the existence of sight segments. This is simply an efficiency consideration.

In Chapter 5, two questions were proposed and solved, one being the *existence* of a stabbing line, and the other being the computation of the *set* of stabbing lines. In the context of visibility, the question of existence is equivalent to the question of mutual visibility. Similarly, the computation of the set of stab-miss lines is equivalent to determining those parts of one polygon which fall into the anti-penumbra cast by another.

One clear extension of this technique is to that of cell-portal rendering. Where Teller [Tel92b] compute the anti-penumbra through a set of portals, our technique allows for the inclusion of occluders which block the anti-penumbra.

6.1.2 Selecting a Good Order of Subtraction

The focus of Chapter 5 is on *solving* the selective stabbing problem. Choosing a good order for subtracting occluders (elements of the miss set) is critical for good performance in practice.

The most important consideration, is that of *early termination*. When subtracting dual occluder polyhedra from the Plücker polytope (see Section 5.3.1), it is sufficient to terminate early when occlusion has been established. This is equivalent to terminating once the remaining polytope complex (which at all times represents the unoccluded line space) is empty[1].

During the subtraction process we have noted the following: initially, the number of polytopes in the complex typically grows as a function of the number of subtractions. This is due to the split operations performed during subtraction. Next, a turning point is reached, and the number of polytopes then begin to decrease rapidly. This is a direct result of the increase of the granularity of the complex: the finer it is subdivided, the more likely it is that more polytope elements will be removed during each subtraction.

[1]Those complex elements that do not intersect he Plücker hypersurface are removed implicity during the subtraction process

Naïve Ordering

Our first choice of order was a naïve ordering. We simply subtracted the occluders in the order in which they occurred in the mesh. We have found this to be a pathological case. Adjacent mesh elements are typically spatially coherent. Since spatial coherence translates to line space coherence[2], successive subtractions are performed in the same spatial locality (in Plücker space) as their recent predecessors. This generates subdivision upon subdivision, resulting in a near linear initial growth in the number of elements in the complex as a function of the number of subtractions. This linear growth in the number of complex elements results in a near quadratic run-time, since each successive subtraction is tested or applied to a growing number of polytopes that cannot be trivially rejected due to the Plücker space coherence.

We have developed several heuristics that improve greatly on this naïve subtraction order (i.e., they result in earlier termination). These include prioritising by occluder size (weighted by angle), randomising the subtraction order, as well as a novel line space sampling methodology. All of these heuristics avoid the pathological case described above, since there is little or no coupling of the ordering to spatial coherence. In Section 6.3 we see that slightly super-linear growth is achievable.

Area-angle Measure

The solid angle refers to the projected area (in steradians) of an object onto a unit sphere surrounding some reference point. This area can be approximated (see Coorg and Teller [CT97]) as a function of distance, area and angle. The situation is somewhat different in the context of a visibility query, since we are dealing with visibility from a region, not a single point. Also, the situation is different from typical from-region visibility [KCO00], since we are dealing with visibility from a bounded region, to a bounded region. Koltun and Cohen-Or [KCO00] use an umbra-volume approximation as a heuristic. Ideally, we should be able to determine the area of the umbra on *one* polygon cast by a source polygon. We have used the following metric (see Figure 44 for an explanation of the terms):

$$AreaAngle = a\frac{(r_1 - r_2).\vec{n}}{||r_1 - r_2||} \tag{40}$$

[2]The set of lines intersecting one triangle, will typically lie next to the set of lines of an adjacent triangle sharing an edge. Since the Plücker mapping is isomorphic (and in particular, continuous), this is reflected as spatial coherence in Plücker space.

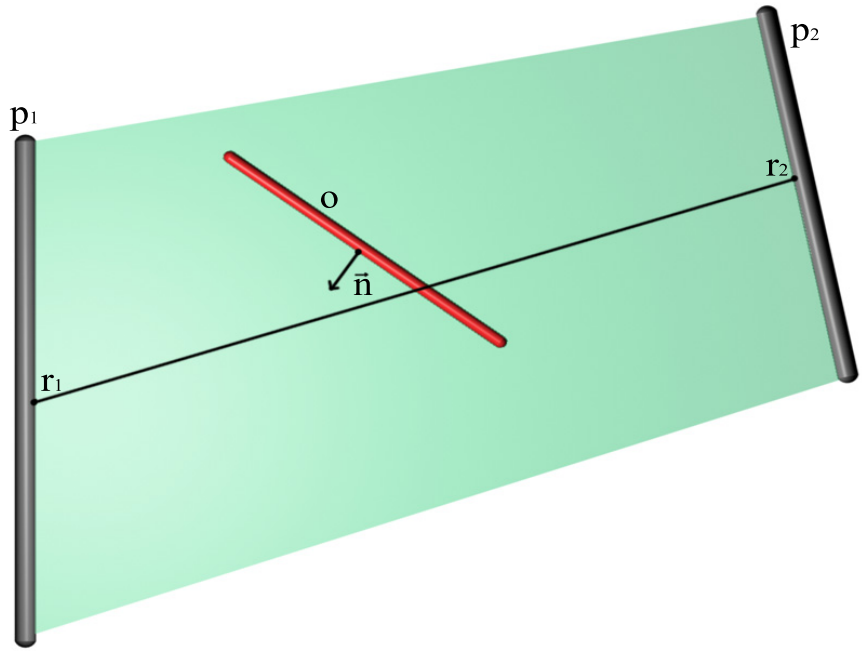


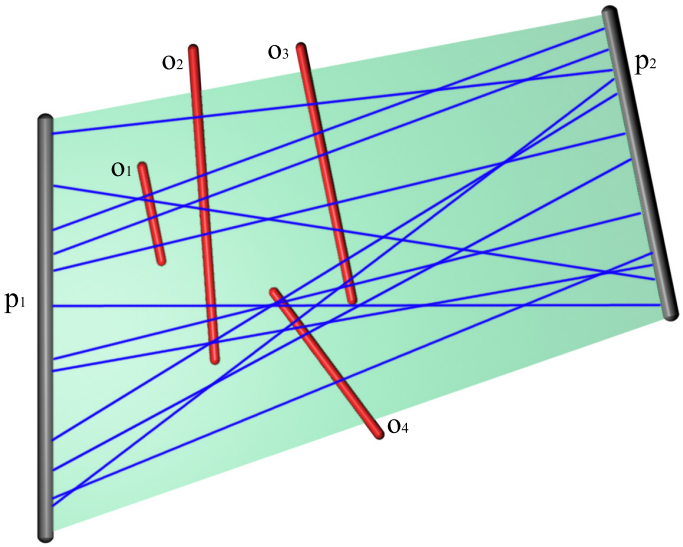
Figure 44: *Area-angle Metric*. A plan view. The area of the polygon is referred to as a, the reference points r_1 and r_2 are chosen to be the respective centroids of p_1 and p_2. $\vec{n}$ is the normal of the polygon o.

Effective Occluders

Undoubtedly, however, the most effective technique we have implemented is a direct sampling of line-space. This does not consider each occluder independently, but actually accounts for potential occluder fusion. The algorithm proceeds as follows: First, rays are cast between the two query polygons[3]. Next, the polygon intersecting the most rays is subtracted. Next, the polygon intersecting the most rays *excluding* those rays already accounted for is subtracted. This last step continues until either occlusion is established, or all the rays are accounted for. In the latter case, we relegate the choice of priority among the remaining polygons to another heuristic. We illustrate this ordering by the example in Figure 45.

We define an *effective occluder* to be a member of the set containing the *minimum* number of occluders required to block the total occluded line space. The ordering defined by the direct line sampling defines both an order of subtraction, as well as a set of occluders that approximates this smallest possible set of occluders.

[3]We cast two rays for every triangle within the convex hull of the query polygons. We consider the determination of a better casting strategy to be an area of future research.



Step	o_1	o_2	o_3	o_4
1	4	**9**	8	7
2	0	-	1	**3**
3	0	-	0	-

Figure 45: *Order of Subtraction*. A plan view of a typical scenario is shown. 12 segments are generated between the query polygon pair p_1 and p_2. The table illustrates the number of segment samples incident on the occluders o_1 to o_4. The polygon with the highest priority is o_2. Since the sampling (9 hits) indicates probabilistically that it blocks the largest volume of line space. Next in the ordering is o_4, since the sampling indicates that it blocks the largest volume of *remaining/unoccluded* line space (3 hits, since 4 of the original 7 hits also intersect o_2). The remaining occluders o_1 and o_3 have no remaining hits, since all the incident samples are accounted for by o_2 and o_4. The ordering of these two defaults to an alternative mechanism (e.g. an area-angle metric).

