Postgraduate Study Report DC-PSR-99-05

Hierarchical Techniques for Visibility Determination

Jiří Bittner

Supervisor: *Pavel Slavík*

March 1999

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University
Karlovo nám. 13
121 35 Prague 2
Czech Republic

email: bittner@fel.cvut.cz
WWW: http://www.cgg.cvut.cz/~bittner/

This report was prepared as a part of the project

Hierarchical Techniques for Output Sensitive Visibility Determination

This research has been supported by grants of the Ministry of Education of the Czech Republic number 1252/1998 and the Internal Grant Agency of the Czech Technical University number 309810103.

....

Jiří Bittner
postgraduate student

....

Pavel Slavík
supervisor

Table of contents

Preface

This report is by no means a classical research paper. Its main contribution is to briefly summarise the work in the field visibility determination over last two decades and present some results I have made during the first two years of my postgraduate studies. Another goal of the report is to provide a valuable background, that organises ideas for the intended dissertation thesis.

Visibility is essential for most techniques of image synthesis and as such it has been studied very intensively in the past. Jim Blinn, a pioneer of computer graphics, summarised the development of the field during past two decades in his keynote contribution to the 25th annual Siggraph conference [19]. With little exaggeration he points out that all easy problems in CG have already been solved. Although, in general visibility problems are not particularly easy, they must have been addressed to obtain computer generated images of three dimensional scenes already thirty years ago. Nowadays, the problem of visibility is very often considered as solved. In many aspects, I must admit this is indeed true. Nevertheless, as the computational resources have developed dramatically in the last decade a lot of traditional approaches are now seen under very different circumstances. We are confronted with large amounts of available data that we would not only like to visualize, but to do it as realistically and as fast as possible.

The key concepts that appear throughout the report are the use of coherence and the output–sensitivity. Coherence or continuity is common property of the real world. Obviously, this property is reflected in problems concerning the a virtual counterpart of the real world. A key observation is that algorithms exploiting coherence can be significantly more effective than those assuming completely unorganised data. The output–sensitivity is another term that becomes increasingly important as the amount of data to be processed increases even faster than the power of available computational resources. The running time of an ideal output–sensitivity algorithm would depend solely on the complexity of the solution and would be completely independent of the size of the input of the algorithm.

The topics discussed in the report cover a large part of computer graphics research. Obviously, there is a tradeoff between a level of detail of description and the number of discussed approaches. I have decided to provide a brief general discussion followed by description of the proposed solution to a specific visibility problem. The report covers visibility problems that have been usually studied separately. I feel important to discuss these problems together, although currently only few links among them are addressed. Finally, I hope this decision will be justified in my future work.

Prague, 19th February　　　　Jiří Bittner

Hierarchical Techniques for Visibility Determination

Jiří Bittner
bittner@fel.cvut.cz
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University
Karlovo nám. 13
121 35 Prague 2
Czech Republic

Abstract

This report introduces a new classification of visibility problems in three dimensions based on the dimension of the space of lines inherited in the particular problem. Three classes of visibility problems are studied — visibility along a line, visibility from a point, and visibility from a region. For each class an algorithm extending previous results is presented. In particular the following algorithms are proposed: ray shooting using kD–tree with neighbour links (trees), hierarchical visibility culling with occlusion trees, and determination of visibility from a polygonal region using a regional occlusion tree. Finally, it is outlined how the results can be used in a unified framework for the hierarchical visibility determination.

Keywords

Visibility, occlusion, ray casting, hidden surface removal, real–time rendering, discontinuity meshing, potentially visible sets, conservative algorithms, hierarchical algorithms, kD–tree, BSP tree.

1 Introduction

One of the ultimate goals of computer graphics is to synthesise realistic images of three dimensional virtual environments, i.e. environments represented as formal models in computer memory [154]. To achieve results that correspond to the perception of the real world the *lightning* (or *illumination*) within the virtual environment must be simulated. The crucial point of this simulation is to determine the light interaction between certain points or regions in the virtual environment. The phenomenon of *visibility* is a major factor influencing this interaction.

The *physical optics* studies properties of light propagation considering the wave character of light. It was shown that even for simple scenes the light simulation based of physical optics is extremely computationally expensive [118]. The work presented in this report corresponds to the *geometrical optics* as we assume that a light–ray propagates "straight" through the environment until it is terminated and rescattered on an object with appropriately modified intensity. Each ray can be associated with certain line* and thus the interaction of lines and lines sets with objects in the virtual environment is crucial for any algorithm of image synthesis. *Visibility determination* aims to solve and describe this interaction so that it can be efficiently used in the scope of rendering.

We have studied three major classes of visibility problems that can be distinguished by the dimensionality of the line–space involved. In general a line intersecting certain region C

*A line is associated with infinitely many rays, that differ in their origin along the line.

in the three dimensional environment can be parametrised by two points on a sphere that contains C. There are four degrees of freedom (DOF) in the description of the line and thus the corresponding line–space is four dimensional ($\mathcal{L} \subset \mathcal{R}^4$). Fixing certain line–parameters (e.g. direction) the corresponding line–space forms a k–dimensional subspace of $\mathcal{L}$ with $k < 4$.

In particular we have studied the following three classes of visibility problems in three dimensional environments:

1. **Visibility along a line** (VAL).
   $\mathcal{L}_{VAL} = \{l\}$.
   Examples: visibility of two points, ray shooting.

2. **Visibility from a point** (VFP).
   $\mathcal{L}_{VFP} \subset \mathcal{R}^2$.
   Examples: point–to–region visibility, visibility map, hidden surface removal.

3. **Visibility from a region** (VFR).
   $\mathcal{L}_{VFR} \subset \mathcal{R}^4$.
   Examples: region–to–region visibility, aspect graph, discontinuity meshing.

The first class of the visibility problems involves a single line. The *point-to-point* visibility query determines whether the line segment between two points is occluded, i.e. has an intersection with an opaque object. Similar visibility query is known as ray shooting (or ray casting). Given a ray we ask which object is first pierced by the ray. Although the queries are similar they differ in the type of the result: ray shooting is a constructive algorithm, whereas point–to–point visibility is a decisive one. Obviously, point–to–point visibility can be solved easily using ray shooting. These queries are used extensively in ray tracing as well as the modern stochastic global illumination algorithms.

The second class of visibility problems involves line–space of dimension two. In this case lines can be parametrised using certain projection of the environment. The VFP class of visibility problems has attracted great attention in the past. This is natural, since viewing the environment from a point (eye, camera) is a fundamental process of the visual perception and consequently the last step of the light propagation simulation *. As opposed to the VAL problems there are infinitely many lines involved. The solutions to the problems may differ upon using a line–space that is continuous or discrete.

A typical problem belonging to the VFP class is known as *hidden surface removal.* Informally, the hidden surface removal aims to draw an image so that objects are drawn in correct depth order, i.e., visible objects cover the invisible ones. In addition, the *visibility map* can be constructed that is a graph encoding topological relationships of visible objects. An example of the decisive VFP algorithms is the *point–to–region visibility.* It classifies a region as visible, invisible or partially visible with respect to the viewpoint and a collection of objects.

The third class involves the most general and the most complex visibility problems. The four dimensional nature of problems belonging to the VFR class makes data structures and algorithms rather complex, memory demanding, and prone to robustness. Under certain circumstances these problems can be solved by solving lower dimensional ones. A typical VFR problem concerns the determination of potentially visible set (PVS) for a given region. Another example is the *discontinuity meshing* used within the *radiosity* global illumination algorithm. It partitions surfaces in the environment so that every piece of the partition (patch) “sees” a topologically equivalent view (back-projection) of an areal light source.

* Obviously an eye or camera does not have a focus that corresponds to a mathematical point, but it is usually a satisfying approximation.

In this report we want to give an overview about typical problems belonging to each of the three classes mentioned above. Further, we present solutions that improve previous results for selected problems. Finally, we briefly address relationships among the problems discussed in the report and outline how the solutions could be used in a unified framework.

The rest of the report is organised as follows: Chapter 2 introduces some basic definitions and terminology. In Chapter 3 the VAL class of visibility problems is discussed. Chapter 4 is devoted to the VFP problems. In Chapter 5 the VFR problems are studied. Finally, Chapter 6 summarises the results and concludes with outlines for future work.

2 Basic Definitions

In this chapter we present definitions of important elementary terms that we use further in the text. We do not attempt to provide formal definitions, but rather to clarify the notion to a reader who is not very familiar with the fields of computational geometry and computer graphics. Most of the definitions are adopted from the book of Preparata and Shamos [136].

Definition 1 *Point, vector.* A d–tuple $(x_1, x_2, ..., x_d)$ denotes a point p in $\mathcal{R}^d$. This point can also be interpreted as a d–component *vector* with respect to the origin of $\mathcal{R}^d$ which has p as a terminus point.

Definition 2 *Line, plane, linear variety.* Given two distinct points q_1 and q_2 in $\mathcal{R}^d$, the linear combination

$$\alpha q_1 + (1 - \alpha) q_2 \quad (\alpha \in \mathcal{R})$$

is a *line* in $\mathcal{R}^d$. More generally, given k linearly independent points $q_1, q_2, ..., q_k$ in $\mathcal{R}^d$ $(k \leq d)$, the linear combination

$$\alpha_1 q_1 + \alpha_2 q_2 + \alpha_{k-1} q_{k-1} + (1 - \alpha_1 - \cdots - \alpha_{k-1}) q_k \quad (\alpha_j \in \mathcal{R}, j = 1, ..., k - 1)$$

is a *linear variety* of dimension $(k - 1)$ in $\mathcal{R}^d$. A linear variety of dimension $(d - 1)$ in $\mathcal{R}^d$ is called *hyperplane.* We call a hyperplane in $\mathcal{R}^3$ a *plane.*

Definition 3 *Convex combination, line segment.* Given two distinct points q_1 and q_2 in $\mathcal{R}^d$, the convex combination of q_1 and q_2 can be expressed as

$$\alpha q_1 + (1 - \alpha) q_2 \quad (\alpha \in \mathcal{R},\ 0 \leq \alpha \leq 1).$$

This convex combination describes a *line segment* $\overline{q_1 q_2}$ joining q_1 and q_2.

Definition 4 *Convex set, convex hull.* A set of points $\mathcal{S} \subset \mathcal{R}^d$ is convex iff for any two points $q_1, q_2 \in \mathcal{S}$ the line segment $\overline{q_1 q_2}$ is entirely contained in $\mathcal{S}$. The *convex hull* of a set of points $\mathcal{U} \subset \mathcal{R}^d$ is the boundary of the smallest convex set in $\mathcal{R}^d$ containing $\mathcal{U}$.

Definition 5 *Polygon, vertex, edge, supporting plane.* A polygon P in $\mathcal{R}^d$ is defined by a finite set $\mathcal{E}$ of line segments (*edges*) such that every segment extreme (*vertex*) is shared by exactly two edges and no subset of edges has the same property. All the edges of $\mathcal{E}$ must form a subset of a 2–dimensional linear variety that we call a *supporting plane* of P.

Definition 6 *Simple, convex polygon.* A polygon P is *simple* if there is no pair of non–consecutive edges of P sharing a point. A simple polygon is *convex* if its interior forms a convex set.

Definition 7 *Polyhedron.* * The *polyhedron* in $\mathcal{R}^d$ is defined defined by a graph of its k–dimensional faces (k–faces) with ($k < d$). Each k–face with $k > 0$ forms a k–dimensional polyhedron. The 0–faces are called *vertices*, the 1–faces are the *edges* and the $(d-1)$–faces are the *facets* of the polyhedron. In $\mathcal{R}^3$ we call the facets of the polyhedron simply *faces.*

Definition 8 *Simple, convex polyhedron.* A polyhedron is *simple* if no pair of non–adjacent k–faces shares an n–face with $0 \leq n < k$. A simple polyhedron is *convex* if its interior is a convex set.

Definition 9 *Object, scene.* An *object* is a set of points $\mathcal{S}$ in $\mathcal{R}^d$ (possibly infite) that is finitely specificable. We assume $\mathcal{S}$ is compact and the object specification is simple (such as for polyhedra or implicit objects — sphere, cone,...). A *scene* is a collection of objects, their material properties and light sources.

Definition 10 *Ray.* A *ray* in $\mathcal{R}^d$ is a half–open line segment described by its point of origin $o \in \mathcal{R}^d$ and direction vector $w \in \mathcal{R}^d$. The ray can be expressed as:

$$o + \alpha w \quad (\alpha \in \mathcal{R}, \alpha \geq 0).$$

Definition 11 *Region.* A *region* in $\mathcal{R}^d$ is a compact set of points in $\mathcal{R}^d$ that is finitely specificable.

Definition 12 *Sight–line.* A *sight–line* between two points $q_1, q_2 \in \mathcal{R}^d$ is a line segment $\overline{q_1 q_2}$ that is unoccluded, i.e. it does not intersect any object in the scene.

Definition 13 *Point–to–point visibility.* Two points q_1 and q_2 in the scene are visible if the line $\overline{q_1 q_2}$ is a sight–line. Otherwise q_1 and q_2 are invisible.

Definition 14 *Point–to–region visibility.* Region A is visible from point p iff all points $q \in A$ are visible from p. Region A is *partially visible* from p iff there are at least two points $q_1, q_2 \in A$ such that q_1 is visible from p whereas q_2 is invisible. Region A is *invisible* from p iff no point $q \in A$ is visible from p.

Definition 15 *Region–to–region visibility.* Two regions A, B are visible iff all pairs of points $a \in A, b \in B$ are visible. A and B are invisible iff no points $a \in A, b \in B$ are visible. Finally, A and B are *partially visible* iff there are two distinct pairs of points $(a_1, b_1), (a_2, b_2),\ a_1, a_2 \in A; b_1, b_2 \in B$, such that points a_1 and b_1 are visible, whereas a_2 and b_2 are invisible.

Definition 16 *Binary Space Partitioning Tree.* A binary space partitioning (BSP) tree T in $\mathcal{R}^d$ is a binary tree defined for set of objects $\mathcal{S}$. Each node v in T represents a convex polyhedral region (cell) R_v. The region associated with the root of the tree is the convex hull (or a bounding box) of all objects from $\mathcal{S}$. Each interior node v of T is assigned a partitioning plane H_v, that divides R_v into two smaller cells R_v^+ and R_v^-. Let H_v^+ be the positive halfspace and H_v^- the negative halfspace bounded by H_v. The cells associated with the left and the right children of v are $R_v \cap H_v^+$ and $R_v \cap H_v^-$, respectively. The left subtree of v is a BSP tree for the set of objects $S_v^- = \{s \cap H_v^- \neq \emptyset | s \in S_v\}$, the right subtree is defined similarly. Each leaf node l contains a list of objects S_l that intersect R_l.

Definition 17 *kD–tree.* A *kD–tree* in $\mathcal{R}^d$ is BSP tree with all partitioning planes aligned with one of the coordinate axis a $(0 \leq a < d)$.

*A precise definition of the polyhedron is more complex. See [20] for an example.

3 Visibility Along Line

The problems of visibility along a single line seem rather simple since all visual events that may occur, are restricted to the given line. Many modern methods of image synthesis form the final image by sampling visibility between two or a chain of points in the scene. To obtain visually convincing results without noise a huge amount of samples must be taken. Thus the speed of the underlying visibility algorithm is crucial for the performance of the whole image synthesis process.

Two types of problems belonging to the VAL class are common in practice. The aim of the *point–to–point* visibility is to determine if two points are visible, i.e. if the line segment between them is unoccluded. The point–to–point visibility can be solved by means of a more general problem known as *ray shooting* or *ray casting*.

Further in this chapter essentials of ray shooting acceleration techniques are described. We discuss two state–of–art acceleration techniques for ray shooting that are based on uniform grids and kD–trees, respectively. In particular, we focus on kD–trees enriched by *neighbour–links* and *neighbour–trees*; structures that are used to exploit spatial coherence in the scope of the tree traversal. Two new improvements are introduced — the *boundary trees* and *spatial subdivision cleanup*. We give a comparison of experimental results between techniques based on the kD–tree and the uniform grid.

3.1 Previous work

The ultimate goal of photo–realistic image synthesis is to solve the *rendering equation* [99]. A rather crude approximation can be obtained by ray tracing [8] that (in its original version) simulates only specular reflections and point light sources. Many modern methods solve the rendering equation by Monte Carlo integration [9, 199, 101, 163, 200]. All these methods invoke a huge amount of ray shooting queries to sample visibility within the scene. Some algorithms based on the *finite element approach* use ray shooting to approximate visibility between certain regions (patches) [201, 69, 154, 163, 169, 168]. Ray shooting is also used within many hybrid approaches [156, 103, 104, 96, 195].

A naive ray shooting algorithm would test all objects for intersection with a given ray to find the closest visible object along the ray in $\Theta(n)$ time. For complex scenes even the linear time complexity is very restrictive since a huge amount of rays is needed to synthesise an image.

Researchers from the computational geometry community aim to establish tight bounds of the time and memory complexities of the proposed algorithms. In order to obtain theoretically provable results the scene description is usually restricted to polygons. Mark De Berg presents an algorithm [12] that reaches $O(\log n)$ time complexity with $O(n^{4+\epsilon})$ preprocessing time and storage, where ϵ is an arbitrarily small positive constant, and n is a number of polygons. It was shown that the $O(\log n)$ time complexity is asymptotically optimal [181]. The worst–case–optimal results are important theoretically, but the initial assumptions, the storage complexity and implementation problems usually disallow their use for real world data.

A comprehensive overview of practical acceleration techniques for ray shooting can be found in [8], a recent survey is presented in [158]. In these works the average time of the ray shooting query is stressed and the techniques presented are based on observations and heuristics.

Most of the proposed methods organise the scene objects in spatial indexing data structures, that allow efficient determination of objects that can be possibly pierced by a given ray. The goal is to minimise the number of ray/object intersection tests. Examples of such structures are uniform grids [64], bounding volume hierarchies [73], octrees [144, 132], kD–trees [67, 115, 176], irregular grids [157], or hierarchies of sorted lists [61]. Recently, hierarchical grids have been introduced [25, 26, 102]. Hybrid approaches combining several hierarchical structures were proposed in [174, 175]. Methods that are based on subdivision of line–space [131] (or ray–space) were introduced in [7, 159, 108]. The main disadvantage of these methods is the high memory complexity of a discrete subdivision of the higher dimensional (4D or 5D) space.

Some researchers have studied methods that trace infinitely many rays that are enclosed by a *ray–beam* [90, 198, 137, 65, 185]. In our classification these methods correspond to the VFP visibility problems that will be discussed in the next chapter.

For the purpose of ray tracing acceleration, methods that aim to reduce the number of rays to be traced have been proposed. Firstly, the coherence of radiance along similar rays can be utilised [3, 111, 104, 186]. Secondly a visibility predetermination can avoid casting rays between invisible regions (points) [85, 207, 114].

Further acceleration of ray shooting can be achieved by speeding up the ray/object intersection itself. This is usually carried out by object bounding volumes such as simple axis aligned boxes. A different approach that uses duality to speedup the ray/polyhedron intersection was introduced in [105].

Another topic concerns the usage of temporal coherence within ray–traced animation sequencies [72, 98, 29, 97, 13, 81]. The resulting image is usually very similar to the previous one in the sequence and also the sets of rays cast in the successive frames are usually similar.

In the next section we present an overview of using a uniform grid to accelerate ray shooting queries. In Section 3.3 we discuss usage of kD–trees for ray shooting. Further in this section methods that use *neighbour links* and *neighbour trees* are presented. Experimental results measured using an implementation of the proposed methods are evaluated in Section 3.4. Finally, Section 3.5 concludes the chapter.

3.2 Ray shooting using Uniform Grid

The *uniform grid* (UG) is one of the first accelerating data structures for ray shooting [64]. It subdivides the scene into equally sized cells. Within each cell a list of objects that intersect the cell is maintained. Since the UG does not adapt to the distribution of the scene geometry, typically a large number of cells must be created most of which are usually vacant.

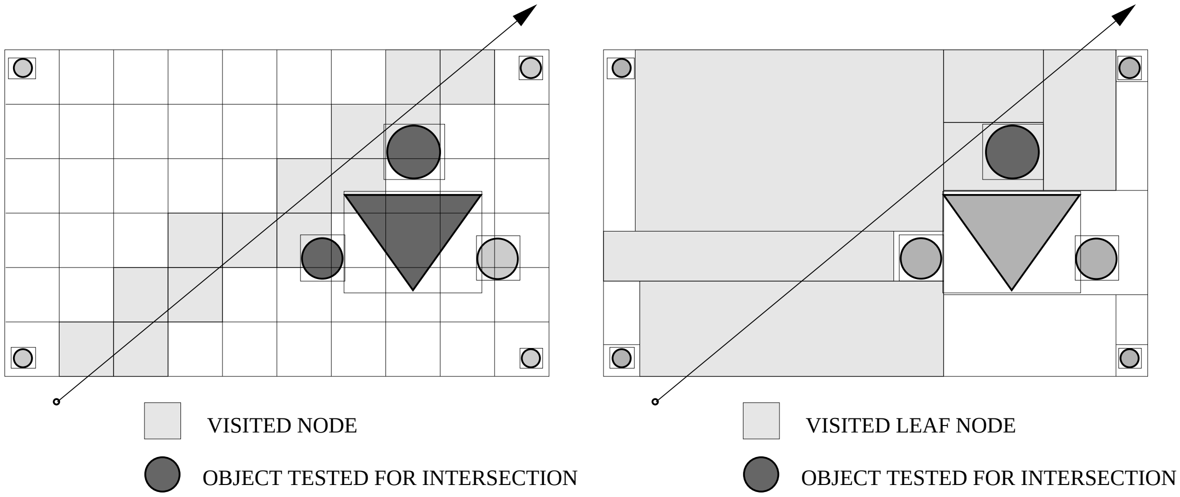
On the other hand the ray shooting algorithm for the UG can be implemented very efficiently using a 3D–DDA [5, 64, 41]. Starting from the cell containing the ray origin the UG is traversed along the ray direction. Within each cell objects contained in the corresponding list are tested for intersection with the ray (see Figure 1). The traversal is terminated when an object intersecting the ray is found or all relevant cells of the UG have been visited. Typically many cells contain no objects, thus the traversal algorithm needs to perform many steps until the intersection is found. Methods using certain description of "safe zones" to overcome this problem have been proposed in [53, 49, 142].

3.3 Ray shooting using kD–tree

KD–tree is a common data structure used to organise multidimensional data [145]. The fundamental factor influencing the properties of a kD–tree is the positioning of partitioning planes. The construction of kD–trees for ray shooting purposes was first studied by Kaplan [100]. A more elaborate method of MacDonald and Booth [115] uses a *surface area heuristics* to estimate the *cost* of the resulting tree. The kD–tree is built using a greedy algorithm that minimises the expected cost of the tree when choosing a partitioning plane.

3.3.1 Ray Traversal Algorithms

In [100] Kaplan introduced the *sequential* traversal algorithm. An efficient *recursive* algorithm was introduced by Jansen in [95] and further elaborated by Arvo [6], and Sung and Shirley [179].



(a) (b)

Figure 1: (a) ray shooting using the uniform grid. (b) ray shooting using the kD–tree.

A recursive algorithm performs a constrained depth first traversal (DFT) starting from the root of the tree. The traversal order and its constraints are determined by the mutual position of the partitioning plane corresponding to the currently visited node and the ray. The ability of the kD–trees to adapt to the scene geometry is paid by the overhead of the traversal of the interior nodes of the hierarchy. Recall the UG traversal where only a single step is performed to locate the next cell pierced by the ray. Additionally, even if the location of the origin of the ray is known * the recursive traversal proceeds again from the root of the tree.

A method overcoming these problems was introduced in [115], where neighbour links for kD–trees were proposed and used in the scope of non–recursive traversal algorithm. The paper states almost no evaluation of the method based on experimental analysis. We discuss this approach in more detail and present results obtained by implementation.

3.3.2 Neighbour Links for kD–tree

Each leaf of a kD–tree corresponds to an axis-aligned *leaf–cell* with six rectangular *leaf–faces*. The cells intersecting a *leaf–face* F of cell C are called *neighbours* of C on F. A *unique–neighbour* N on F is a node, which corresponds to a smallest cell which contains F completely. A *neighbour link* is a link from a leaf–face to its unique–neighbour.

There are two mutual relations for a given leaf–face and the corresponding neighbour leaf–cells. Firstly, there is only one neighbour leaf–cell on the leaf–face that is also a unique–neighbour. Secondly, there are more neighbour leaf–cells. In this case the unique–neighbour is an interior node of the kD–tree (see Figure 2).

Construction Algorithm

For each leaf–cell of the kD–tree six neighbour–links are established. For a given leaf–face the hierarchy is searched starting from the root node. In each step the search continues in the subtree, which corresponds to a cell intersecting the face. The search is terminated if a leaf is reached or the face is split by the partitioning plane referred in the currently reached node (i.e., the face intersects both sub–cells).

*This is common for ray tracing, where secondary and shadows rays originate from the last intersection point.

Assume that the kD–tree has n leaves and its average depth is $O(\log n)$. For each leaf–face the down traversal takes $O(\log n)$ steps on average. It is applied on $6n$ faces, hence the time–complexity of the neighbour links construction algorithm is $O(n \log n)$.

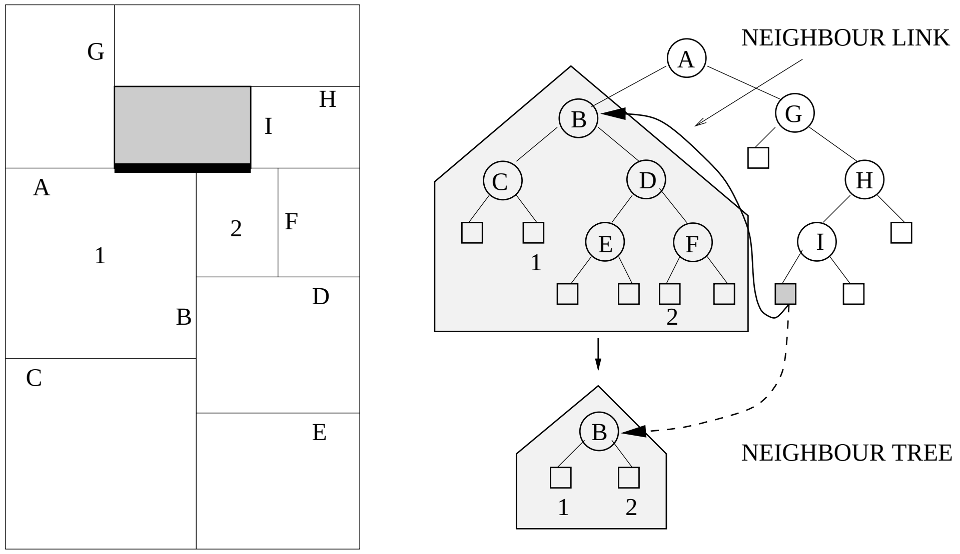


Figure 2: The neighbour link and the neighbour tree determined for the highlighted face the marked cell.

Neighbour Trees

The neighbour links are used to locate the next cell visited in the scope of the ray traversal algorithm. The link points either to a leaf (*direct link*) or an interior node of the tree (*indirect link*). An indirect link can be replaced by a neighbour–tree. The neighbour tree on face F is formed by partitioning planes $\mathcal{P}$ that intersect F. These planes are a subset of the planes of the tree rooted at the unique neighbour of F. Planes $\mathcal{P}$ form a two dimensional partition of that face.

The algorithm of neighbour tree construction is straightforward. Starting from the appropriate unique–neighbour it performs a constrained DFT on the kD–tree. Only nodes corresponding to cells intersecting the face are visited during the DFT. Partitioning planes of the visited nodes that intersect the face are added to the constructed neighbour–tree.

We have observed that the average number of neighbour cells per face is bound by a small constant in practice. In such a case the algorithm takes $O(n)$ time, requiring $O(n)$ memory for neighbour trees.

Boundary Trees

The neighbour trees are used to partition faces of the cell of the kD–tree in order to locate the next cell pierced by the ray. If the origin of the ray is located outside cell C_R corresponding to the root of the tree (scene bounding box), the first leaf–cell pierced by the ray must be located differently. In order to speedup this computation we compute the *boundary trees* that can be viewed as neighbour trees constructed for each of the six faces of C_R. The boundary trees eliminate traversal of nodes of the hierarchy that are irrelevant for the location of the "first–hit" leaf–cell. The construction of the boundary trees proceeds as the construction of neighbour trees using each face of C_R and the root node of the kD–tree as the corresponding neighbour–link.

Traversal Algorithm

The traversal algorithm that exploits the neighbour links or neighbour trees consists of two main components: the *exit–face determination* and the *point location*.

Without loss of generality assume that the traversal starts in a leaf–cell. If the ray does not intersect any object in the cell it is required to locate the next leaf–cell pierced by the ray. First we determine the *exit–face*, that is the face of the current leaf–cell intersected by the ray along its direction. Knowing the exit–face the *exit–point* on the face is computed. Then the neighbour link corresponding to the exit–face is followed. If it is an indirect link, the point location must be applied. The point location is a simple down traversal of the tree the link points to. This tree can be either a subtree of the main kD–tree or a neighbour tree (if it was established).

If the ray traversal is terminated in a leaf–cell, the corresponding leaf can be used to start the traversal for all rays spawned from the terminated ray. This is fundamental for the ray tracing algorithm, where large number of shadow and secondary rays can be cast.

If the origin of the ray is outside the cell C_R corresponding to the root of the kD–tree, the closest intersection of the ray and the cell is computed. The intersection point P (if any) and the corresponding face F of C_R is used to locate the first leaf–cell pierced by the ray. The leaf–cell is found using the point location using the boundary tree established for F. Alternatively, the point location can be applied starting at the root of the whole kD–tree.