The number of effective occluders is at most as large as the set of visible polygons within the query shaft. Using this approach we observe that since each effective occluder blocks at least one ray that is unblocked by another effective occluder, that *every* subtraction will reduce the volume of the polytope complex to some degree.

It should be noted that we do not perform standard ray-casting, since our priority is not based on a "first hit" result. Our technique would be more aptly named *segment intersection testing*. Our ray casting solution is a greedy algorithm, and does not guarantee that the computed set of occluders is the smallest. However, most important wasteful cases are avoided. These are catalogued in Figure 46.

A very important side effect of this approach is that it seamlessly integrates an efficient means to determine trivial acceptance: if any ray should not intersect any occluder, then the query polygons can be considered to be mutually visible. A vast majority of visible polygons are found in this way. This makes it far more likely that those queries on which the selective stabbing algorithm is actually

executed, are occluded. This further establishes the performance benefit of having a query algorithm that has a run time complexity that is sensitive to the number of effective occluders.

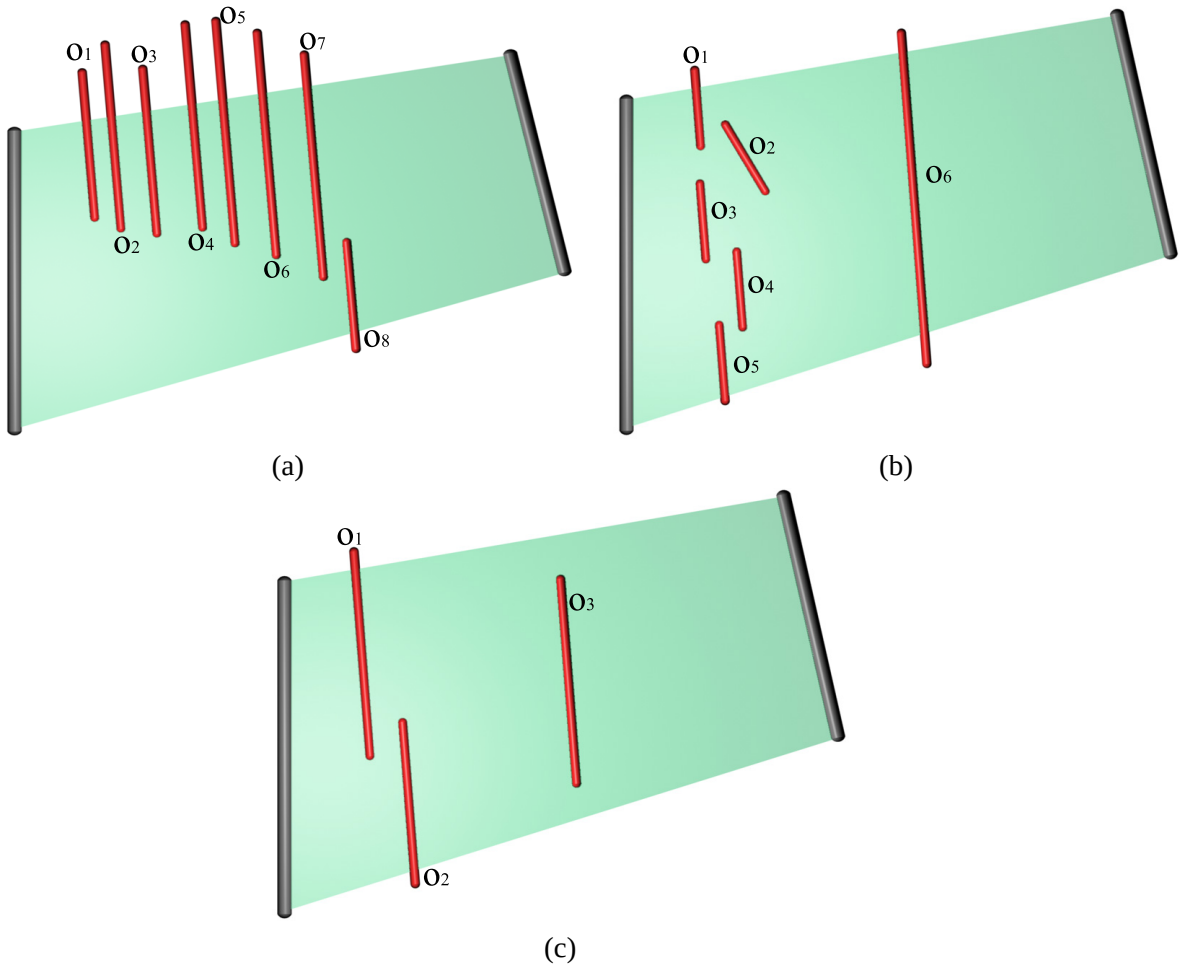


Figure 46: *Selection of Cases*. We assume that *many* segments have been cast. (a) A solid angle based priority would iterate through occluders o_7 to o_1, and then establish occlusion only when o_8 is subtracted. Our line space sampling approach will immediately isolate o_7 and o_8 and subtract them to establish occlusion. (b) If a first-hit ray cast was performed, the segments o_1 through o_5 would be the first subtracted. The line space sampling would immediately prioritise o_6. (c) An example of sub-optimality. o_3 is prioritised first, however *only* the combination of o_1 and o_2 is necessary to establish occlusion.

6.1.3 The Query Algorithm in Context

Both the visibility complex [DDP96] and skeleton [DDP97b] 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 primitive are visible. Their goal is to use visibility primarily for accurate illumination simulations.

In contrast, our goal is from-region qualitative per-polygon visibility. Our visibility query algorithm is 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. Furthermore, at any point of execution, only the parts of the complex representing the currently unblocked set of lines between the query polygons is resident in memory.

Durand *et al.* [DDP97b] present an on-demand construction of a subset of the visibility skeleton. The construction of this subset involves an explicit combinatorial iteration through the various elements of the scene geometry, in order to construct the lower dimensional elements of the complex. Just as for the visibility complex, this computes more than is required for our purposes, namely the visibility relationships between all geometry (i.e., the query polygons and all 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[4] for the scenes in which we are most interested. The lack of topological information in the skeleton (due to the omission 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 [Pel90] 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* (isotopy) 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 isotopy class, then they must stab the same polygons. Theoretically, this structure can therefore be used to perform a query by searching for an isotopy class that meets our stab-miss criteria.

To put our algorithm into context, we *only* compute the subset of these qualitatively distinct regions relating to those lines intersecting our query pair of polygons. This is explicit in our initial

[4] Primarily due to the numerous test rays cast to each polygon beforehand.

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 *whether* it is blocked. By subtracting blocked regions, rather than 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 that we are interested in 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 somewhat lower. We have demonstrated this experimentally (see Section 6.3).

6.2 Exact Visibility from a 3D Region

The objective of our framework 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.* [KCCO01]. Once again, their solution is for the much simpler $2\frac{1}{2}$D problem. Unlike Koltun *et al.* we do not make use of a multi-tiered hierarchy since the combinatorial complexity of line space in 3D is such that efficiency is gained by querying many small objects, rather than a single large object. Our framework is also different in that it takes advantage of previously computed results. We believe this last technique would also further enhance the algorithm of Koltun *et al.*

6.2.1 Querying Clusters of Geometry

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 surface 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.

6.2.2 Reuse of Parent Line-space

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* [Tel92b][5].

The unobstructed line segments can be extended through the bounding box to form a set of lines. Then, a necessary but not sufficient test of visibility for objects within the bounding box is that they intersect (at least partially) this set of lines.

This allows a fast, conservative, but relatively accurate rejection test to be applied[6]: Each polygon within the bounding box is transformed to its hyper-plane representation in the Plücker coordinate system, and if each stabbing polytope of the previously computed complex (for the bounding box) does not intersect the polyhedron defined by this transformation, then the object is necessarily invisible. This can be computed efficiently, but conservatively, by testing to see whether every stabbing polytope falls exterior to at least one half-space defining the polyhedron. Any objects which pass this test will have to be queried individually.

6.2.3 Virtual Occluders

During the process of computing the visibility status of bounding boxes, opportunities to extract "virtual occluders" arise. As coined by Koltun *et al.* [KCCO00], 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

[5] More precisely, it is the subset of the union between the anti-umbra and anti-penumbra that illuminates only the target polygon

[6] This test is similar to the *tube* interference test used by Teller and Hanrahan [TH93a] for blocker maintenance.

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, in conjunction with the sensitivity of the query algorithm on only the number of effective occluders, 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 [KS00]), is quickly rejected since the relatively large size of the virtual occluders imply that only very few occluders are necessary to confirm invisibility. A screen-shot of our algorithm output, along with various visualisations illustrating this process, is depicted in Figure 47.