In [115] it was shown that for an octree there is less than two neighbour leaf–cells per leaf–face on average. This result does hold for a general kD–tree since there can be as much as $O(n)$ neighbour leaf–cells on average. Nevertheless, preliminary results supported by a simulation show that the average number of neighbour leaf–cells is small in practice (< 4) and independent of the depth of the tree. This observation is very important, since it bounds the average size of the neighbour trees as well as the average number of the point location steps (down traversal of the neighbour tree). However, it seems difficult to find proof for an expected average number of the neighbour leaf–cells for the case of the general kD–tree.

3.3.3 Spatial Subdivision Cleanup

During the construction of a kD–tree, axis aligned bounding boxes of scene objects are utilised. A bounding box is repeatedly used to determine if the corresponding object belongs to a given axis–aligned cell or to establish a new partitioning plane. It might be the case that the bounding box is very loose (especially for oblong and rod objects). Hence cells of the spatial subdivision can contain links to objects, that actually do not intersect these cells. Although, the situation can be solved using tighter bounding volumes [203], it would make the preprocessing and the hierarchy construction more demanding. Instead, we propose the following simple technique. The spatial subdivision is constructed using the axis–aligned bounding boxes of objects as usual. After the construction all objects referred in a given cell are precisely checked for an intersection with the cell. If an object does not intersect the cell, it is removed from the corresponding object list.

The intersection of an object and an axis–aligned cell is performed using precise intersection test that uses the actual geometry of the object. These tests are rather simple for most geometric primitives (see for example [77]). Due to the lack of space we do not discuss the intersection tests in detail.

3.4 Results

In this section we present and discuss efficiency of three methods of ray shooting acceleration used within our GOLEM ray tracing system. We have implemented the 3D–DDA algorithm for uniform grids [5], the kD–tree recursive traversal algorithm [95] [179], and the traversal algorithm for the kD–tree with neighbour trees.

Δ	N_C	number of (non–hierarchical) cells of spatial data structure
	$R_{ETNC}[\%]$	(number of empty cells)/(number of all cells)
	$R_{EVWV}[\%]$	(volume of empty cells)/(volume of whole scene)
	N_{ADC}	(number of references to objects)/(number of objects) - 1
	N_{AOIFC}	average number of objects in non-empty cells
	N_{RPRT}	the ratio of all intersection tests performed to minimal intersection tests
	N_{AT}	the average number of traversal steps per ray
Λ	T_{CB}	the time required to build up spatial data structure
	T_{TR}	the time of rendering itself (only intersections + traversal)

Table 1: Efficiency Measure: n-tuple Δ and Λ

The experimental measure $< \Lambda, \Delta >$ introduced in [87] was used to compare the efficiency of the three methods. The meaning of the variables for this measure is briefly described in Table 1. We have used scenes from the *Standard Procedural Database* introduced by Haines [84]. All images were rendered in 513×513 resolution and the maximal ray–recursion depth of ray tracing was set to 4. The concept of *mailboxes* [5] was used in the scope of all acceleration methods. The grid resolution was based on a set of measurements with different strategies [102]) and only results obtained for the grid resolution that led to the best performance are presented. The measurements were conducted on Pentium–MMX based PC, 166 MHz, 32 MBytes RAM, running under *Linux* operating system.

Discussion

The results in Table 2 show that the ray shooting algorithm using the kD–tree built according to surface area heuristics performs superior compared to Kaplan's kD–tree. This difference is rather significant, when objects are non–uniformly distributed in the scene (scene balls and tree).

The grid performance is comparable to the kD–tree algorithm only if the objects are uniformly distributed (gears, mount, rings). In other cases the lack of the adaptability of the UG causes significantly worse performance results.

The traversal algorithm for kD–trees exploiting the neighbour trees decreases the total rendering time by 10 to 20 %. There is an apparent reduction of traversal steps (N_{AT}). The ratio of the number of interior nodes of all neighbour trees to the number interior nodes of BSP tree varies between 3.77 and 6.27. This ratio indicates the storage needed for the neighbour trees. Since the size of interior node of neighbour tree is much smaller than the size of a leaf of kD–tree (where also additional information is stored), the total memory consumed is increased by factor smaller than two.

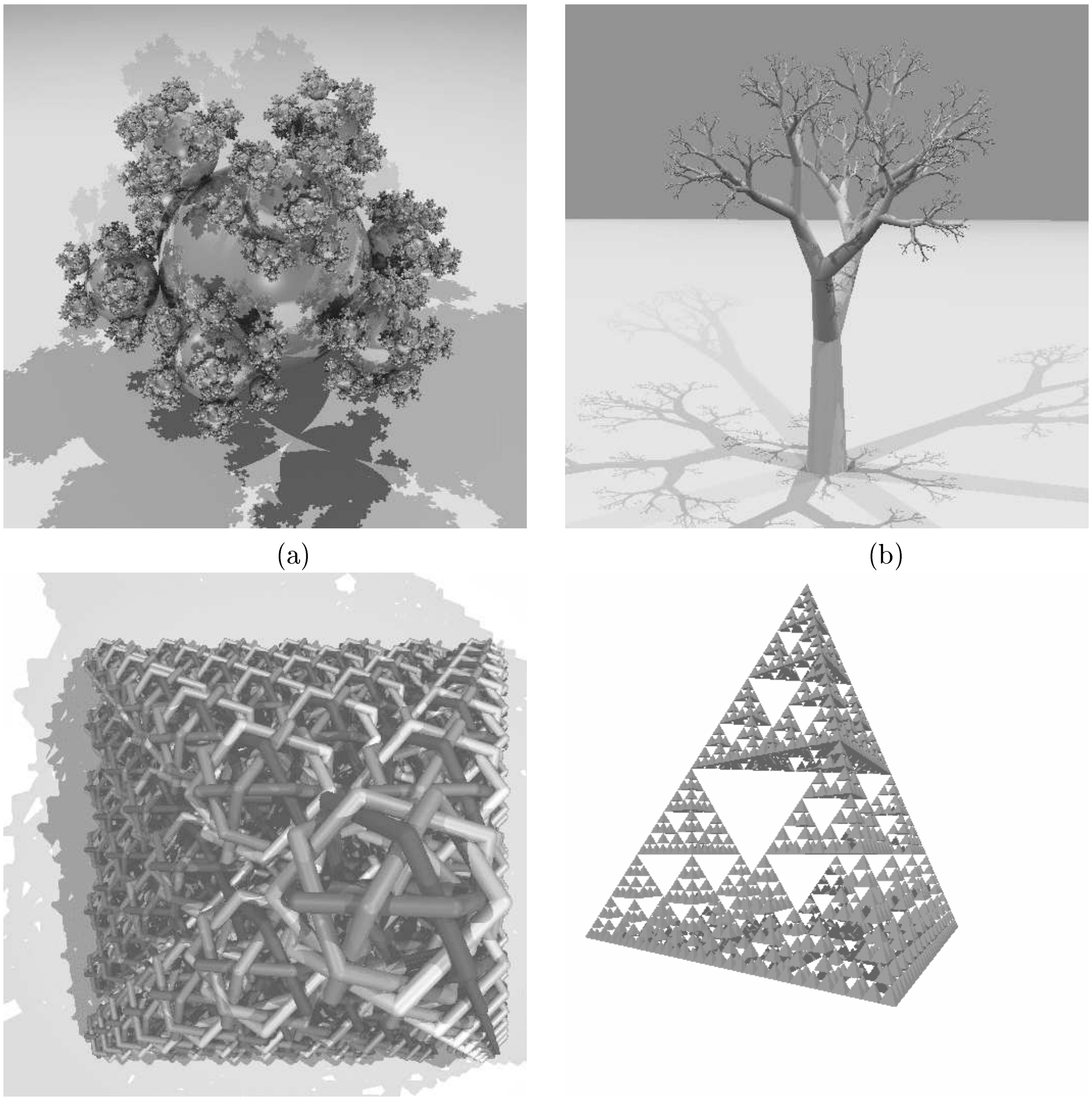
3.5 Conclusions

In this chapter we have discussed visibility problems involving a single line. We have focused on ray shooting, that is fundamental for ray tracing and other global illumination algorithms. We have outlined the concepts of uniform grids and kD–trees for ray shooting acceleration.

A state–of–art method for ray shooting using kD–trees, that uses neighbour links and neighbour trees was described. This method eliminates certain overhead of the traversal of a hierarchical structure such as kD–tree. We have proposed two simple improvements of the method.

Finally, we compared the efficiency of ray shooting using uniform grids, "regular" kD–trees built according to Kaplan [100] and kD–trees constructed using the surface area heuristics [115]. For kD–trees the asset of neighbour links and neighbour trees was evaluated.

The results obtained indicate better efficiency of the adaptive hierarchical structures (kD–tree) compared to the uniform spatial subdivisions (UG). Further, we have shown that augmenting the spatial hierarchy by information about neighbouring relationships leads to further increase of ray shooting efficiency, since the corresponding algorithm makes a better use of spatial coherence.



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

Figure 3: Some of the SPD–scenes rendered using the GOLEM rendering system: (a) balls, (b) tree, (c) rings, (d) tetra.

scene	method	n-tuples Δ and Λ								
		N_C	R_{ETNC}	R_{EVWV}	N_{ADC}	N_{AOIFC}	N_{RPRT}	N_{AT}	T_{CB}	T_{TR}
		$[-]$	$[\%]$	$[\%]$	$[-]$	$[-]$	$[-]$	$[-]$	$[s]$	$[s]$
balls	A	1245	29.6	56.6	1.04	17.2	207.0	65.7	1.17	315.6
(7382)	B	8950	16.7	0.2	1.63	2.61	13.0	25.0	3.53	68.4
	B_C	8950	16.7	0.2	1.17	2.14	11.1	25.3	5.32	68.2
bbox=	$C(2.02)$	8950	16.7	0.2	1.63	2.61	13.0	13.3	4.46	56.7
(24,24,	$C_C(2.02)$	8950	16.7	0.2	1.17	2.14	11.1	13.6	6.42	54.5
1.35)	$G_9^{152}(152)$	207936	88.6	88.6	3.59	1.43	84.7	17.4	1.45	159.5
gears	A	21484	21.5	48.3	10.7	6.48	24.1	31.6	2.94	373.0
(9345)	B	22341	4.1	9.8	4.90	2.58	7.89	22.4	3.5	337.6
	B_C	22341	4.1	9.8	4.5	2.41	7.75	22.4	13.9	327.1
bbox=	$C(1.59)$	22341	4.1	9.8	4.5	2.41	9.62	8.09	5.84	309.5
(4,4,1)	$C_C(1.59)$	22341	4.2	9.8	4.5	2.41	9.37	8.12	16.8	296.0
	$G_{25}^{98}(99)$	242550	85.7	85.7	8.17	2.47	12.6	19.1	2.61	369.4
mount	A	23981	20.1	76.0	9.37	4.44	17.8	30.1	2.3	84.2
(8196)	B	6999	18.9	85.8	0.27	1.83	6.79	22.3	1.58	60.6
	B_C	6999	18.9	85.8	0.22	1.77	6.57	22.3	1.76	60.3
bbox=	$C(1.62)$	6999	18.9	85.8	0.27	1.83	7.23	10.1	2.29	48.4
(2.2,2.2,	$C_C(1.62)$	6999	18.9	85.8	0.22	1.77	7.01	10.1	2.56	47.6
2.1)	$G_{35}^{35}(35)$	42875	87.3	87.3	3.59	6.89	23.6	15.2	1.33	91.5
rings	A	36696	7.7	68.4	14.2	3.77	36.8	60.4	3.3	253.0
(8401)	B	24102	10.8	70.0	5.68	2.61	16.5	39.7	4.37	167.3
	B_C	24102	10.8	69.9	3.76	1.86	13.3	39.7	13.3	157.6
bbox=	$C(1.81)$	24102	10.8	70.0	5.68	2.61	16.5	18.8	7.03	136.0
(19.7,19.3,	$C_C(1.81)$	24102	10.8	69.9	3.76	1.86	13.3	19.0	16.1	126.4
19.7)	$G_{61}^{61}(59)$	219539	84.4	84.4	12.6	3.32	31.7	33.4	2.92	207.5
tetra	A	34552	35.7	91.5	26.0	4.98	48.1	34.2	2.25	17.0
(4096)	B	2991	65.8	97.0	0.0	4.0	10.5	15.0	0.59	8.92
	B_C	2991	65.8	97.0	0.0	4.0	10.5	15.0	0.58	8.74
bbox=	$C(1.36)$	2991	65.8	97.0	0.0	4.0	10.5	10.5	0.72	8.19
(2,2,2)	$C_C(1.36)$	2991	65.8	97.0	0.0	4.0	10.5	10.5	0.75	8.25
	$G_{39}^{38}(39)$	57798	88.1	88.2	9.31	6.16	54.8	18.0	1.11	16.4
tree	A	458	64.9	55.7	0.268	64.5	2009	71.0	1.07	2090
(8191)	B	5514	32.9	0.0	0.531	3.39	24.1	14.4	4.11	54.2
	B_C	5514	32.9	0.0	0.482	3.28	23.7	14.4	5.33	54.2
bbox=	$C(2.01)$	5514	32.9	0.0	0.531	3.39	24.1	8.41	4.67	49.1
(100,100,	$C_C(2.01)$	5514	32.9	0.0	0.482	3.28	23.7	8.42	5.73	48.6
3.2)	$G_{25}^{25}(25)$	15625	95.9	95.9	0.167	14.7	8916	23.5	0.81	10160

Table 2: The n-tuple Δ and Λ for SPD scenes: Methods: A - Kaplan's BSP tree with cyclic change of splitting plane ($x \mapsto y \mapsto z$) placed in spatial median with recursive ray traversal, B - surface area heuristics BSP tree with recursive ray traversal, B_C - surface area heuristics BSP tree with recursive ray traversal and s.s. cleanup [88], $C(x)$ - surface area heuristics BSP tree with neighbour trees ray traversal (x - average number of neighbour leaf–cells per face), $C_C(x)$ - surface area heuristics BSP tree with neighbour trees ray traversal and s.s. cleanup (x as for $C(x)$), $G_y^x(z)$ - grid with 3D–DDA traversal, (x, y, z) is the grid resolution for all axes.

4 Visibility from Point

Visibility from a point has been studied intensively from early days of computer graphics. Depending on the treatment of the two dimensional line–space involved in VFP problems the corresponding algorithms can be divided into two classes: the continuous and the discrete ones. The discrete algorithms utilise the fact that the final solution of the VFP is usually an array of pixels and obtain the solution by sampling within each pixel. Therefore aliasing artifacts can occur if the sampling resolution is insufficient. The complexity of the data structure keeping the solution usually does not grow with the number of objects, but is predetermined by the number of discrete samples. This can be advantageous in certain situations, but can cause wasting of coherence. The continuous algorithms do not suffer from aliasing artifacts, but are usually difficult to implement robustly. The output of continuous algorithms can be overly detailed involving many small features that contribute negligible to the final solution. Obviously, some algorithms use a combination of the discrete and continuous methods.

The complexity of scenes has grown rapidly in the past decade. Many of such scenes exhibit a high–depth–complexity *, what means that many objects are invisible from a given point. Thus, it is very important to design algorithms that are *output–sensitive*,i.e., algorithms with the complexity proportional to the complexity of the visible portion of the scene.

4.1 Previous work

Charles Grant [76] presented a nice survey and classification of visibility algorithms for computer graphics concentrating on variants of the hidden surface removal and "sharp–shadow" algorithms. The hidden surface removal was studied more theoretically by de Berg [12, 50], Mulmuley [120], and Sharir [152]. Parallelisation of the hidden surface removal was proposed by Georges [70] and Franklin [62].

Further in the text, we restrict our discussion to the most popular algorithms nowadays and practical algorithms that aim to achieve the output–sensitivity or to increase the amount of coherence exploited.

The commonly used *depth–buffer* [60] (or z-buffer) is a typical discrete VFP algorithm. It solves the visibility for each pixel of the screen by comparing *depth–values* of objects that project to the pixel. The depth–values can be obtained easily during the object rasterization. In a way it resembles the *ray casting* algorithm, that casts rays corresponding to each pixel of the final image. Nevertheless, there are two major differences. The rasterization process used in the depth–buffer algorithm exploits the *image space coherence* by incremental interpolation of colour, depth, normal, etc. Using ray casting a ray is cast through the scene rather independently of the "neighbouring" rays. On the other hand the ray casting algorithm that uses certain spatial subdivision is output–sensitive: theoretically invisible objects are not tested for intersection with any ray.

Another popular VFP algorithm is based on BSP trees [63], that combines the continuous and discrete methods. The BSP–tree preserves a topological information about a polygonal scene in a binary tree. The tree is build in the object space and is used to obtain a *depth ordering* of the polygons with respect to any point in the scene. The polygons are processed in back–to–front order and the discrete *painter's algorithm* [60] is applied to resolve pixel visibility by overdrawing its previous contents.

The *depth–buffer* and the BSP assisted *painter's* algorithm are not output–sensitive since they must process (rasterize) every object in the scene †. Thus they may spend significant

* Many objects are pierced by each viewing ray.

† Obviously some trivial rejection methods (back face culling [211, 107, 106], view–frustum culling [40]) can be applied before rasterizing.

time rasterizing objects that are actually invisible. The depth buffer is simple to implement in hardware hence it is commonly used in today's rendering systems. The lack of output sensitivity is not restrictive for scenes where most of the scene elements are visible, such as a single albeit complex object. For large scenes with *high depth–complexity* the processing of invisible objects causes significant "overdraw" — the wasted effort.

One approach to overcome this problem solves the VFP approximately and feeds the underlying visibility algorithm (possibly implemented in hardware) only with objects that are probably visible. This technique is usually called visibility culling. It is desirable to ensure that the visibility is resolved correctly and no objects are missing in the final image. This holds for *conservative algorithms*, that identify a *superset* of visible objects.

A lot of research has been devoted to the concept of potentially visible sets (PVS) in environments that can be decomposed to cells that are connected with transparent portals [2, 193]. In our classification these approaches belong to the VFR class of visibility problems that will be discussed in Chapter 5. An exception is the algorithm of Luebke and Georges that computes the PVS for a given viewpoint [113].

Nowadays, the BSP trees are commonly used to organise polygonal scenes. The BSP based visibility algorithm has been improved by Gordon and Chen to achieve output–sensitivity of the algorithm for certain type of environment [74]. An interesting output–sensitive hidden surface removal algorithm for scenes described solely as a BSP tree was introduced by Naylor [126]. BSP trees have also been used to represent a collection arbitrary polyhedra and to perform set operations on the polyhedra [194, 122, 124, 125, 123]. Dynamic changes to BSP trees were studied in [197, 38]. Recently, Naylor introduced an algorithm for converting discrete images to 2D BSP trees [177]. Other work on BSP trees can be found in [119, 68, 1, 93, 116, 22, 51, 121, 127, 196].

The *hierarchical z–buffer* algorithm introduced by Greene [80] uses a discrete *z–pyramid* to represent the *occlusion map*. It exploits spatial coherence by processing the object (spatial) hierarchy through the z–pyramid. Although it is a very promising approach if using hardware resources, the simulation of the z–pyramid in software causes a significant overhead. The extension of graphics hardware to support occlusion queries was further studied by Bartz et al. [11].

In [79] Greene combines hierarchical visibility culling with antialiasing techniques to avoid visual artifacts for scenes with many small polygons. Another method of Greene [78], suitable for high resolution rendering, uses hierarchy of discrete coverage masks to represent the occlusion map.

An algorithm that uses discrete image space representation of the occlusion map was introduced by Zhang et al. [212]. This technique exploits hardware to create a hierarchical occlusion map using texture filtering operation. While taking advantage of hardware rendering, this method can suffer if the rendering support is insufficient. The resolution of the occlusion map must be rather high, even if only few occluders are used to build the map.

The work presented in this chapter is closely related to object space visibility culling algorithms presented by Hudson et.al [92] and Coorg [45, 46, 47]. In [92] a *shadow frustum* is constructed for each selected occluder. These frusta are used to classify visibility of the regions of the spatial hierarchy. Since a region is tested against each frustum independently, the occlusion caused by multiple unconnected occluders is not discovered.

VFP problems also include determination of shadows in scenes with point light sources [208, 161, 32]. Parts of the scene that are invisible from the light source are in shadow, whereas the visible ones are lit. Another VFP problem – visibility map construction was studied recently by Stewart and Karkanis [173]. The VFP problem must also be solved within most systems for real time rendering [149, 141, 14, 204, 52, 28]. Recently, image based rendering have become a popular method that accelerates rendering of complex environments [151, 147, 4, 146]. These methods solve visibility to create an image based representation of certain scene objects or portions of the scene. The image is reused until it is no longer a satisfying representation. A method combining image based rendering and ray casting was presented by Wimmer et al. in [205].

An interesting algorithm for global illumination was introduced by Szirmay–Kalos [180, 182]. It uses visibility determination for set of parallel rays, what corresponds to visibility determination from a point in infinity. Another application of VFP in global illumination techniques was studied by Sillion and Drettakis [155, 56].

A lot of research has been devoted to visibility from point in $2D$ and $2\frac{1}{2}D$ dimensional environments such as terrains [128, 170, 43]. A recent algorithm for visibility determination in urban environments was described by Wonka and Schmalstieg [206]. These problems can be usually thought of as involving one dimensional line–space and so they are simpler to solve than the general VFP problems.

Further in this chapter we present algorithms that solve the VFP problem conservatively. The proposed algorithms can be used to assist the traditional techniques (depth–buffer, BSP–tree) to achieve the desired output–sensitive behaviour. The proposed methods use *occlusion trees* and a kD–tree based spatial hierarchy to quickly identify invisible portion of the scene. The spatial hierarchy is used to organise objects of the scene. For a current viewpoint a set of polygonal *occluders* is determined that are used to build the occlusion tree. The occlusion tree merges *occlusion volumes* (set of lines blocked behind the occluder) of the selected occluders. Visibility from the viewpoint is determined by processing the spatial hierarchy and classifying the visibility of its regions. In this process the occlusion tree is used to determine the viewpoint–to–region visibility efficiently.

4.2 Hierarchical Visibility Culling using Occlusion Trees

Assume we are able to classify visibility of a polyhedral region from the viewpoint as: *visible, partially visible*, or *invisible*. We can apply the visibility classification to all bounding volumes of objects in the scene. The superset to be considered for exact visibility would consist solely of objects classified as visible or partially visible.

We can make use of the spatial coherence by exploiting a spatial hierarchy that groups close objects together in a recursive fashion. Each node of the hierarchy corresponds to certain spatial region. Starting from the root of the hierarchy, visibility of each node can be determined as follows: If a node is found visible, all its descendants are visible. Similarly, if a node is found invisible, all its descendants are invisible. Descendants of nodes classified as partially visible must be further tested to refine their visibility. When the visibility of all leaves is known, objects from fully visible and partially visible leaves are gathered. These form the desired superset of visible objects and can be passed to an underlying visibility algorithm such as depth–buffer.

In many scenes most its parts are occluded by a few large objects near the viewpoint. Assume that the scene objects are decomposed into convex polygons that we call *occluders*. We identify several occluders for the current viewpoint and for each the occlusion volume (frustum) is determined. The frustum is an intersection of $(e+1)$ half–spaces, where e is number of edges of the polygon. The half–spaces are formed by planes passing through the viewpoint and edges of the polygon and the supporting plane of the polygon (see Figure 5). The occlusion frusta are merged into the *occlusion tree*, that is a variant of the *shadow volume BSP* tree introduced by Chin and Feiner [32].

Further we show that the visibility of a closed polyhedral region can be determined by combining visibility classifications of its faces. The visibility of a polygonal face is determined efficiently using the occlusion tree. We also introduce a *modified occlusion tree*, that is used to establish visibility of a region without testing the visibility of its boundary faces. The only

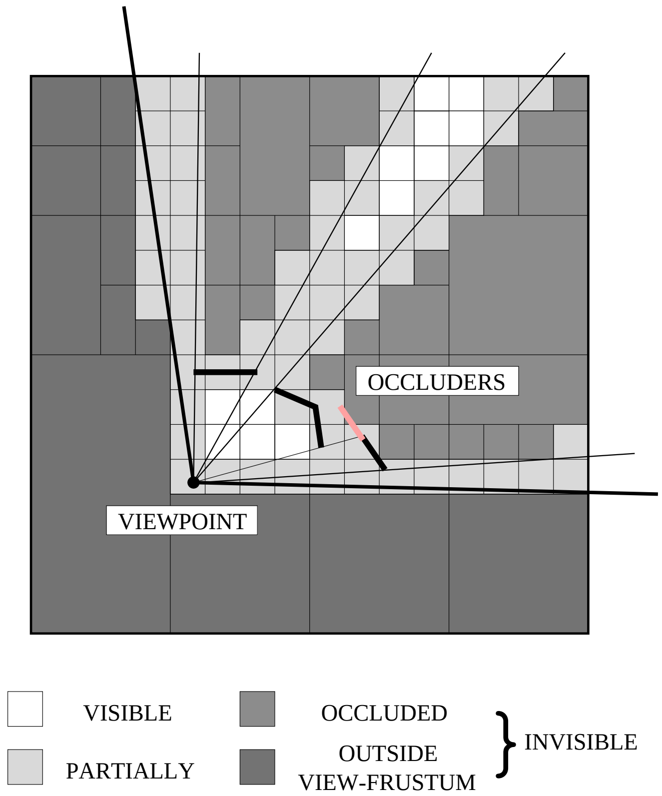


Figure 4: An example of the hierarchical visibility culling. The visibility test uses merged occlusion volumes of four occluders.

operation involved in the visibility test is the determination of the position of the region with respect to a plane. Although this method is only conservative (can identify an invisible region as partially visible), we observed its superior performance in practice.

The rest of the chapter is organised as follows: Section 4.2.1 focuses on the spatial hierarchy. The occluder selection is outlined in Section 4.2.2. In Section 4.2.3 we discuss the motivation for building a unique data structure that represents the occlusion map. In Section 4.2.4 the occlusion tree and algorithms of its construction and traversal are presented. The modified occlusion tree is introduced in Section 4.2.5. In Section 4.3 some general improvements that exploit spatial and temporal coherence in the scope of the hierarchical visibility algorithms are presented. In Section 4.4 the results are evaluated. Finally, Section 4.5 concludes.

4.2.1 Spatial Hierarchy

In our work we have used a kD–tree [100], because of its high flexibility and simplicity of building and traversal. The most important step during the building of the kD–tree is the choice of the partitioning plane. In the previous chapter we have already discussed the construction of the kD–tree for ray casting purposes. Recall the elementary steps of the tree construction: The partitioning plane is selected that subdivides the current region and two descendants of the current node are created. Objects are distributed into the descendants according to their position to the partitioning plane. Initially the root node of the BSP tree corresponds to the bounding box of the scene. Applying the algorithm recursively, the whole kD–tree is built. The recursion is terminated when the number of objects in the current node falls under the specified threshold or the specified maximum depth of the hierarchy is reached.

In certain cases an object lies on both sides of the plane (i.e. in both positive and negative

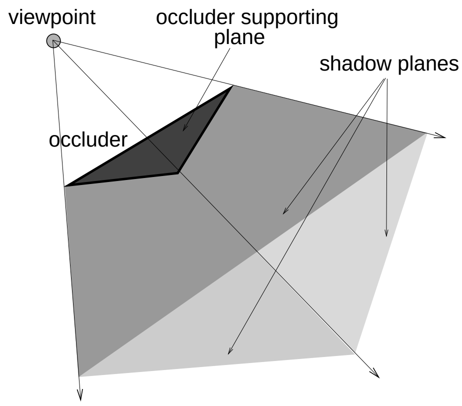


Figure 5: Occlusion volume of a polygon. The occlusion volume is formed by three shadow planes and the supporting plane of the polygon.

half–spaces induced by the plane). Such objects are "duplicated" in both new nodes. For visibility culling purposes we want the object duplications in leaf nodes of the tree to be minimised while keeping a well–balanced tree. To achieve this goal we have used the following heuristics:

For the current node we identify a principal axis x with the largest extent of the corresponding cell C. Only planes perpendicular to the axis x are considered. Further, the boundaries of object bounding boxes located within a certain distance from the spatial median of C are identified. Each identified boundary induces one *boundary plane.* We evaluate the number of objects that are split by each boundary plane. The boundary plane with the lowest number of splits is selected to partition the cell C.

It worth mentioning that the binary tree structure can be used easily to simulate trees with other branching factor (such as "irregular" octrees).

4.2.2 Dynamic Occluder Selection

The goal of the dynamic occluder selection is to determine a specified number of polygonal occluders, given a viewpoint and a viewing direction. The algorithm uses the *area–angle* heuristics [45] to estimate the quality of an occluder. The area–angle is expressed as:

$$M = \frac{-A(\vec{N} \cdot \vec{V})}{\|\vec{D}\|^2}, \tag{1}$$

where A is the area of the occluder, $\vec{N}$ denotes the occluder normal, $\vec{V}$ the viewing direction, and $\vec{D}$ corresponds to the vector from the viewpoint to the centre of the occluder ($\|\vec{N}\| = \|\vec{V}\| = 1$). The dynamic occluder selection is performed after each change of the viewpoint or the viewing direction and the set of occluders obtained in the current frame is used to perform visibility culling in the next frame.

The algorithm proceeds as follows: We identify all visible and partially visible leaves of the hierarchy, that correspond to regions located within a certain distance λ from the viewpoint. For each potential occluder referred in these leaves the area–angle is computed. These values are used to select k occluders with the largest area–angle, that form the desired occluder set for the next frame.