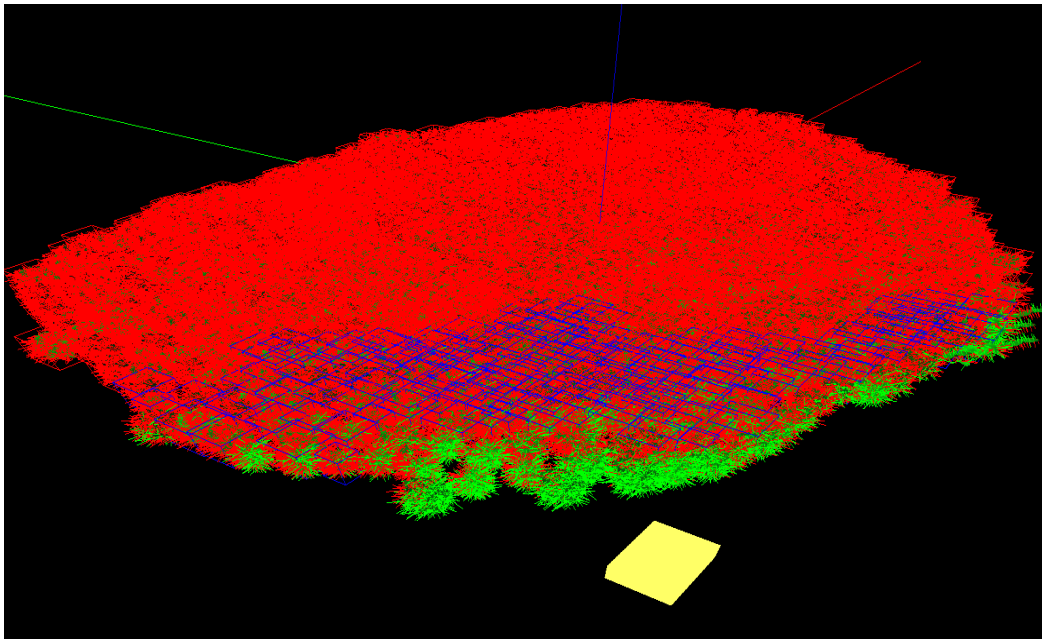


Figure 47: *Virtual Occluders*. Visibility is computed from the cuboid highlighted in yellow. Green polygons are visible, red ones are invisible. Many of the red polygons are surrounded by blue bounding boxes. These are those boxes that were queried and found to be invisible. Those bounding boxes which are trivially visible (found via ray casting), are not displayed. The red bounding boxes are those determined to be occluded by the blue bounding boxes (virtual occluders).

6.2.4 The Framework in Context

Since the publication of the algorithm presented here [NBG02], one other technique by Bittner [Bit02] has been published that computes exact from-region visibility (see Section 2.5.8).

An advantage of the approach taken by Bittner, is that the algorithm computes exactly those fragments of each visible polygon that are visible. This can be applied to the generation of discontinuity meshes. The cost of this flexibility is that it is less scalable than our approach. Our experiments successfully use models that are significantly[7] more complex than those used by Bittner.

One reason for this is the uniform treatment of polygons by Bittner. As a visible polygon is added to the occlusion tree, its dual polytope is tested against the tree, splitting nodes as it is inserted. This process is computationally expensive (see Section 2.5.8). Our framework trivially accepts a vast majority of visible polygons due to our line space sampling heuristic (see Section 6.1.2). This allows computational effort to be concentrated on those objects that are not trivially visible. This, of course, precludes the use of our algorithm for determining the visible fragments of objects.

A second benefit of our approach, is that we use effective occluders (see Section 6.1.2) to determine the occlusion of objects, whereas Bittner always uses the (larger, and thus more inefficient) set of visible polygons as occluders. A similar issue, is that during our front to back traversal of the scene, our technique eventually integrates large virtual occluders (see Section 6.2.3) into the effective set, aggregating the effect of several smaller polygon occluders efficiently. In contrast, the occlusion tree requires that all objects are tested against the visible polygon set. This is somewhat ameliorated by the persistency of the occluder tree, since splitting operations are only required on insertion, where we perform splitting for every non-trivial query. Polytope-hyperplane sidedness tests are still required.

One must take into consideration that a direct theoretical comparison should not be made, between our *query* algorithm and Bittner's *visibility* algorithm since the query algorithm exists in a visibility framework where it may be executed many times per polygonal region, whereas Bittner's algorithm is executed once per directional stratum, per polygonal region, and returns a list of visible polygons for that stratum. It is of key importance to realise that non-trivial visibility queries are executed very selectively, and that when they are executed, they proceed very rapidly[8].

To contrast the difference between our query algorithm, and Bittner's full visibility algorithm, consider the following: Our algorithm incrementally refines the set of *unoccluded* lines. This is

[7] Our scenes are two orders of magnitude larger, and are more likely to generate many visibility events. Also, the average visible set size is approximately two orders of magnitude larger.

[8] We will show experimentally, that the query is only slightly super-linearly dependent on the number of effective occluders.

advantageous, since at any one time, only a representation of the unoccluded line set is resident in memory. Bittner incrementally maintains both the unoccluded line set *and* the currently occluded line set. For the latter, a leaf node exists for every connected set of lines that terminate at (or "see") the same polygon. As noted by Bittner, the algorithm execution time may be as large as $O(n^4 \log n)$, and the tree size is bounded above by $O(n^5)$ (see Section 2.5.8). This leads to heavy computational and memory requirements. The memory requirements are managed by subdividing the set of lines leaving the source region into directional strata.

Effectively, where Bittner's algorithm is coupled to the combinatorial complexity of visibility events in line space, our algorithm is coupled to the complexity of the set of lines representing the *gaps* though which a target polygon may be seen from a region. In most cases these gaps are few, and tend to be simple[9].

For those cases where large gaps do appear, the target polygon is rapidly determined to be visible by ray shooting. It is possible to construct a scenario where a large polygon is visible only through many small holes. This type of scene is rare, but not necessarily contrived. E.g., a forest scene. Such a scene also represents a bad case for Bittner's occlusion tree algorithm, since all the gaps have to represented, along with all the possible interactions of the visible polygons with each other. We have, however, developed a technique that allows for fast and accurate evaluations for even these cases (see Section 7.1.3).

6.3 Results and Discussion

We have implemented the exact visibility query algorithm of Section 6.1.1 and integrated it into the framework proposed in Section 6.2. Our implementation has been tested 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[10]. Our second scene is the *forest* scene used by Durand [Dur99]. 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

[9] Requiring only one polytope for representation.
[10] It has a high vertical complexity.