The λ distance has an impact on the time spent by the occluder selection. In our implementation it is a user specified constant defined as a multiple of the observer's step size. The number of desired occluders (k) influences the size of the occlusion tree and in turn the time of visibility determination and its efficiency. In our experiments we have used k between 6 and 256 (details follow in Section 4.4).

4.2.3 Representation of Occlusion Map

The crucial part of the visibility culling algorithm is the representation of the occlusion map. Unlike previous continuous methods used in the field of visibility culling we build an auxiliary object space data structure to represent the occlusion map. This is desirable, since treating occlusion volumes separately can identify an invisible region as partially visible in many cases [92].

The proposed data structure efficiently merges occlusion volumes of the selected occluders. This allows us to discover occlusion caused by multiple connected occluders and even occluders completely disjunct in space, that overlap after the projection. It is obvious that merging the occlusion volumes requires an additional time to build the appropriate data structure. We have observed that by exploiting the spatial coherence of occluders, the additional time is recovered during the visibility queries.

As a basis for our research we have adopted the concept of *shadow volume BSP trees* (SVBSP) introduced by Chin and Feiner [32]. In their work the SVBSP was used for fast generation of shadows cast by polygonal objects in scenes with point light sources. Note, that casting shadows from point–light sources represents another problem from the VFP class.

Next we present a brief overview of the SVBSP data structure and the algorithm suitable for the generation of shadows. Further, we discuss how to adapt the concept of SVBSP for the viewpoint–to–region visibility determination.

Generation of Shadows with SVBSP

The SVBSP is a variant of the BSP tree for representing polyhedra introduced by Naylor and Thibault [194, 124]. It represents a union of shadow volumes cast by convex polygons (occluders) facing a point light source. Each internal node of the tree is associated with a *shadow plane* passing through the light source and an edge of the occluder.

The direction of the shadow plane normal is used to determine the half–space in which the occluder and its shadow are located. The normals are oriented so that the shadow volume and the occluder itself lie in the negative half–space (back side) of the plane. Each leaf node of the tree corresponds to a semi–infinite polyhedral cell (frustum). The leaves are classified as *in* or *out*. A leaf is marked as in if the corresponding cell lies in shadow. Similarly, an out–leaf indicates that the corresponding cell is lit by the light source. The total shadow volume is a union of all cells corresponding to in–leaves. An example of the SVBSP tree is depicted in Figure 6.

Assume that the front–to–back ordering of polygons is determined. Obviously, farther polygons cannot cast shadows on polygons lying closer to the light source. The SVBSP is constructed incrementally by processing the polygons in the given order. The contribution of one polygon to the SVBSP is determined as follows:

1. Lit fragments of the polygon are determined.

2. The SVBSP is enlarged by shadow volumes cast by the lit fragments.

The lit fragments of the polygon are determined by *filtering* the polygon down the SVBSP. The filtering is essentially a constrained depth–first–traversal of the SVBSP, that starts from the root. For the current node the position of the polygon with respect to node's shadow plane is determined. If the polygon is located completely on the back or front side of the

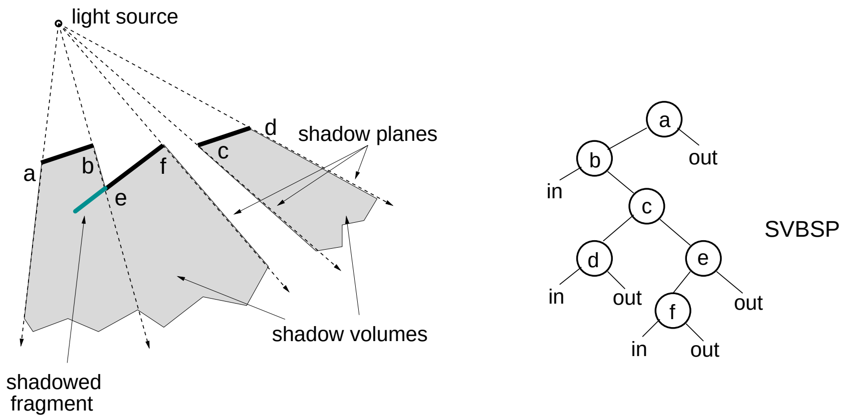


Figure 6: A 2D example of three polygons facing a point light source and the corresponding SVBSP tree.

shadow plane, it is filtered down the back or front child of the node, respectively. Otherwise, the polygon is split by the shadow plane into two fragments and these are filtered down the relevant children.

Reaching leaf nodes a set of convex polygonal fragments is obtained. These are either lit (out–leaves) or shadowed (in–leaves). In the appropriate out–leaves the tree is enlarged by the shadow volumes cast by the lit fragments. For each lit fragment e new nodes are used to replace the corresponding leaf (e is the number of edges of the fragment). The algorithm merging the shadow of a polygon into the SVBSP is given in Figure 7.

```
Algorithm FilterDown(Node, Polygon, Viewpoint)
begin
   if Node is leaf then
      if Node is out-leaf then
        replace Node by
        OcclusionVolume(Polygon, Viewpoint)
      else
        do nothing
   else
      case Split(Polygon, Node.Plane, Back, Front) of
        FRONT : (* pass the polygon to the front subtree *)
              FilterDown(Node.FrontChild, Polygon, Viewpoint);
        BACK  : (* pass the polygon to the back subtree *)
              FilterDown(Node.BackChild, Polygon, Viewpoint);
        SPLIT : (* pass fragments to apropriate subtrees *)
              FilterDown(Node.FrontChild, Front, Viewpoint)
              FilterDown(Node.BackChild, Back, Viewpoint);
      end
end
```

Figure 7: Pseudo–code of the algorithm processing a polygon through the SVBSP tree.

For shadow generation purposes, the lit and shadowed fragments can be stored within the original polygon. During rendering the colour of these fragments can be set accordingly.

The front–to–back ordering of polygons can be achieved by building a BSP tree and its appropriate traversal [63]. Alternatively, hybrid approaches combining graph based algorithms and BSP trees can be used [31, 94, 117, 164].

Visibility Determination and the SVBSP

The SVBSP tree is a hierarchical data structure allowing fast incremental determination of lit and shadowed fragments of scene polygons. We will refer to the algorithm described previously as the *SVBSP algorithm*. All lit fragments obtained by the SVBSP algorithm are visible from the light source. Similarly, all fragments located in shadow are invisible from the light source. From now on assume the position of the point light source to be a viewpoint. Instead of lit and shadowed, the terms visible and invisible (occluded) will be used in the following text.

Assume the SVBSP is built with respect to the selected set of occluders and the current viewpoint. Given a polyhedral region (cell) we want to quickly determine if this region is visible, partially visible or invisible. We will use the term *visibility algorithm*, referring to the traversal of the SVBSP, which appropriately classifies the visibility of a region.

The differences between the desired visibility algorithm and the SVBSP algorithm can be summarised as follows:

- Only selected occluders are used to build the tree, and in the scope of the visibility algorithm the tree is not modified any more. The SVBSP algorithm assumes all polygons to be processed and the tree to be always updated accordingly.

- The subject of the visibility algorithm is a polyhedral region, whereas in the SVBSP algorithm it is a convex polygon.

- The region of which the visibility is determined may lie in front of some occluders. Hence the front–to–back ordering is not satisfied in the visibility algorithm.

- The visibility algorithm is decisive whereas the SVBSP is a constructive one. The result of the visibility algorithm is only one of the three visibility states. In the original SVBSP algorithm the goal is to obtain lit and shadowed fragments of a polygon.

With these differences in mind, we have designed the occlusion tree, modified occlusion tree, and appropriate visibility algorithms (i.e. algorithms of their traversal).

4.2.4 Occlusion Tree

An *occlusion tree* (OT) is a BSP tree built with respect to a set of occluders and a viewpoint. Before the construction of the OT the occluders are used to build an ordinary BSP–tree to establish their front–to–back ordering. Then the OT is constructed as follows: The occluders are processed in front–to–back order and the corresponding occlusion frusta are used to enlarge the OT. The construction proceeds as the construction of SVBSP. Additionally, in each in–leaf a link is stored to the fragment occluding the frustum, that corresponds to this leaf. These links are used in the visibility algorithm to determine if a polyhedron tested for visibility lies behind the occluder.

Visibility of a closed polyhedral region is determined by combining visibility classifications of its faces, that are convex polygons. In the next section we describe how to classify visibility of a convex polygon using the OT. Further we present the visibility algorithm for a convex polyhedron. Both these algorithms classify the visibility *exactly* with respect to the occluders the OT was constructed for. Recall that if these occluders form only a subset of all objects in the scene, the hierarchical visibility algorithm is only conservative.

Visibility of Polygon

Given a polygon the OT is traversed by constrained depth first traversal (DFT). In each interior node of the OT the position of the polygon with respect to the plane referred in the node is determined. If the polygon lies completely in front or back of the plane, the visibility algorithm is applied on the appropriate child of the current node. Otherwise the polygon is split in two fragments and the algorithm is applied on both children using the relevant fragments.

In each node reached by the DFT the visibility of the corresponding fragment of the polygon is computed by combining visibility states of node's children according to a simple rule shown in Table 3.

When a leaf is reached, the visibility of the current fragment of the polygon is classified. Reaching an out–leaf the fragment is fully visible. The visibility in an in–leaf can evaluate to any of the three possible states, hence, an additional test must be applied. This test will be explained later in this section.

Fragment A	Fragment B	A $\cup$ B
F	F	F
I	I	I
P	X	P
X	P	P
I	F	P
F	I	P

Table 3: Combining visibility states of fragments. Abbreviations: I – Invisible; P – Partially visible; F – Fully visible; X – any of the I,P,F states.

The visibility of the whole polygon corresponds to a visibility state of the root node of the OT. The DFT can be terminated whenever a fragment is found partially visible. This follows from the fact that if a fragment of the polygon is partially visible, the polygon itself must be partially visible (see Table 3). This constraint can significantly accelerate the visibility algorithm, particularly for large polygons, which are likely to be partially visible.

Note, that the polygon visibility algorithm closely resembles the filtering used in the tree construction. Nevertheless the actions performed reaching leaf nodes differ significantly.

As already mentioned, the polygon tested for visibility need not lie behind all occluders. Therefore, reaching an in–leaf node an additional test must be applied. We use the link to the occluder–fragment occluding the frustum corresponding to the leaf. The supporting plane of the occluder–fragment is used to establish visibility of corresponding fragment as follows: If the fragment is completely in front of the plane, it is fully visible. If it is completely on the back side of the plane, it is invisible. Otherwise, it lies on both sides of the plane and it is partially visible. The visibility algorithm for a polygon with respect to the occlusion tree is summarised in Figure 8.

Visibility of Polyhedron

As already mentioned the visibility state of a polyhedron can be determined by combining the visibility of its faces. Assuming these faces are convex polygons, the above presented polygon visibility algorithm can be applied. The visibility of the polyhedron is refined incrementally, processing its faces one by one. We call the *current visibility* of the polyhedron the visibility of the union of those polyhedron faces which have already been processed. Visibility of a

```
Algorithm Visibility(Node, Polygon)
begin
    if Node is leaf then
        if Node is out-leaf then
            Visibility ← VISIBLE
        else
            Visibility ← visibility state based on
            FragmentIntersection(Node.Fragment, Polygon);
    else
        case Split(Polygon, Node.Splitter, Back, Front) of
          FRONT : (* pass the polygon to the front subtree *)
                Visibility ← Visibility(Node.FrontChild, Polygon);
          BACK  : (* pass the polygon to the back subtree *)
                Visibility ← Visibility(Node.BackChild, Polygon);
          SPLIT : (* pass fragments to apropriate subtrees *)
                Visibility ← Visibility(Node.FrontChild, Front)
                if Visibility <> PARTIALLY then
                begin
                    aux ← Visibility(Node.BackChild, Back)
                    Visibility ← CombineVisibility(aux, Visibility);
                end
        end
end
```

Figure 8: Pseudo–code of the polygon visibility algorithm using the OT.

face of the polyhedron which is facing the viewpoint is computed using the polygon visibility algorithm. If it is not the first face processed, the current polyhedron visibility is combined with the visibility of the face recently processed using the rule shown in Table 3. Whenever the current polyhedron visibility reaches partially visibility, the algorithm can be terminated. Otherwise, it proceeds with a next face until all faces have been processed.

In the case of kD–tree based spatial hierarchy, the cells corresponding to its nodes are parallelepipeds. To determine visibility of such a cell at most three rectangular polygons must be tested for visibility (these polygons can be determined by a table lookup).

4.2.5 Modified Occlusion Tree

The OT data structure is used extensively in the scope of the hierarchical visibility algorithm. The elementary operation, which takes place in both the tree construction and the visibility algorithms, is the polygon splitting. Recall that the splitting operation determines fragments of the polygon lying in the negative and positive half–spaces induced by a plane.

Although the splitting can be implemented rather efficiently, there is still an overhead of fragment allocation when the polygon is split by the plane. The splitting operation was also the reason why the polyhedra visibility could not be determined by processing the polyhedron itself. This follows from the complexity of the splitting operation for polyhedra as well as maintaining polyhedra fragments. Therefore, the decomposition of polyhedra and the combination of visibility states of their faces have been used.

Motivated by the idea of the visibility algorithm (tree traversal) without the necessity of the polygon splitting, we have designed the concept of the modified occlusion tree (MOT). It is based on observation that carefully removing some nodes (shadow planes) of the OT, the tree still contains a complete information about the occlusion volume. Additionally such a data structure can be traversed more efficiently using the *modified visibility algorithm.* The algorithm is generally conservative with respect to the occluders used to built the MOT. Despite this fact, we have observed its superior performance in practice.

Construction of the MOT

The MOT is constructed similarly as the OT. Assume an occlusion frustum of a polygon is being merged to the MOT. The polygon splitting operation used during filtering is enriched by *flagging*, that marks edges of the polygon embedded to any shadow plane. If new fragments are created by the splitting operation, the edges of both new fragments lying in the shadow plane that splits the polygon are flagged as well.

When an out–leaf is reached during the polygon filtering, only unflagged polygon edges are used to create shadow planes enlarging the MOT. Shadow planes that would have been created by flagged edges must be already present in the MOT (otherwise the edges would not be flagged). If an out–leaf is reached and all edges of the filtered fragment are flagged, no internal nodes are added to the tree. Instead, the out–leaf is replaced by an in–leaf and the link to the fragment in the new in–leaf is set. The difference between an occlusion tree and its modified version is illustrated in Figure 9.