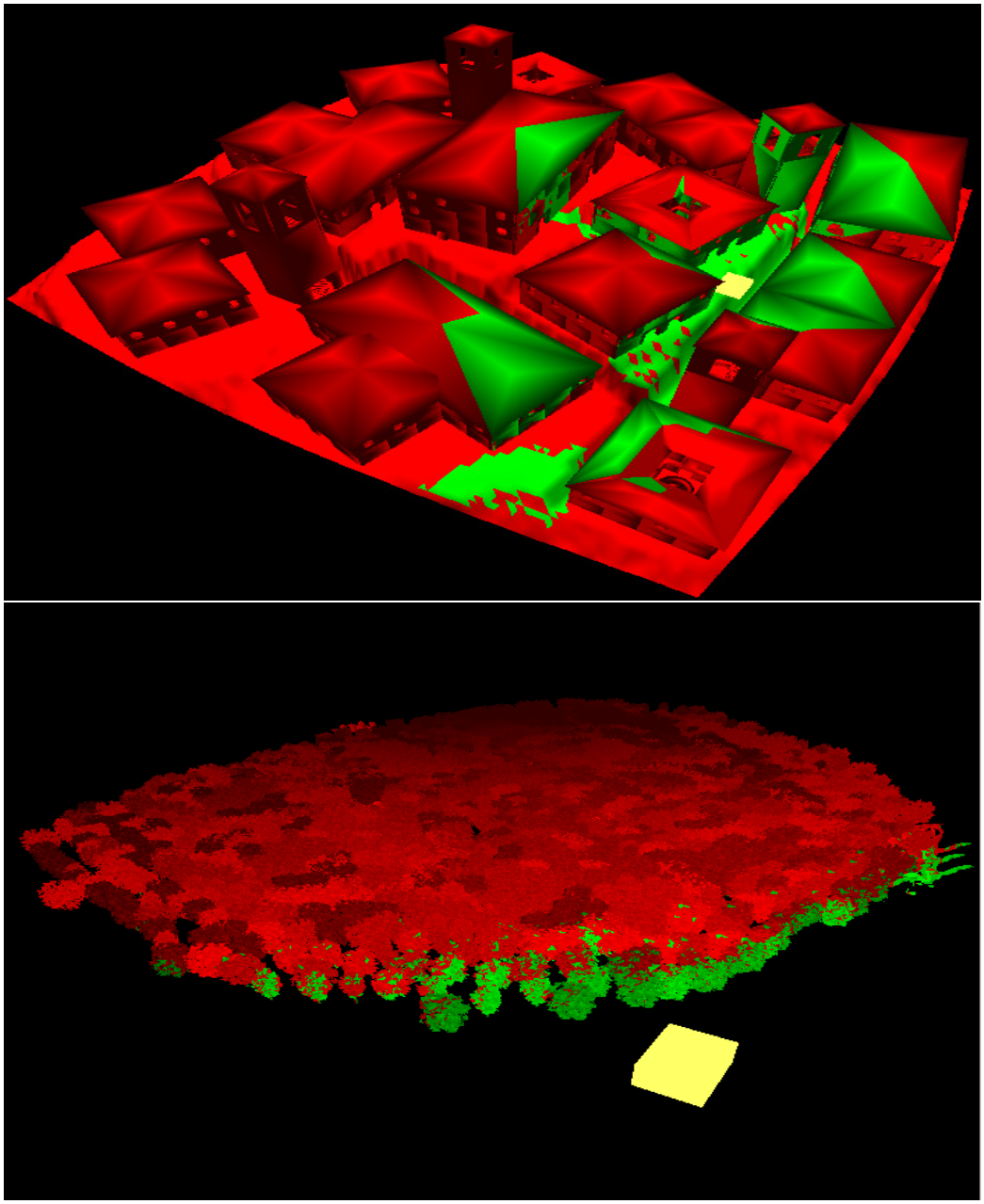


Figure 48: *Algorithm results*. The top image shows a view of our town scene, while the bottom shows a view of the forest. In both images, the yellow block corresponds to a view-cell. The geometry visible from the view cell is rendered in green, while occluded geometry is rendered in red.

Plücker dual polytope. See Figure 48 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 walk-through application. 16384 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 with 1.2GB RAM, and solved for distinct cells in parallel. For both scenes we have culled and timed the visibility of a random selection of 100 cells. These timings on a per-cell basis are presented in Table 9.

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

Table 9: *Experimental result summary.*

To solve for the whole town scene adequately, 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 pre-process is a once-only cost, and is more than offset by the advantages of our approach. Note the large degree of culling achieved (Table 9). We note that our technique determined that the extended projections algorithm of Durand [Dur99, DDTP00] overestimated the average visibility set by a factor of approximately 28.4.

When only a single workstation is available for pre-processing, 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 therefore very loosely coupled, making it suitable even for informal clusters with a low bandwidth infrastructure.

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 (or shaft), and the time complexity of the query algorithm. In order to quantify time complexity, we measured the time taken by the algorithm as a function of our approximation to the number of *effective occluders* (see Section 6.1.2).

We time approximately 48000 queries, running on a single processor. For each of these we count the number of effective occluders m, and compute the average time taken to query this many effective occluders. We use a least-squares fit, and find the growth to be in the order of $O(m^{1.15})$. This is depicted in Figure 49a. To increase the confidence of our fit, we exclude timings corresponding to fewer than 5 samples. The data resulting from this noise reduction is plotted in Figure 49b. We note that this excludes most of the larger queries, since they are infrequent. We further note a distinct inflection where the number of effective occluders is 30. Further investigation has shown this to be caused by our polytope complex becoming larger than the L2 cache of our test machine. This behaviour is reproduced in *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. Given our justification for anticipating such a result (see Section 6.1.3) and the low variance, we do not expect this complexity to vary significantly with any realistic scenes.

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 can 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 extracted).

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 but by incorporating our general framework, 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 an exact 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

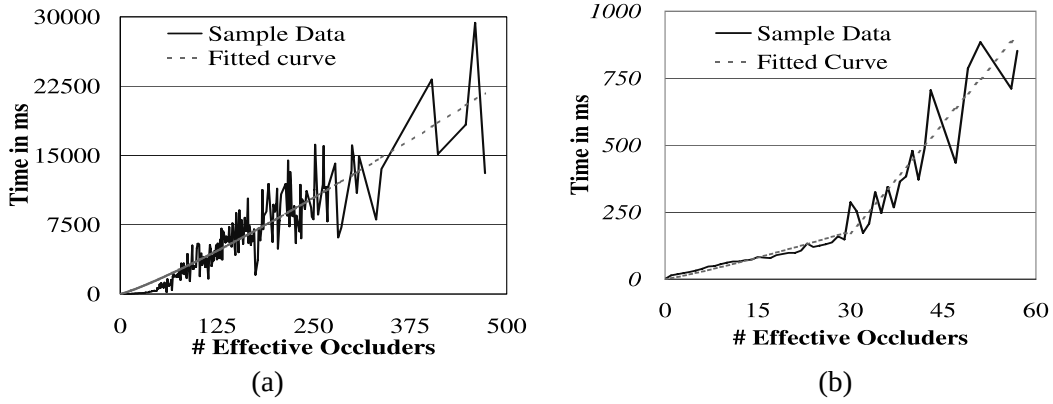


Figure 49:
Time vs. Effective occluders. (a) The average time for queries consisting of varying numbers of effective occluders. The fitting curve shows growth of the order $O(m^{1.15})$. The average distance from the fitting curve to the sample data is 1285(ms). (b) The same data set as for (a), however 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).

dissertation. It is clear however, that the average order is bounded above by $O(nm^{1.15})$, which in itself shows 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, our initial polytope 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 6.2.3) results in approximately 97% of the pre-process 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-time complexity to be a function of what is *visible*. This shows the output sensitivity of our algorithm.

6.4 Conclusion

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

The query algorithm is implemented as a special case of the *selective stabbing* problem we presented and solved in Section 5.

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.

Chapter 7

Conclusion

We have presented two from-region visibility culling algorithms. Existing techniques try to achieve optimal results for the three goals of preprocessing speed, run-time performance and image accuracy. We recognise that the achievement of all these goals is seldom required simultaneously, and that a better overall solution may be found by taking this into consideration.

We present one algorithm that minimises preprocessing time and maximises run-time performance. The cost is a potential loss of image accuracy.

A core component of this algorithm is an aggressive from-surface visibility algorithm. We have developed heuristic error measures that guide an adaptive sampling process in a successful attempt to minimise this error. Experiments show the resulting error to be negligible in practice. However, no non-trivial upper bound is placed on the error.

The from-surface aggressive algorithm is integrated into a sophisticated framework. The framework effectively manages a sample cache, ensuring that only samples that will definitely be required again are retained. The caching scheme allows for the progressive generation of cell partitions at no additional computational cost. The algorithm is output sensitive and has only a logarithmic dependence on the number of view cells.

This algorithm can be used while drafting models (where preprocessing speed is valued over accuracy), or for the effective preprocessing of highly complex models (where run-time performance and preprocessing speed are valued over accuracy). We demonstrate that we are able to process very large, very complex scenes of up to 5 million polygons in less than 80 minutes on a standard workstation, and achieve an average of 91.32% culling with very low error (0.338% on average).

We extend the aggressive algorithm to process 5D ray-space. The goal being a new method to

accelerate ray shooting. Preliminary results show a great reduction in the number of ray-triangle intersections per ray cast (less than 2.5% that of a naïve algorithm). The algorithm is comparable with existing techniques. Unlike these other techniques, however, the traversal stage is very inexpensive. With further optimisation, we believe that this approach may perform better than the current state of the art.

Our second algorithm solves the long standing problem of discovering a tractable solution to exact from-region visibility. The algorithm consists of a novel polygon to polygon visibility query algorithm that is integrated into a powerful framework.

The framework successfully avoids redundant computations by reusing previously computed results whenever possible. This is achieved through the construction of large virtual occluders, and the effective usage of the line space structures generated by prior queries. The query algorithm uses a novel occluder order prioritisation algorithm to select an efficient order for the treatment of occluders.

This algorithm maximises run-time performance and results in no image error. The cost is a lengthy preprocess. Experiments have shown that this cost is due to a large constant computational factor, and that the algorithm is bounded above by $O(nm^{1.15})$, where $m << n$. This shows scalability, however, the algorithm performs better in practice.

This algorithm is suited to computing optimal visibility for any model that has been finalised (where run-time performance and image accuracy are valued over preprocessing time). Since the per-region computations of the algorithm are independent, it may be distributed easily.

We test our algorithm using scenes of moderate to high depth complexity. We demonstrate that our algorithm can solve for very complex scenes, as large as 1.45 million polygons in a time of 70 hours on a standard workstation. We are able to achieve as much as 99.12% culling. This is 28.4 times more effective than the conservative solution of Durand [Dur99, DDTP00].

In the process of solving the exact visibility problem, we have also solved the theoretical problem of *selective stabbing*. This problem is a generalisation of both the stabbing problem and (with minor modification) the exact from-region visibility query. Our solution is efficient, as evidenced by the success of our exact visibility algorithm.