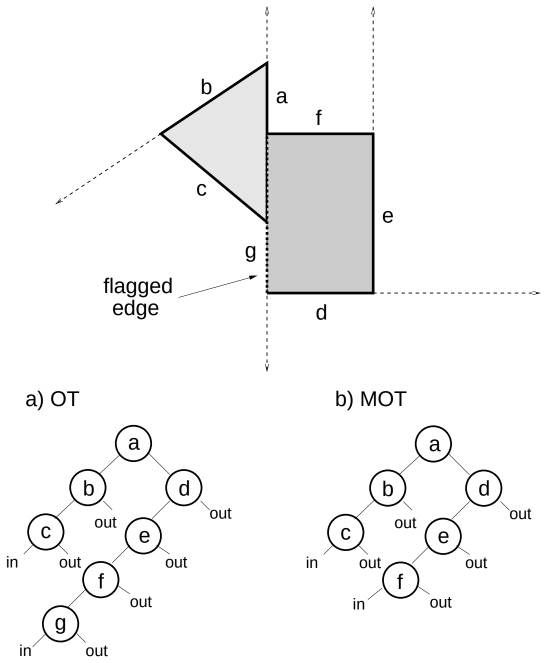


Figure 9: The difference between the OT and the MOT. Both trees are constructed with respect to the same occluders. The occluders are shown as seen from the viewpoint. Node g is not present in the MOT, since the corresponding occluder–edge was flagged.

It is obvious that the MOT contains at most the number of nodes than the OT built using the same set of occluders.

Let us focus on the structure of the MOT. Assume an out–leaf is reached during filtering and the corresponding fragment contains flagged edges. Some shadow planes are not added to the tree because the corresponding edges were flagged. Ignoring these shadow planes the occlusion volume of a fragment is enlarged. This larger occlusion volume is merged to the tree by replacing certain out–leaf. Obviously, each node of the tree implies a unique path from the root. During the polygon visibility algorithm, the filtered polygon is clipped by all the planes on the path to the node. In the subtree of the node it is not necessary to clip the position of the filtered fragment to any plane which already occurred on the path. The flagged edges would generate exactly such planes. Hence, ignoring these planes does not effect the correctness of the visibility algorithm.

In the next section we present a modified visibility algorithm for the MOT. In the previously presented algorithms, the polygon visibility algorithm has been used to determine the visibility of a polyhedron. The modified algorithm determines the visibility of regions of various shapes "directly", i.e. without the face decomposition. The only operation involved is the classification of the position of the region with respect to a plane.

Conservative Visibility of a Region

Assume that the position of a region with respect to a plane can be determined. It indicates if the region lies in negative (back), positive (front), or both half–spaces. Using this operation, the MOT is traversed similarly to the OT polygon visibility algorithm. In each internal node of the tree we determine the position of the region with respect to the shadow plane and apply the algorithm recursively on appropriate subtrees. No attempt to split the region is made even if it lies on both sides of the plane.

Reaching an in–leaf L, the region is tested for position to the supporting plane of the occluder–fragment F referred in the leaf. As mentioned previously, if the region lies on both sides (half–spaces) of the supporting plane, it would be classified partially visible. Since the region splitting was not applied it is possible that the part of the region crossing the supporting plane is not contained in the frustum corresponding to the fragment F. In this case it can be occluded by another occluder–fragment than F.

Such a situation can be discovered by testing the region for an intersection with the fragment F itself. If they do not intersect, the region is classified invisible within the frustum corresponding to L. Note that the region can still be found partially visible when the visibility of all leaves reached during the visibility algorithm is combined. In the case of box–shaped regions, a fast algorithm can be used for the box/polygon intersection [77]. Nevertheless, we have observed that the possible improvement in correctness of the algorithm is usually not worth the time spent by the additional intersection test (see Section 4.4).

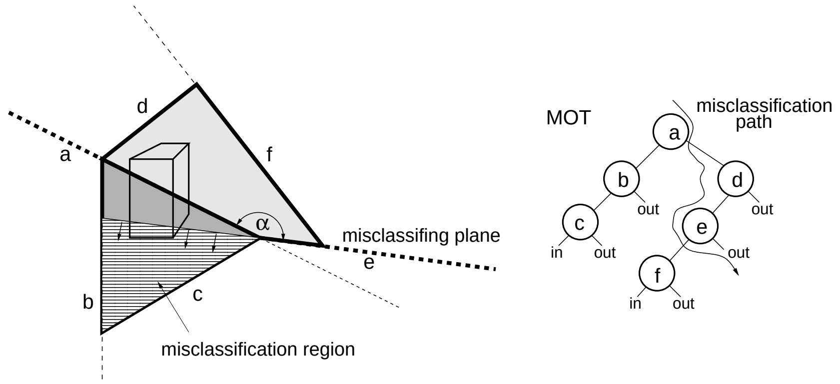


Figure 10: An example of a disadvantageous configuration of occluders. The shadow planes of the MOT are shown by thin lines. The invisible polyhedron intersecting both the shadow plane a and the marked region is classified as partially visible.

The modified visibility algorithm is only conservative. This implies that the results (i.e. visibility classification of leaves) can vary comparing to the previously mentioned exact algorithms. The situation when an invisible region is misclassified as partially visible is depicted in Figure 10. It can be seen that the algorithm is likely to give an imprecise (conservative) result if the angle α between the planes a and e gets larger. In general, it is difficult to give a probabilistic analysis of the expected number of cases when the algorithm fails to give an exact result. Nevertheless, we have observed that the algorithm gives conservative results only in a very few cases in practice.

It is worth mentioning that the algorithm can be modified to always give a correct answer. This can be accomplished using region/wedge intersection tests in the interior nodes of the MOT. Nevertheless, such an algorithm would loose the desired features of the modified visibility algorithm — the simplicity of traversal, generality, and speed. Therefore we have decided not to present the modification in detail.

4.3 Exploiting Spatial and Temporal Coherence

In this section we present a series of simple improvements that make use of temporal and spatial coherence in the scope of hierarchical visibility algorithms. The *node skipping* avoids visibility tests of certain interior nodes of the hierarchy. The *visibility propagation* algorithm reuses information about visibility of neighbouring spatial regions. Finally, the *conservative node skipping* avoids visibility tests of the hierarchy nodes that are expected to remain visible. The presented methods were evaluated in the context of the hierarchical visibility culling using *occlusion trees*.

4.3.1 Overview

Exploiting various types of coherence is one of the main goals of today's computer graphics. In the scope of visibility algorithms at least three types of coherence can be exploited: object space, image space, and temporal. Hierarchical visibility algorithms exploit the spatial coherence inherently by utilising a spatial hierarchy. For densely occluded scenes they may achieve a great benefit by quickly identifying groups of invisible objects, that need not be considered for rendering.

A typical hierarchical visibility algorithm uses a *visibility test*, that classifies a node of the spatial hierarchy as completely visible, partially visible or invisible depending on the visibility of the spatial region corresponding to that node (see previous sections). Here we do not focus on the amount of image space or temporal coherence, that may be exploited by the visibility test itself. Instead we suggest a more general framework, that is rather independent of the particular visibility algorithm.

Traditional hierarchical visibility algorithms traverse the spatial hierarchy starting at the root node. Firstly, we propose the *node skipping*, that saves up to half* of the visibility tests by skipping certain interior nodes of the hierarchy. Using the *node skipping* the resulting visibility classification does not differ from the one obtained by the classical traversal. Secondly, we describe an algorithm, that reuses visibility classifications of nodes already processed in the current pass. The nodes are processed in front–to–back order and the algorithm tries to determine visibility of the region corresponding to the current node by combining visibility states of neighbouring regions. If it fails, the usual visibility test is applied. Finally, we propose a conservative method, that aims to avoid repeated visibility tests of nodes that probably remain visible.

*This holds for a binary hierarchy.

4.3.2 Related Work

Some visibility algorithms exploit temporal coherence in a specialised way. Greene et al. [80] uses the set of visible objects from one frame to initialise the *z–pyramid* in the next frame and so reduces "overdraw" of the *hierarchical z–buffer*. Coorg and Teller [46] present a visibility algorithm that uses *relevant planes* which form a subset of visual events. They restrict the hierarchy traversal to nodes corresponding to planes that were crossed between successive viewpoint positions. Another approach [47] exploits temporal coherence by caching occlusion relationships.

Chrysanthou and Slater have proposed a probabilistic scheme for view–frustum culling [162]. They partition objects into groups, which are sampled according to their distance from the view–frustum. It is difficult to generalise this method for visibility algorithms, since the "visible volumes" can be very complex, and usually they are not explicitly reconstructed. Moreover, this method is not conservative unless changes in viewing direction and position of the viewpoint are restricted.

The methods presented here can be used to make use of temporal coherence in the scope of existing visibility algorithms, that utilise a spatial hierarchy. Examples of these are algorithms based on hierarchical occlusion maps [212], coverage masks [78], shadow frusta [92], and occlusion trees [15].

4.3.3 Classical Approach

An elementary step of the hierarchical visibility culling is the *node visibility test*, i.e., visibility classification of a single node of the hierarchy using certain occlusion map. We assume that given a viewpoint and a viewing direction the visibility algorithm classifies visibility of the node as *completely visible*, *partially visible* or *invisible*. We have used the polyhedra visibility algorithm described in the previous sections.

The classical hierarchical visibility culling proceeds as follows: Starting from the root node of the hierarchy, the view–frustum culling is applied on the current node. If the node is outside the view–frustum it is classified invisible. Otherwise, the node visibility test is performed. If the node is found visible all its descendants are visible. Similarly, if a node is invisible, all its children are invisible. Descendants of nodes classified as partially visible are tested further to refine their visibility (see Figure 4). When the visibility of all leaves is known, objects from fully visible and partially visible leaves can be gathered and rendered using a low–level exact visibility solver (such as depth–buffer).

A simple improvement can be used to avoid visibility tests of hierarchy nodes that contain only few objects and so the associated cost of rendering is lower than the cost of the visibility determination. In such a case the node can be simply classified as visible.

4.3.4 Modifications Overview

In order to give an overview of the proposed modifications we first show how they are exploited in the scope of the hierarchical visibility algorithm (see Figure 11). The *node skipping* test is applied first. This test eventually decides to skip all the remaining steps and to continue determining visibility of descendants of the current node. The *view–frustum* culling can report the node as invisible if it is outside the view–frustum. Otherwise the *visibility propagation* is applied, that can succeed classifying the node as visible or invisible. The *conservative node skipping* classifies some nodes as visible with certain probability. If all previous steps failed in determining node's visibility the *visibility test* is applied. Note, that the steps are applied in order of increasing computational cost, what reflects the main idea of culling: use more complicated test only when the simple one fails to find solution.

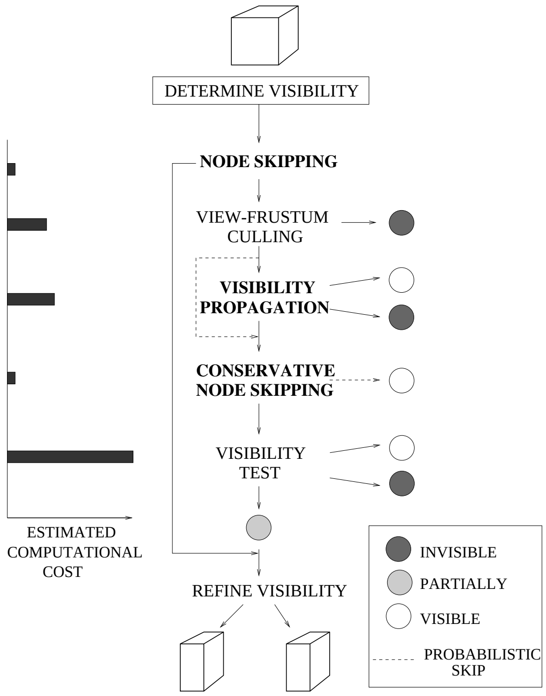


Figure 11: Series of steps determining visibility of a node of the hierarchy. The novel methods are highlighted.

The rest of this section is organised as follows: In Section 4.3.5 the node skipping method is introduced. The visibility propagation is presented in Section 4.3.6. In Section 4.3.7 a conservative modification of the node skipping algorithm is outlined.

4.3.5 Node Skipping

The hierarchical visibility algorithm can be viewed as a traversal of the hierarchy, that is terminated either at leaves or nodes classified either as visible or invisible. Let us call such nodes the *termination nodes* and nodes that have been classified partially visible the *opened nodes*. Denote sets of termination and opened nodes in the i-th frame $\mathcal{T}_i$ and $\mathcal{O}_i$, respectively. For the classical approach $\mathcal{T}_i \cup \mathcal{O}_i = \mathcal{V}_i$, where $\mathcal{V}_i$ is the set of all nodes visited in the i-th frame.

Imagine the viewpoint is fixed. The visibility of all nodes of the hierarchy does not change and the sets $\mathcal{T}_i$, $\mathcal{O}_i$, and $\mathcal{V}_i$ are fixed as well. Nevertheless, the classical algorithm repeatedly tests visibility of all nodes $\mathcal{V}_i$.

The node skipping is a simple modification as it aims to eliminate the repeated visibility tests of the set of opened nodes from the previous frame. It skips all nodes of $\mathcal{O}_{i-1}$ and applies visibility tests only on nodes of $\mathcal{T}_{i-1}$. To propagate eventual changes in visibility up into the hierarchy the visibility states determined at the termination nodes are pulled up according to a simple rule: The visibility state of the node is updated as visible or invisible if all its children have been classified as visible or invisible, respectively. Otherwise it remains partially visible and thus opened. The pseudo–code of the hierarchical visibility algorithm with node skipping is outlined in Figure 12.

Consequently, the modification does not change the final visibility classification, that is the same as the one obtained using the classical approach. The behaviour of the modified hierarchical visibility algorithm is illustrated in Figure 13. Note, that if the pull up would not take place the algorithm could end up with the termination nodes being all leaves of the hierarchy. Hence, it would loose all advantages of the hierarchical algorithm.

```
Algorithm HierarchicalVisibility( NODE )
begin
   if NODE is leaf or NODE.visibility ≠ PARTIALLY
      (* termination nodes *)
      or NODE.frame < frame-1 then
   begin
      NODE.visibility ← TestVisibility( NODE );
      NODE.frame ← frame;
   end
   case NODE.visibility of
     VISIBLE   : Render subtree of NODE;
     PARTIALLY :
       if NODE is leaf then Render NODE;
       else
         for all children C of NODE do
            HierarchicalVisibility( C );
       (* pull-up *)
       if visibilty of all children equals V then
          begin
          NODE.visibility ← V;
          NODE.frame ← frame;
         end
     INVISIBLE : (* terminate the DFS *)
    end
 end
```

Figure 12: Pseudo–code of the hierarchical visibility culling with node skipping. The set of termination nodes is not explicitly maintained.

For kD–trees $|\mathcal{O}_i| = |\mathcal{T}_i| - 1$. Thus the node skipping can save almost a half of the visibility test, that would be applied on the interior nodes of the hierarchy.

4.3.6 Visibility Propagation

The hierarchical visibility culling already makes use of spatial coherence by utilising a spatial hierarchy (kD–tree). However, we can further increase the amount of coherence exploited by reusing visibility information computed for neighbouring regions.

Suppose, that the nodes of the spatial hierarchy are processed in front–to–back order with respect to the viewpoint. Using kD–tree this ordering is very simply determined [63]. We first try to determine visibility of the currently processed node by combining visibility classifications of its relevant neighbours. If the combination fails, we revert to the visibility test.

Let us denote the box corresponding to node N as B_N. The visibility of N can be determined combining visibility of potentially visible faces* $\mathcal{F}_{B_N}$ of B_N ($|\mathcal{F}_{B_N}| \leq 3$). Consequently, visibility of a face $F \in \mathcal{F}_{B_N}$ can be determined combining visibility of appropriate *neighbour nodes*. If all faces of $\mathcal{F}_{B_N}$ are invisible, the node N is invisible. Similarly, if all faces of $\mathcal{F}_{B_N}$ are visible and there is no occluder intersecting B_N, N can be classified as completely visible. Otherwise the visibility propagation fails and the usual visibility test must be applied. An example of a node that can be classified as invisible is depicted in Figure 14.

A neighbour node of N on a face F is a node U of the kD–tree with B_U lying in the opposite halfspace (induced by F) than B_N and having non–empty intersection with F. Naturally,

*These are faces that face the viewpoint. They can be determined by table lookup.

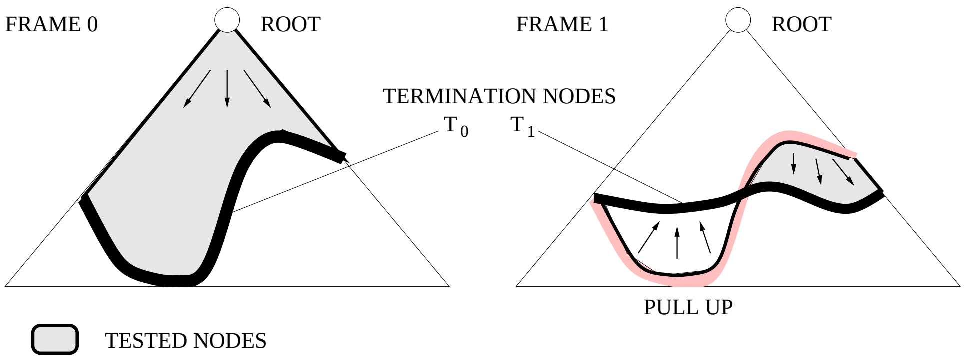


Figure 13: Illustration of the node skipping. Initially the algorithm proceeds starting at the root of the hierarchy (left). In the second frame the opened nodes $\mathcal{O}_0$ are skipped and the visibility tests are applied on the termination nodes $\mathcal{T}_0$ (and eventually "below"). Visibility changes are propagated up to the hierarchy and the new set of termination nodes $\mathcal{T}_1$ is established.

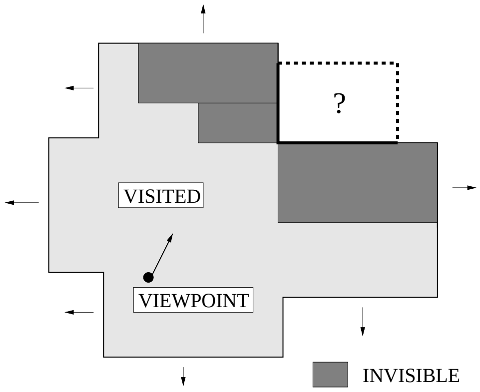


Figure 14: An example of a node that can be classified invisible, since all its appropriate neighbours are invisible.

we could keep a list of neighbour nodes for each face. Instead, we have used neighbour links (ropes) for kD–trees [88], that have low memory requirements and allow hierarchical visibility propagation.

Within each face F we associate a link to a neighbour node U that has a smallest box containing the face completely ($F \cap B_U = F$). When determining the visibility of a face there are three possible situations:

1. the link points to a node that is visible/invisible,
2. the link points to a node that is partially visible,
3. the link points to a node that has not been visited in the current frame.

The first case is trivial; the visibility of the face can be set immediately. In the second case we perform a constrained DFS and combine visibility of reached nodes. The search is constrained to nodes having non–empty intersection with the face and terminates at the termination nodes $\mathcal{T}_i$. This process is illustrated in Figure 15. The visibility combination is performed using the same rule as in the pull up pass of the node skipping (Section 4.3.5). Nevertheless, we can terminate the DFS whenever the combination results in partial visibility.

The third case is solved as follows: If the link is pointing to a node U that has not been visited in the current frame there must be some termination node on the path from U to the

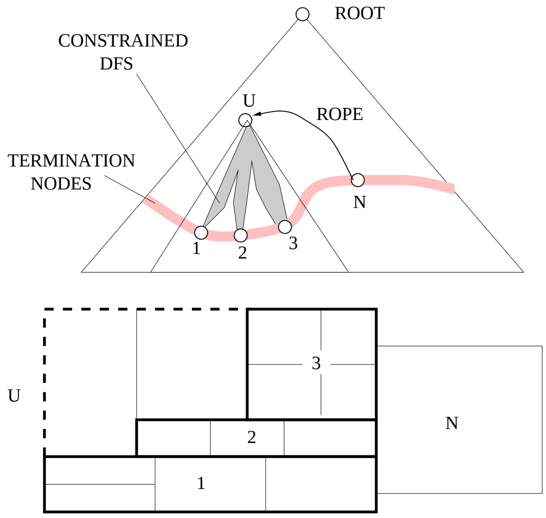


Figure 15: Illustration of the hierarchical visibility propagation using ropes.

root. This path is followed until the termination node is reached (see Figure 16). Note, that if we would propagate the visibility into subtrees of the termination nodes, the third case would never occur.

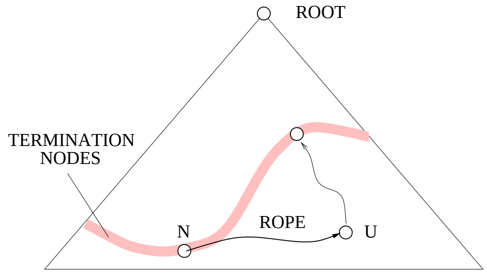


Figure 16: Lazy propagation of the visibility classification. Node U was not visited in the current frame. The algorithm must follow the path to the root to determine visibility of U.

Temporal coherence

The visibility propagation does not always succeed to determine visibility of the processed node. In such a case it introduces an additional overhead into the visibility determination. However, we can use information obtained in the previous frame to guide the algorithm in the current frame.

Firstly, we avoid visibility propagation on nodes that we expect to remain partially visible and thus the visibility propagation would probably fail. So the visibility propagation is applied only on nodes that have not been classified as partially visible in the previous frame. Secondly, if for a given node the visibility propagation succeeded in the previous frame, it is applied in the current frame as well. Otherwise, it is applied with certain probability $p_{vp} < 1$.

4.3.7 Conservative Node Skipping

The node skipping method ensures, that on each path to a leaf node of the hierarchy at least one node is tested for visibility. We can further reduce the expected number of visibility tests at the cost of the conservative behaviour of the modified algorithm. The conservative node skipping produces a superset of visible nodes determined by the previously described algorithms.

Due to the complexity of occlusion volume it is difficult to predict changes in visibility unless a specialised visibility algorithm is involved [46]. To keep the conservative behaviour of the algorithm we cannot classify a node as invisible without really determining its visibility. Nevertheless, assuming the visibility does not change significantly over successive frames visibility states of visible and partially visible nodes do not have to be updated in each frame. Instead, we skip the visibility determination of these nodes with certain specified probability p_{skip} and mark them as visible. With $1 - p_{skip}$ probability, the algorithm updates visibility of the node invoking the node visibility test.

4.4 Results

In this section we evaluate the behaviour of the visibility culling algorithm using occlusion trees. We compare the proposed algorithms with the hierarchical view–frustum culling [40, 141]. Further in this section we evaluate the asset of the methods increasing the amount of the temporal and spatial coherence used.

4.4.1 Conservative Visibility Culling with Occlusion Trees

We have used a simulated walk through several models of architectural interiors to compare efficiency of the proposed algorithms. As a reference the hierarchical view–frustum culling was used. The results are summarised in Table 4. Each line in the table corresponds to average values obtained in the scope of one walk–through.

The *time* field is an average frame time. The *overhead* field depicts an additional overhead of the visibility culling algorithms. This includes the dynamic occluder selection, building of the BSP of occluders, building the OT (MOT), and the hierarchical visibility culling. The *speedup* is a fraction of the average frame time of the pure view–frustum culling and the actual average frame time. The *rendered polygons* field contains the average number of polygons rendered in one frame. The other two parameters shown in the table are the user–specified constants influencing behaviour of the algorithm. The number of occluders used to build the OT (MOT) is shown in the *occluders* field. The *method* field identifies the type of the algorithm used. Its meaning is explained below the table. The *minimum cost* of a node to be tested for visibility was set to 50. Recall that the cost of a node expresses the number of polygons referred in the node. For each measurement 100 "detailed" objects were spread randomly in the scene. We used a virtual plant consisting of 644 polygons.

The speedup of the rendering achieved for the tested scenes varies between 1.75 and 3.75. We observed that the speedup is not linearly proportional to the number of occluders used for the visibility culling. Increasing the number of occluders by factor of two, the speedup is usually increased much less. In such a situation the overhead of visibility culling is also increased less than linearly, what documents the output–sensitivity and the favourable "logarithmic–behaviour" of the occlusion tree based algorithm. Informally, we explain such behaviour as follows: Firstly, assume the occlusion tree contains occluders that occlude large portion of the view. It is of high probability that another occluder is found invisible. In this case the occlusion tree is not enlarged. Secondly, the occlusion tree inherits the logarithmic search properties of hierarchies. Therefore, the complexity of the polyhedron visibility algorithm with the occlusion tree is expected to depend sublinearly on the number of occluders.

It follows from Table 4 that the best results were achieved by the FME method (see explanations below the table). It is the combination of the MOT and the conservative visibility algorithm with the leaf–fragment–intersection test. Under certain circumstances, we observed that the FSE method led to better results. Particularly, when a detailed object was found invisible by FSE (and not by FME), the time saved in rendering exceeded the time needed for

scene	method	occluders [—]	rendered polygons [—]	overhead [ms]	time [ms]	speedup [—]
	F	—	18192	—	276.5	1.00
	FME	16	7390	5.9	139.0	1.99
soda–5	FME	24	6537	7.9	116.0	2.38
	FME	32	5941	9.8	109.8	2.52
total # of polygons = 66085	FSE	16	6512	12.7	120.5	2.29
# of occluders = 1685	FSE	24	5569	16.0	110.5	2.50
# of kD–tree nodes = 2527	FSE	32	4725	19.1	100.1	2.76
	FMC	16	7988	5.4	135.9	2.03
	FMC	24	7362	7.3	129.3	2.14
	F	—	12587	—	182.0	1.00
	FME	16	3641	8.8	71.5	2.55
big–7	FME	24	2286	10.2	54.6	3.33
	FME	32	1818	11.6	48.5	3.75
total # of polygons = 66876	FSE	24	2214	19.9	63.4	2.87
# of occluders = 2476	FMC	16	3640	8.4	74.3	2.44
# of kD–tree nodes = 3439	FMC	24	2263	9.7	51.4	3.54
soda						
total # of polygons = 73529	F	—	15297	—	315.2	1.00
# of occluders = 9129	FME	24	4744	10.1	119.7	2.63
# of kD–tree nodes = 10475	FSE	24	4388	23.7	128.0	2.46

F – view-frustum culling
S – OT + standard visibility algorithm
M – MOT + conservative visibility algorithm
E – exact occluder-fragment/parallelepiped intersection test
C – conservative occluder-fragment/parallelepiped intersection test

Table 4: Results of the hierarchical visibility culling. The table shows the average number of polygons rendered, the average frame-time and the speedup over the view-frustum culling for different scenes and methods of the visibility culling. Measured on SGI O_2, 64MB RAM.

more complex visibility determination.

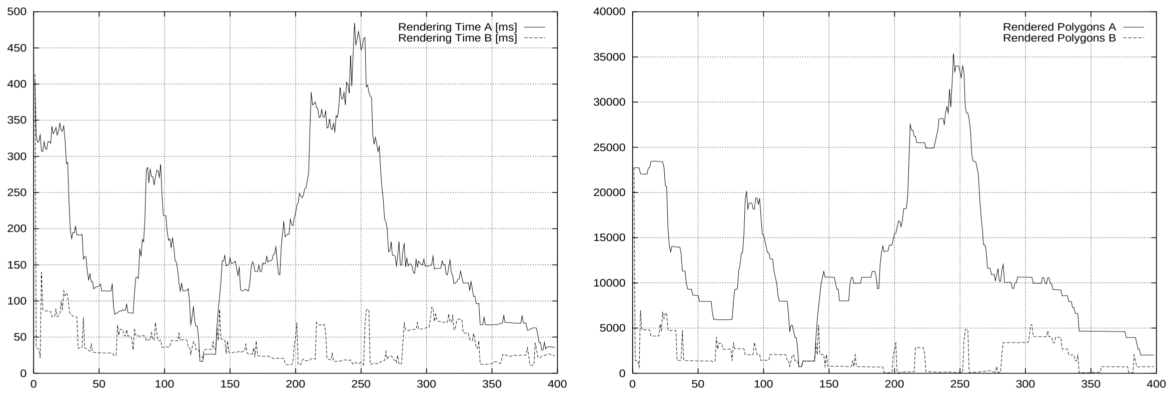
Plots of frame times and numbers of rendered polygons measured during a walk–through of an architectural model (*big-7*) are shown in Figures 17–a and 17–b.

4.4.2 Temporal and Spatial Coherence

The methods increasing the amount of temporal and spatial coherence were tested by walking through the model of the fifth floor of the Soda–Hall*. The constructed kD–tree consisted of 1187 nodes. In all measurements we used the visibility algorithm based on MOT. For each position of the viewpoint 16 occluders were identified and used to build the OT. In the scope of one walk the path depicted in Figure 21 was followed. If not stated differently all presented values are averaged per one frame of the walkthrough. The following methods were evaluated:

A — the classical approach,

*http://graphics.lcs.mit.edu/ becca/research/SodaHall/



(a) (b)

Figure 17: Evaluation of a sequence of 400 frames during walkthrough of the big-7. (a) Rendering times. Curve A corresponds to view–frustum culling only; curve B includes the hierarchical visibility culling using 32 occluders (FME). (b) The amount of rendered polygons. Curve A corresponds to view–frustum culling only; curve B includes the hierarchical visibility culling using 32 occluders (FME).

B — node skipping applied,

C — node skipping + visibility propagation with probability $p_{vp} = 0.5$,

D — as **C** + conservative node skipping with probability $p_{skip} = 0.5$.