We have proved several novel mathematical results that facilitate our solution to selective stabbing. We determine that the dual form of the set of lines through two polygons is a four dimensional polytope extruded along a particular direction in Plücker space. We provide a *closed form* solution that evaluates both this polytope and this direction.

We have also improved and augmented an existing multi-dimensional polyhedral complex splitting algorithm. We have improved performance, making the algorithm fully sensitive to the zone of the splitting hyperplane. We have further augmented it so that it can be used as an efficient, context aware, multi-dimensional polyhedral subtraction algorithm.

7.1 Future Work

An important milestone in visibility research has been reached by developing an exact from-region visibility solution. In terms of preprocessing for static scenes, we believe the future lies in developing faster exact visibility solutions, and the treatment of scenes with higher order primitives.

As a first step, we expect algorithms to be developed that are essentially conservative or aggressive variants of exact algorithms. The purpose being the reduction of preprocessing time costs. Although this may be seen as a step back towards using suboptimal, multipurpose algorithms, the novelty is that, given sufficient, but finite, time, the solution will become exact. No existing algorithms support this: either suboptimal results are reported, or infinite time is required. The goal of this flexibility is that, given a particular time budget, the best possible solution is found.

7.1.1 Including From-point Acceleration Techniques

In our aggressive algorithm, point sample and area sample rendering contributes a major part of the preprocess. Although we exploit occlusion data that has already been computed, it is possible to use other rendering acceleration technique for further improvements[1]. We intend to use the occlusion testing functionality of recent graphics cards to accelerate the rendering process and we expect this to result in significant time savings.

7.1.2 5D Ray Space Partitioning

We believe that there are several ways in which the ray-space preprocessing can be improved. The algorithm results are dependent on the particular spatial subdivision. An investigation into different subdivision heuristics should result in far fewer ray-triangle intersection tests. In particular, we believe that a ray-space subdivision that seeks to minimisie cell surface area while minimising the subdivision depth, would be highly beneficial.

[1] Our current implementation uses frustum culling only.

7.1.3 Feedback in Selective Stabbing

We intend to explore the possibility of a feedback loop within our exact visibility query algorithm. Instead of precomputing an ordering for occluder subtractions, we believe that considerable improvement may be achieved by using the current state of the complex to sample unoccluded line space.

The result of this, is that those occluders that are to small to be classified as potentially necessary effective occluders, will be found directly. In the case where the sampling reveals that no occluders exist, visibility is established. This is particularly useful for scenes where the query polygons are visible only through small gaps, since the query will terminate when the first gap is encountered (i.e., a sample ray will be generated through the gap, and visibility will be established).

We have demonstrated that our exact algorithm can handle such complex scenes. We believe, however, that with the inclusion of this feedback process, we can still achieve a significant performance increase.

Appendix A

Theorems

In this appendix we present a selection of theorems from within this thesis. The theorems in this section are either non-original, or trivial, and are presented purely for the convenience of the rigorous reader. All proofs in this section are referred to in context within the dissertation.

Theorem A.1 $EP(A) \cup EP(B) \subseteq EP(A \cup B)$

Given an arbitrary point x such that $x \in EP(A) \cup EP(B)$, we have:

$$x \in EP(A) = \bigcap_{W \in cell} view_W A \textbf{ or } x \in EP(B) = \bigcap_{W \in cell} view_W B \quad (41)$$

$$\textbf{by definition} \quad EP(A \cup B) = \bigcap_{W \in cell} view_W (A \cup B) \quad (42)$$

$$\Rightarrow \quad x \in \bigcap_{W \in cell} view_W A \subseteq \bigcap_{W \in cell} view_W (A \cup B) = EP(A \cup B) \quad (43)$$

$$\textbf{or} \quad x \in \bigcap_{W \in cell} view_W B \subseteq \bigcap_{W \in cell} view_W (A \cup B) = EP(A \cup B). \quad (44)$$

$$\Rightarrow \quad x \in EP(A \cup B). \quad (45)$$

$$\Rightarrow \quad EP(A) \cup EP(B) \subseteq EP(A \cup B). \quad (46)$$

To show that these two quantities are not necessarily equal we give a counter example in Figure 50b and Figure 50c.

□

Lemma A.1 *We define $< \vec{x}, \vec{y} >: \mathbb{R}^6 \times \mathbb{R}^6 \to \mathbb{R}$ to be equivalent to $D_{\vec{x}}(\vec{y})$. Only axioms 1, 2 and 3 of the standard inner product axioms are satisfied. These axioms are:*
Axiom 1 *Linearity in the first variable. $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^6, < \vec{x} + \vec{y}, \vec{z} >=< \vec{x}, \vec{z} > + < \vec{y}, \vec{z} >$.*
Axiom 2 *Linearity in the second variable. $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^6, < \vec{x}, \vec{y} + \vec{z} >=< \vec{x}, \vec{y} > + < \vec{x}, \vec{z} >$.*

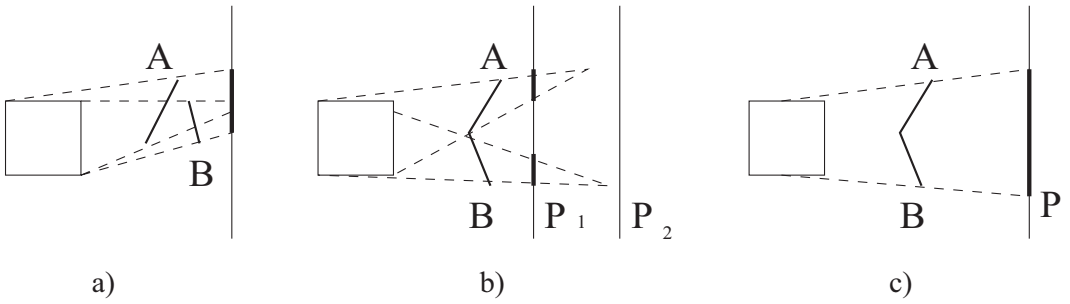


Figure 50: **a)** shows how the extended projections of occluders **A** and **B** can be fused to create a larger extended occluder. **b)** shows the unconnected union of the extended projections of **A** and **B** onto the planes P_1 and P_2 (in which the fusion of the extended projections is the empty set). Note that the size of the extended projections (and hence the union of the extended projections) become smaller as the projection planes progress farther from the cell. Finally, **c)** shows the much larger extended projection of the union of **A** and **B**. Note that extended projection of the unions would grow as P progresses farther from the cell. This is because the extended occluder is larger than the cell (if projected).

Axiom 3 *Commutativity.* $\forall \vec{x}, \vec{y} \in \mathbb{R}^6, < \vec{x}, \vec{y} >=< \vec{y}, \vec{x} >$.
Axiom 4 $\forall \vec{x} \in \mathbb{R}^6, < \vec{x}, \vec{x} >\geq 0$.
Axiom 5 $\forall \vec{x} \in \mathbb{R}^6, < \vec{x}, \vec{x} >= 0$ *iff.* $\vec{x} = 0$

Proof:
The first three axioms make use of the fact that these same axioms are true for the standard inner product (**.**) of $\mathbb{R}^6$.
Axiom 1:

$$< \vec{x}+\vec{y}, \vec{z} >= \begin{pmatrix} x_0+y_0 \\ x_1+y_1 \\ x_2+y_2 \\ x_3+y_3 \\ x_4+y_4 \\ x_5+y_5 \end{pmatrix} \cdot \begin{pmatrix} z_3 \\ z_4 \\ z_5 \\ z_0 \\ z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \cdot \begin{pmatrix} z_3 \\ z_4 \\ z_5 \\ z_0 \\ z_1 \\ z_2 \end{pmatrix} + \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} \cdot \begin{pmatrix} z_3 \\ z_4 \\ z_5 \\ z_0 \\ z_1 \\ z_2 \end{pmatrix} =< \vec{x}, \vec{z} > + < \vec{y}, \vec{z} > \quad (47)$$

Axiom 2:
As for axiom 1.
Axiom 3:

$$< \vec{x}, \vec{y} >= \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \cdot \begin{pmatrix} y_3 \\ y_4 \\ y_5 \\ y_0 \\ y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_3 \\ x_4 \\ x_5 \\ x_0 \\ x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ x_4 \\ x_5 \\ x_0 \\ x_1 \\ x_2 \end{pmatrix} =< \vec{y}, \vec{x} > \quad (48)$$

Axiom 4:

This is clearly false, since every point which does not lie on the Plücker hypersurface will serve as

a counterexample.

Axiom 5:

This is cleraly false, since every point on the Plücker hypersurface (indeed, every real line) will serve as a counterexample.

□

We also prove *distribution*:

Lemma A.2 *Given* $\forall \vec{x}, \vec{y}, < \alpha\vec{x}, \vec{y} >= \alpha < \vec{x}, \vec{y} >$.

Proof:

We use the distributive property of the standard inner product (**.**) of $\mathbb{R}^6$.

$$< \alpha\vec{x}, \vec{y} >= \begin{pmatrix} \alpha x_0 \\ \alpha x_1 \\ \alpha x_2 \\ \alpha x_3 \\ \alpha x_4 \\ \alpha x_5 \end{pmatrix} \cdot \begin{pmatrix} y_3 \\ y_4 \\ y_5 \\ y_0 \\ y_1 \\ y_2 \end{pmatrix} = \alpha \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \cdot \begin{pmatrix} y_3 \\ y_4 \\ y_5 \\ y_0 \\ y_1 \\ y_2 \end{pmatrix} = \alpha < \vec{x}, \vec{y} > \tag{49}$$

□

Lemma A.3 *Using the previous derivation of* $\vec{r}$, $< \vec{r}, \vec{r} >= 0$.

Proof:

$$\vec{r} = \frac{d_x}{\gamma} \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} \begin{pmatrix} 0 \\ n_z \\ n_y \\ q \\ 0 \\ 0 \end{pmatrix} \tag{50}$$

$$< \vec{r}, \vec{r} > = \left(\frac{d_x}{\gamma} \begin{vmatrix} b_y - a_y & b_z - a_z \\ a_y - c_y & a_z - c_z \end{vmatrix} \right)^2 \begin{pmatrix} 0 \\ n_z \\ n_y \\ q \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} q \\ 0 \\ 0 \\ 0 \\ n_z \\ n_y \end{pmatrix} \tag{51}$$

$$= 0 \tag{52}$$

□

Appendix B

Compression

Visibility preprocessing algorithms often generate a significant amount of data that needs to be compressed. It is also often beneficial to compress data during the execution of the preprocess in order to reduce the number of page faults, or even to prevent the possible overflow of address space. This appendix covers the compression techniques that we have used in our various implementations.

These techniques were developed at the beginning of the research presented in this dissertation (i.e., in 1999). Independently, during this time van de Panne and Stewart[vdPS99] developed and later published similar ideas. Where we developed two techniques that take advantage of two types of visibility coherence, van de Panne and Stewart have presented an elegant and uniform solution that exploits both types of coherence simultaneously.

B.1 Taking advantage of spatial coherence

Lossless techniques seek to locate coherence in order to achieve compression. A technique where the differences between the visibility of a group of neighbouring cells and the similarities of the group of cells are evaluated, is presented by Nadler *et al.* [NFLYCO99]. Similarity is evaluated simply as the intersection of the respective visibility sets, while the differences are exposed by those elements that do not fall into the intersection.

The similarities are stored *once* in a parent cell, while the differences are stored in the child cells. The original visibility set for a cell can be reconstructed simply by taking the union of the cell's visibility set and the visibility set of its parent. The lossless technique we present is a generalisation of this approach that is equivalent to the technique proposed by Nadler *et al.* in the worst case.

The coherence exploited here, is the property that nearby viewpoints, and hence nearby cells,

have a tendency to see a similar set of objects.

Viewpoint Coherence Compression Algorithm

Our algorithm works in a similar fashion to that of Nadler *et al.* [NFLYCO99]. Two differentiations being that we do not limit the hierarchy to two levels and that we also take advantage of visibility sets that are *mostly* similar with respect to the visibility of a particular polygon, rather than exactly the same.

In a similar fashion to the filter-bank approach used by wavelet compression, we associate an average set with each parent cell and difference sets with that cells children: A *difference* set of polygons is associated with each cell. These sets are constructed such that the difference sets from a leaf cell to the root (through traversing the leaf cell ancestors) can be decoded to give the set of visible polygons from the cell. The decompression is simple: We begin with the *current* cell being that of the desired leaf cell. If an object index exists in the current cell's parents set and its own set, it is considered invisible, if an index does not exist in either set it is invisible, and if an index exists in only one of either sets, it is considered visible[1]. This process occurs recursively for each ancestor of the leaf, the until the root node is reached. The current cell is promoted to its parent at each iteration.

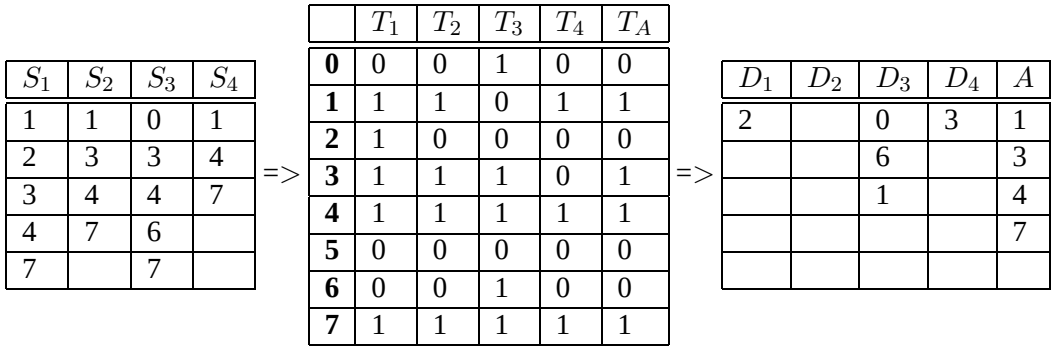
To build the difference sets we assume a bottom up approach. Given a set of adjacent leaf cells (starting point), we let an *average* set be the set of polygons contained in at least half plus one[2] of the leaf selected leaf cells. The difference set is then calculated as being the exclusive-or[3] of average set and the visibility set.

It is the sharing of data in the parent set that gives the compression. If a polygon is visible from most of the children it is stored only once in the parent. In the case where most, but not all of the children sets contain a polygon, the polygon is inserted into the parent set as visible, but it is *also* inserted into the child set. This indicates that the child differs from the parent set by that polygon.

As an example, consider the case where a polygon is visible from seven out of eight child cells of a parent cell. Since seven is greater than half the number of sets plus one (four), the polygon is stored in the parent set. To represent that it is not in one child set, that child set will also contain the polygon. The presence of a polygon in a child set represents its *difference* from its parent set. An example is shown in Figure 51.

[1] This is an exclusive-or relation.
[2] The "one" accounts for the storage of the polygon in the parent list.
[3] Since XOR is its own inverse.

The algorithm often makes use of a tuple representation of visibility sets. These are simply lists whose ith value is true if and only if i is in the original set. Since this relationship is bijective, a visibility set or a tuple can always be reconstructed from the other.



S_1	S_2	S_3	S_4
1	1	0	1
2	3	3	4
3	4	4	7
4	7	6	
7		7	

=>

	T_1	T_2	T_3	T_4	T_A
0	0	0	1	0	0
1	1	1	0	1	1
2	1	0	0	0	0
3	1	1	1	0	1
4	1	1	1	1	1
5	0	0	0	0	0
6	0	0	1	0	0
7	1	1	1	1	1

=>

D_1	D_2	D_3	D_4	A
2		0	3	1
		6		3
		1		4
				7

Figure 51: *Example of our Lossless compression technique.* 17 numbers are reduced to 9. The columns ($S_{1..4}$) in the first table represent the visibility sets to be compressed. The columns ($T_{1..4}$)in the middle row are the same sets in their tuple form, except that the *parent* tuple (T_A) is also appended. An element in this latter tuple is true if three $(4/2 + 1)$ or more of its associated children have a true value. Finally, the tuples are converted back into sets ($D_{1..4}$) after they are XORed with T_A. Notice that D_3 and D_4 did not originally contain 1 and 3 respectively. Decompression can be achieved by recombining the tuple form of any set $D_{1..4}$ with T_A (readily derived from A) using XOR.

Our compression technique can be applied either incrementally, during visibility calculation or subsequently.

van de Panne and Stewart cluster by rows and columns. The clustering by rows is equivalent to the clustering of view cells, and accounts for the same coherence we do. The algorithms differ in the choice of cells and how the clusters are represented. van de Panne and Stewart also present a lossy version of the algorithm that reduces space at the cost of conservativity[4]

B.2 Index-Spatial Coherence Compression

In Chapter 4, many samples of visibility are taken. We have applied the technique above and a 0-based run length encoding (RLE) technique to compress the data. We have found the RLE technique to perform considerably better. As mentioned above, these techniques take advantage of different forms of coherence. The technique presented in Section B.1 takes advantage of the property that view points near each other have a tendency to see similar objects, while the technique presented

[4]The potentially visible sets may be overestimated.

here exploits the property that indices that are adjacent tend to be associated with spatially related[5] primitives. Such primitives have a tendency to all be visible or invisible from most view points[6].

The result of this coherence, is that in a *visibility tuple*[7], the bits typically form in clusters.

A standard RLE compression algorithm that takes advantage of such runs would work very well. However, we use a 0-based RLE algorithm that only encodes runs of zeros, but encodes them more cheaply than other RLE algorithms. This is to take advantage of depth complexity, since high depth complexity is usually a feature of scenes to which visibility algorithms are applied. Such scenes result in tuples that are almost entirely zero.

Once again, the algorithm of van de Panne and Stewart capture this coherence in terms of their column clustering. Their choice of clusters and actual compression technique (gzip) differ from ours.

[5]For example, triangle n and triangle $n+1$ usually share an edge or vertex, thereby implying that the two triangles are near.

[6]For example, two triangles representing a window of a building will both be either visible of invisible from most view points.

[7]A structure of n bits (for n primitives), where the nth bit is set *iff* the nth primitive is visible.

Bibliography

[AF92] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8:295–313, 1992.

[AF96] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

[Aga92] Pankaj K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *SIAM Journal on Computing*, 21(3):540–570, 1992.

[AGS00] T. Asano, S. K. Ghosh, and T. Shermer. Visibility in the plane. *Handbook in Computational Geometry*, pages 829–876, 2000.

[AK87] James Arvo and David Kirk. Fast ray tracing by ray classification. In *Computer Graphics, Annual Conference Series (SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64. ACM, July 1987.

[Ame92] Nina Amenta. Finding a line transversal of axial objects in three dimensions. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 66–71. Society for Industrial and Applied Mathematics, 1992.

[APS93] B Aronov, M Pellegrini, and M Sharir. On the zone of an algebraic surface in a hyperplane arrangment. *Discrete and Computational Geometry*, 9(2):177–188, 1993.

[ARJ90] John M. Airey, John H. Rohlf, and Frederick P. Brooks Jr. 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.

[AS93] Pankaj K. Agarwal and Micha Sharir. Ray shooting amidst convex polytopes in three dimensions. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 260–270. ACM Press, 1993.

[ASVNB00] Carlos Andújar, Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum*, 19(3):499–506, August 2000. ISSN 1067-7055.

[BHS98] J. Bittner, V. Havran, and P. Slavk. Hierarchical visibility culling with occlusion trees. In *Computer Graphics International 1998*. IEEE Computer Society, June 1998.

[Bit02] Jiří Bittner. *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University in Prague, October 2002.

[BKES00] Fausto Bernardini, James T. Klosowski, and Jihad El-Sana. Directional discretized occluders for accelerated occlusion culling. *Computer Graphics Forum*, 19(3):507–516, August 2000. ISSN 1067-7055.

[BMH98] Dirk Bartz, Michael Meißner, and Tobias Hüttner. Extending graphics hardware for occlusion queries in opengl. *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 97–104, August 1998. Held in Lisbon, Portugal.

[BMH99] Dirk Bartz, Michael Meißner, and Tobias Hüttner. Opengl-assisted occlusion culling for large polygonal models. *Computers & Graphics*, 23(5):667–679, October 1999. ISSN 0097-8493.

[BNRSV01] P. Brunet, I. Navazo, J. Rossignac, and C. Saona-Vásquez. Hoops: 3d curves as conservative occluders for cell visibility. *Computer Graphics Forum*, 20(3), 2001.

[BP96] Chandrajit L. Bajaj and Valerio Pascucci. Splitting a complex of convex apolytopes in any dimension. In *12th Symposium on Computational Geometry*, pages 88–97. ACM, May 1996.

[BWW01] 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.

[CCOL98] Y. Chrysanthou, Daniel Cohen-Or, and Dani Lischinski. Fast approximate quantitative visibility for complex scenes. In *Computer Graphics International 1998*, Hannover, Germany, June 1998. IEEE Computer Society.

[CCOZ98] Yiorgos Chrysanthou, Daniel Cohen-Or, and Eyal Zadicario. Viewspace partitioning of densely occluded scenes. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 413–414. ACM Press, 1998.

[CEG+96] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, and Jorge Stolfi. Lines in space: Combinatorics and algorithms. *Algorithmica*, 15(5):428–447, May 1996.

[CG85] Michael F. Cohen and Donald 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.

[CKS02] Wagner T. Corrêa, James T. Klosowski, and Cláudio T. Silva. iWalk: Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University, 2002. Submitted for publication.

[COCSD03] Daniel Cohen-Or, Yiorgos Chrysanthou, Cláudio T. Silva, and Frédo Durand. A survey of visibility for walkthrough applications. *IEEE TVCG*, 9(3):412–431, July-September 2003.

[COFHZ98] 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.

[COKT02] Daniel Cohen-Or, Shuly Lev-Yehudiand Adi Karol, and Ayellet Tal. Inner-cover of non-convex shapes. In *Proceedings of the 4th Israel-Korea Bi-National Conference on Geometric Modeling*, 2002.

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

[COZ98] D. Cohen-Or and E. Zadicario. Visibility streaming for network-based walkthroughs. In *Graphics Interface*, pages 1–7, 1998.

[CT96] Satyan Coorg and Seth Teller. Temporally coherent conservative visiblity. In *12th Annual ACM Symposium on Computational Geometry*, pages 78–87, 1996.

[CT97] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluder. In *Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[DD02] Florent Duguet and George Drettakis. Robust epsilon visibility. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 567–575. ACM Press, 2002.

[DDP96] Frédo Durand, George Drettakis, and Claude 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.

[DDP97a] F. Durand, G. Drettakis, and C. Puech. 3d visibility made visibly simple. In *In video 13th Annual ACM Symposium on Computational Geometry*, 1997.

[DDP97b] Frédo Durand, George Drettakis, and Claude Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In *Proceedings of SIGGRAPH 97*, pages 89–100. ACM, 1997.

[DDP97c] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex: a unified datastructure for global visibility of scenes of polygons and smooth objects. In *Canadian Conference on Computational Geometry*, August 1997.

[DDP98] F Durand, George Drettakis, and Claude Puech. Visibility driven hierarchical radiosity, 1998. SIGGRAPH technical sketch.

[DDP99] Frédo Durand, George Drettakis, and Claude Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics (TOG)*, 18(2):128–170, 1999.

[DDP02] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex. *ACM Transactions on Graphics (TOG)*, 21(2):176–206, 2002.

[DDS03] Xavier Décoret, Gilles Debunne, and François Sillion. Erosion based visibility preprocessing. In *Proceedings of the Eurographics Symposium on Rendering*. Eurographics, 2003.

[DDTP00] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.

[DF94] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 223–230. ACM Press, 1994.

[Dur99] Frédo Durand. *3D Visibility, analysis and applications*. PhD thesis, U. Joseph Fourier, 1999. http://graphics.lcs.mit.edu/˜ fredo.

[Ede87] H Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[EOS86] H Edelsbrunner, J O'Rourke, and R Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal of Computing*, pages 341–363, 1986.

[FP96] K. Fukuda and A. Prodon. Double description method revisited. *Combinatorics and Computer Science*, 1120 of Lecture Notes in Computer Science:91–111, 1996.

[FR94] K Fukuda and V Rosta. Combinatorial face enumeration in convex polytopes. *Computational Geometry*, 4:191–198, 1994.

[FST92] Tom Funkhouser, Carlo H. Séquin, and Teller. Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, volume 25(2), pages 11–20, March 1992.

[GCS91] 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.

[GK93] Ned Greene and M. Kass. Hierarchical z-buffer visibility. *Proceedings of SIGGRAPH 93*, pages 231–240, 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[GM90] 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.

[Gre96] Ned Greene. Hierarchical polygon tiling with coverage masks. *Proceedings of SIGGRAPH 96*, pages 65–74, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[GSF99] Craig Gotsman, Oded Sudarsky, and Jeffrey A. Fayman. Optimized occlusion culling using five-dimensional subdivision. *Computers & Graphics*, 23(5):645–654, October 1999. ISSN 0097-8493.

[HA00] Nicholas Holzschuch and Laurent Alonso. Using graphics hardware to speed-up visibility queries. *Journal of Graphics Tools*, 5(2):33–47, 2000.

[Hai00] Eric Haines. Shaft culling tool. *Journal of Graphics Tools*, 5(1):23–26, 2000.

[Han94] Andrew J. Hanson. Geometry in n dimensions. In Paul S. Heckbert, editor, *Graphics Gems IV*. AP Professional (Academic Press), Boston, 1994.

[Hav00] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[HDS03] Denis Haumont, Olivier Debeir, and Franois Sillion. Volumetric cell-and-portal generation. In *Computer Graphics Forum*, Eurographics Conference Proceedings. Blackwell Publishers, 2003.

[Hec92] Paul Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pages 203–216, May 1992.

[HMC+97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 1–10. ACM Press, 1997.

[HT92] M. Hohmeyer and S. Teller. Stabbing isothetic rectangles and boxes in $O(n \log n)$ time. *Computational Geometry Theory and Applications*, 4:201–207, 1992.

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

[KCCO00] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel 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.

[KCCO01] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel 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.

[KCO00] Vladlen Koltun and Daniel Cohen-Or. Selecting effective occluders for visibility culling. In *Eurographics (short presentations track)*, pages 165–169, 2000.

[KS99] James T. Klosowski and Cláudio 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.

[KS00] James T. Klosowski and Cláudio T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, 2000.

[KS01] J. T. Klosowski and Cláudio T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):365–379, October - November 2001. ISSN 1077-2626.

[KvD76] J Koenderink and A.J. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, 24(1):51–59, 1976.

[KvD79] J Koenderink and A van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32(1):211–216, 1979.

[LCCO03] Alan Lerner, Yiorgos Chrysanthou, and Daniel Cohen-Or. Breaking the walls: Scene partitioning and portal creation. In *Pacific Graphics*, 2003.

[LG95] David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. *1995 Symposium on Interactive 3D Graphics*, pages 105–106, April 1995. ISBN 0-89791-736-7.

[LSCO03] Tommer Leyvand, Olga Sorkine, and Daniel Cohen-Or. Ray space factorization for from-region visibility. *ACM Transactions on Graphics*, 22(3):595–604, July 2003.

[MMB98] Daniel Meneveaux, Eric Maisel, and Kadi Bouatouch. A new partitioning method for architectural environemnts. *Journal of Vizualisation and Computer Animation*, 9(4):195–213, 1998.

[MO88] M McKenna and J O'Rourke. Arrangements of lines in 3-space: a data structure with applications. In *The 4th Annual Symposium on Computational Geometry*, pages 371–380, 1988.

[NAT90] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP trees yields polyhedral set operations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 115–124. ACM Press, 1990.

[NBG02] S. Nirenstein, E. Blake, and J. Gain. Exact from-region visibility culling. In *Proceedings of the 13th workshop on Rendering*, pages 191–202. Eurographics Association, June 2002.

[NFLYCO99] Boaz Nadler, Gadi Fibich, Shuly Lev-Yehudi, and Daniel Cohen-Or. A qualitative and quantitative visibility analysis in urban scenes. *Computers & Graphics*, 23(5):655–666, 1999. ISSN 0097-8493.

[NR03] Isabel Navazo and Jarek Rossignac. Shieldtester: Cell-to-cell visibility test for surface occluders. In *Eurographics*, 2003.

[OLBN01a] Gary Oberholster, John Lewis, Edwin Blake, and Shaun Nirenstein. An aggressive parallel visibility preprocessor. Technical Report CS01-22-00, University of Cape Town, 2001. http://www.cs.uct.ac.za/Research/CVC/projects/parvis/paper.pdf.

[OLBN01b] Gary Oberholster, John Lewis, Edwin Blake, and Shaun Nirenstein. Aggressive parallel visibility preprocessor. Undergraduate thesis report, University of Cape Town, 2001. http://www.cs.uct.ac.za/Research/CVC/projects/parvis/report.pdf.

[PD87] H. Plantinga and C. R. Dyer. The asp: a continuous viewer-centered representation for 3d object recognition. In *The First International Conference on Computer Vision*, pages 626–630, 1987.

[PD90] W. Plantinga and C. Dyer. Visibility, occlusion and the aspect graph. *International. Journal of Computer Vision*, 5(2):137–160, 1990.

[Pel90] Marco Pellegrini. Stabbing and ray shooting in 3-dimensional space. In *6th ACM Symposium on Computational Geometry*, pages 177–186. ACM Press, 1990.

[Pel93] Marco Pellegrini. Ray-shooting on triangles in 3-dimensional space. *Algorithmica*, 9:471–494, 1993.

[Pel97] Marco Pellegrini. Ray-shooting and lines in space. *CRC Handbook of Discrete and Computational Geometry*, pages 599–614, 1997.

[Pla92] Harry Plantinga. An algorithm for finding the wealky visible faces from a polygon in 3d. In *Fourth Canadian Conference on Computational Geometry*, pages 45–51, 1992.

[Pla93] Harry Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3d scenes. In *Graphics Interface*, pages 166–173, 1993.

[Pla98] William Harry Plantinga. *The asp: a continuous, viewer-centered object representation for computer vision*. PhD thesis, The University of Wisconsin, 1998.

[Plü65] J. Plücker. On a new geometry of space. *Philosophical Transactions of the Royal Society*, 155:725–791, 1865.

[PS85] F P Preparata and M I Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[Pu98] Fan-Tao Pu. *Data Structures for Global Illumination and Visibility Queries in 3-Space*. PhD thesis, University of Maryland, College Park, MD, 1998.

[PV93] Michel Pocchiola and Gert 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.

[Reg02] Ashu Rege. Occlusion (hp and nv extensions). http://www.nvidia.com/dev_content/gdc2002/GDC2002_occlusion_files/frame.htm, 2002.

[RW80] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 110–116, 1980.

[SD94] G. Simiakakis and A. M. Day. Five-dimensional adaptive subdivision for ray tracing. *Computer Graphics Forum*, 13(2):133–140, 1994.

[SDDS00] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000. ISBN 1-58113-208-5.

[SG93] A. James Stewart and Sherif Ghali. An output sensitive algorithm for the computation of shadow boundaries. In *Fifth Canadian Conference on Computational Geometry*, pages 291–296, August 1993.

[SG94] A. James Stewart and Sherif Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 231–238. ACM Press, 1994.

[SG99] Oded Sudarsky and Craig Gotsman. Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):13–29, January - March 1999. ISSN 1077-2626.

[SHN+03] Adrian Sharpe, Matthew Hampton, Shaun Nirenstein, James Gain, and Edwin Blake. Accelerating ray shooting through aggressive 5d visibility preprocessing. In *Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*, pages 95–100. ACM Press, 2003.

[SOG98] N. Scott, D. Olsen, and E. Gannett. An overview of the visualize fx graphics accelerator hardware. *The Hewlett-Packard Journal*, pages 28–34, 1998.

[Som59] D.M.Y. Sommerville. *Analytical Geometry of Three Dimensions*. Cambridge University Press, 1959.

[SSG89] D. Salesin, J Stolfi, and L. Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 208–217. ACM Press, 1989.

[SSS74] Evan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys (CSUR)*, 6(1):1–55, 1974.

[Sto91] Jorge Stolfi. *Oriented Projective Geometry : A Framework for Geometric Computations*. Academic Press, 1991. ISBM 0126720258.

[Sud] Oded Sudarsky. Personal communication.

[Sud98] Oded Sudarsky. Dynamic scene occlusion culling. Technion – Israel Institute of Technology, January 1998. Research Thesis (Doctor of Science).

[SVNB99] 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.

[Tel92a] Seth Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley, 1992.

[Tel92b] Seth 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.

[TH93a] Seth Teller and Pat Hanrahan. Visibility computations for global illumination algorithms. In *Computer Graphics (Proceedings of SIGGRAPH 93)*, volume 27, pages 239–246, July 1993.

[TH93b] Seth Teller and Michael 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.

[TH99] Seth Teller and Michael Hohmeyer. Determining the lines through four lines. *Journal of Graphics Tools*, 4(3):11–22, 1999. ISSN 1086-7651.

[TL01] Eyal Teler and Dani Lischinski. Streaming of complex 3d scenes for remote walkthroughs. *Computer Graphics Forum*, 3(20), 2001.

[TN87] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 153–162. ACM Press, 1987.

[TS91] Seth J. Teller and Carlo 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.

[vdPS99] Michael van de Panne and James Stewart. Efficient compression techniques for precomputed visibility. *Eurographics Rendering Workshop 1999*, June 1999. Held in Granada, Spain.

[WA77] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics (Proceedings of SIGGRAPH 77)*, 11(2):214–222, July 1977. Held in San Jose, California.

[WBP98] Yigang Wang, Hujun Bao, and Qunsheng Peng. Accelerated walkthroughs of virtual environments based on visibility preprocessing and simplification. *Computer Graphics Forum*, 17(3):187–194, 1998. ISSN 1067-7055.

[WWS00] Peter Wonka, Michael Wimmer, and Dieter 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.

[WWS01] Peter Wonka, Michael Wimmer, and François Sillion. Instant visibility. In *Eurographics*. Eurographics, A. Chalmers and T.-M. Rhyne, 2001.

[YR96] Roni Yagel and William Ray. Visibility computation for efficient walkthrough of complex environments. *PRESENCE*, 5, 1:1–16, 1996.

[Zha98] Hansong Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, University of North Carolina at Chapel Hill, 1998.

[ZMHI97] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff 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.

Index