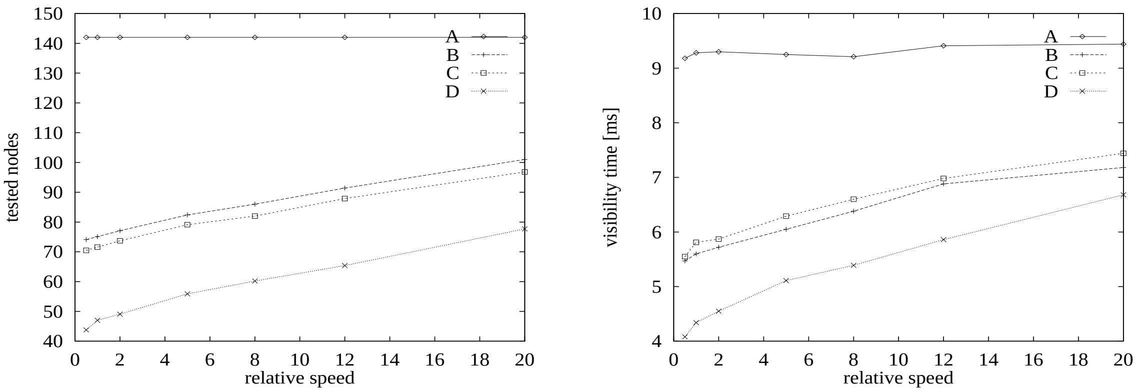
First three plots illustrate the dependence of the algorithms on the relative speed of the walk (Figures 18–a, 18–b and 19–a). Particularly the number of node visibility tests, the time spent by the hierarchical visibility determination, and the total frame time were measured.

All the new methods exhibit a very slow growth of the number of necessary visibility tests. For a walk of relative speed 1.0 following savings in number of visibility tests were achieved (compared to **A**): method **B** – 47%, method **C** – 50%, method **D** – 67%. The node skipping (method **B**) saves almost half of the visibility tests as expected. We have observed that the visibility propagation (method **C**) succeeds in determining visibility of only few nodes, that usually correspond to rather large regions. The **D** method significantly decreases the number of visibility tests. This is paid by a higher number of nodes classified as partially visible or visible (details follow further in the text).

Figure 18-b documents that the time spent by the hierarchical visibility culling was roughly proportional to the number of visibility tests. Nevertheless, we can see that the time spent by the visibility propagation (method **C**) is not recovered by the savings in number visibility tests (particularly, this follows from the fact that using the occlusion tree the visibility test is rather fast).

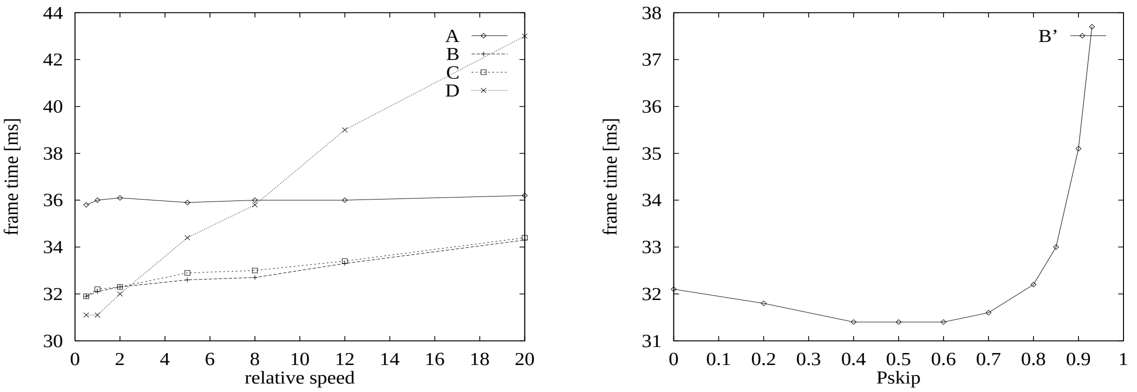
In Figure 19-a we can observe the conservative behaviour of the method **D**. When the viewpoint moves slowly the method achieves better frame times than the other ones. As the relative speed of the walk increases the visibility states of many nodes change quickly. Hence "reusing" some previously visible nodes leads to a larger set of nodes to render and the frame time is increased.

Finally, the behaviour of the conservative node skipping algorithm in dependence on the probability p_{skip} was measured (Figure 19-b). We can observe a local minimum in the average frame time at $p_{skip} = 0.5$. For probabilities greater that this minimum the savings in visibility classification do not recover the time necessary for rendering otherwise invisible objects.



(a) (b)

Figure 18: (a) Dependence of the number of node visibility tests on the relative speed of the walk. (b) Average time spent by the hierarchical visibility algorithm.



(a) (b)

Figure 19: (a) Average frame time depending on the relative speed of the walk. (b) Dependence of the average frame time on the probability p_{skip} using the conservative node skipping.

4.5 Conclusions

We have introduced the concept of occlusion trees and modified occlusion trees. It was shown that the occlusion trees can be used to determine the conservative viewpoint–to–region visibility efficiently by extensively exploiting the spatial coherence. We presented an algorithm determining the visibility of a polygon and polyhedron exactly with respect to the selected occluders. Further, a fast conservative visibility algorithm applicable to regions of more general shapes was described. It was observed that the conservative algorithm performs superior compared to the exact one. The concept of occlusion trees was used in the hierarchical visibility culling for the rendering of complex scenes. It was shown that using the occlusion tree, we can efficiently identify invisible parts of the scene and thus avoid their rendering. For scenes with high depth complexity these savings are significant.

We have introduced a series of modifications of the classical hierarchical visibility culling that make use of temporal and spatial coherence. The node skipping proved to perform well in practice as it saves almost half of the visibility tests that would have to be applied using the classical approach. We have observed that the visibility propagation saves only few visibility tests. This documents the fact that the spatial coherence is exploited well in the classical approach already. It was shown that the conservative node skipping can improve the overall frame time for certain settings.

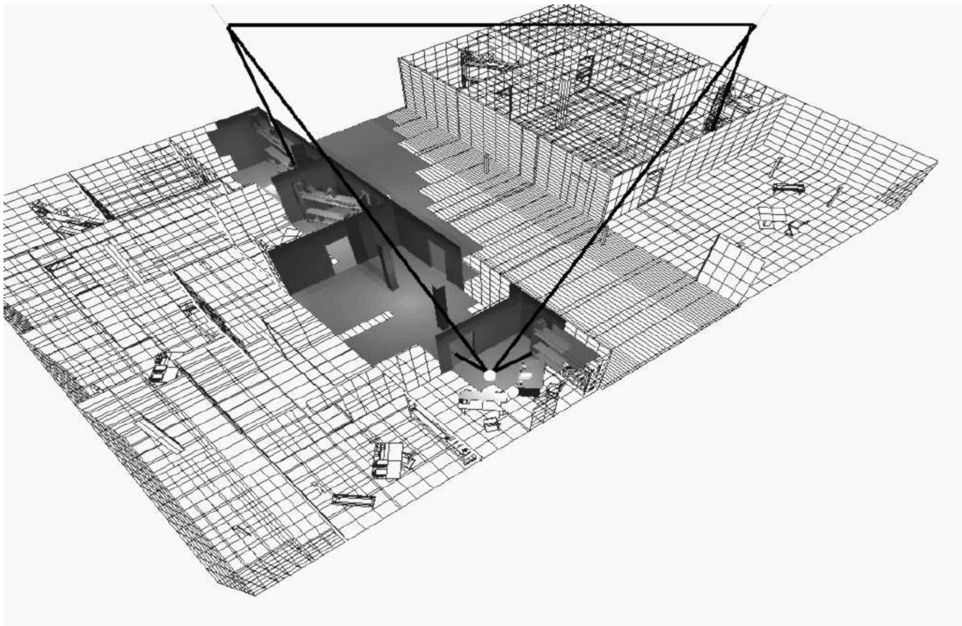


Figure 20: An example of the hierarchical visibility culling from the bird's view. Shaded regions were classified as visible or partially visible. The rest of the scene was found invisible to the observer.

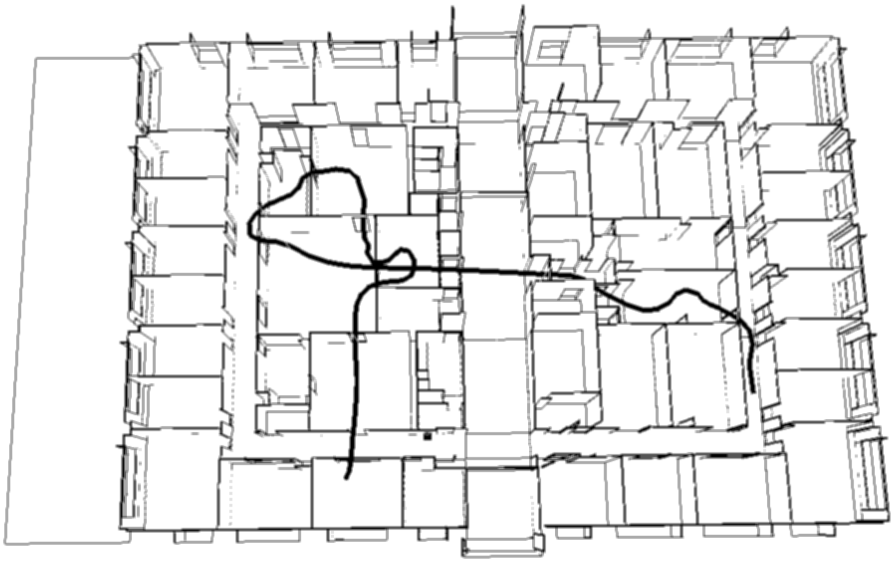


Figure 21: The path used for a walk through the model of the Soda–Hall. For relative speed of the walk equal to 1.0 the walk consists of 980 steps.

5 Visibility from Region

This chapter is devoted to the most general visibility problems. These problems were almost ignored by the research community for a long time [30]. Usually, approximate solutions based on solving lower–dimensional problems or slow and complicated "almost" exact ones were used. The four dimensional character of the line–space involved in the VFR problems makes their solutions difficult. In four dimensions discrete solutions lead to huge amount of samples that must be taken to obtain a reasonably precise solution. Also, if the visibility information is maintained, the discrete representation of four–dimensional space has large memory overhead. The continuous solutions are difficult to implement robustly and their complexity is usually bound by $O(n^4)$ in the worse case. Some methods make use of the fact that for restricted types of objects (e.g. polygons) the "interesting" subset of line–space is only finite and thus the solution can be found by deterministic sampling (VAL algorithms). Nevertheless, in such a case it is difficult to achieve the desired output–sensitivity.

In this chapter we present a brief overview of problems belonging to the VFR class. Further, we introduce an algorithm solving the problem of visibility from a polygonal region that uses *regional occlusion trees*. Finally, we present some preliminary results.

5.1 Previous Work

The first approaches describing visibility from a region belong to the field computer vision. The aspect graph approach [71, 134, 165] partitions the space into cells where from which the projection of the scene does change qualitatively. The *aspect* is a graph describing the *view* of the scene for each cell of the partitioning. The major drawback of this approach is that for polygonal scenes with n polygons there can be $\Theta(n^9)$ cells in the partitioning (or $\Theta(n^6)$ if viewing an object from "outside"). Attempts to overcome the worst case complexity have been made using a scale space [59, 153], thus avoiding some "unimportant" features.

In computer graphics community Airey [2] introduced the concept of *potentially visible sets* (PVS). He assumes existence of a natural subdivision of the environment into cells. For models architectural interiors these cells roughly correspond to rooms. For each cell C the PVS is formed by cells visible from any point of C. Airey uses ray–casting to approximate visibility between two cells of the subdivision and so the PVS is not conservative.

This concept was further elaborated by Teller et al. [191, 193] to establish a conservative PVS. A cell *adjacency graph* is constructed where nodes correspond to cells. Two cells are connected by an edge if there is a transparent boundary (portal) between them. The idea behind their approach is to solve the stabbing line problem through a set of polygonal portals. Teller proposes an exact solution of this problem using five dimensional plücker coordinates [190]. He also presents a simpler and more robust conservative solution [192]. The portal based methods achieve very good performance if applied to static densely occluded environments with a particular structure (architectural interiors). However, for less structured models they can face a combinatorial explosion of complexity [192]. Yagel and Ray [209] present an algorithm, that uses a regular spatial subdivision. Although, their approach is not sensitive to the structure of the model in terms of complexity, its efficiency can be significantly lower compared with the portal based methods.

Plantinga introduced a PVS algorithm based on conservative viewspace partitioning by evaluating visual events [133]. Similar approach was studied by Chrysanthou et al. [37], Cohen–Or et al. [42], and Sagadic [143]. Output–sensitive visibility algorithms for dynamic scenes were studied by Sudarsky and Gotsman [178]. An interesting conservative algorithm determining visibility of an ϵ-neighbourhood of a given viewpoint [44] was used for network based walkthroughs. A recent algorithm of Wang et al. [202], precomputes visibility within beams originating from the restricted viewpoint region. The approach is very similar to the 5D subdivision for ray tracing [159] and so exhibits similar problems, namely the inadequate memory and preprocessing complexities.

Further problems belonging to the VFR class include algorithms used to determine "soft" shadows due to areal light sources. Continuous algorithms for real–time soft shadow generation were studied by Chin and Feiner [33], Loscos and Drettakis [112], and Chrysanthou [34, 35, 39]. Discrete solutions have been proposed by Nishita [129], Brotman and Badler [21], and Soler and Sillion [167]. An exact algorithm computing antipenumbrae of an areal light source was introduced by Teller [190].

In the scope of radiosity global illumination algorithm the VFR algorithms form a core of discontinuity meshing. First approximate discontinuity meshing algorithms were studied by Campbell [24, 23], Lischinski [110], and Heckbert [89]. More elaborated methods were introduced by Drettakis [54, 55] and Stewart and Ghali [171, 172]. These methods are capable of creating complete discontinuity mesh, that encodes all visual events involving a light–source.

Classical global illumination algorithms based on radiosity method evaluate form factors between two patches (or regions) [150]. The visibility determination is a crucial step in form factor evaluation [188, 86, 187, 127, 184]. Similar visibility determination takes place in the scope hierarchical radiosity algorithms [166, 56, 48].

Certain VFP problems that determine visibility over a period of time can be transformed to a corresponding static VFR problem. This transformation can be particularly useful for antialiasing purposes [75].

Pocchiola and Vegter introduced the visibility complex [135]. It describes visibility in two dimensional scenes. The visibility complex has been applied to solve various visibility problems in the plane [138, 140, 139, 130]. The approach was generalised to three dimensions by Durand et al. [57]. Nevertheless, no implementation of the 3D visibility complex is known to the author. Recently, Durand et al. [58] introduced the *visibility skeleton*. It is a graph structure that can be taught of as a skeleton of the 3D visibility complex. The visibility skeleton was verified experimentally and the results indicate that its worst case complexity $O(n^4 \log(n))$ is much better in practice.

Discrete approaches that aim to describe visibility in a unique structure were presented by Chrysanthou et al. [36] and Blais and Poulin [16]. These structures are closely related to the *lumigraph* or *light field* [109]. An interesting approach for two dimensional scenes was presented by Hinkenjann and Müller [91]. Integral geometry tools [27] will hopefully be used in the future to provide a useful "guess" on the size of these structures.

Further in the chapter, we briefly outline an algorithm that resembles the ideas of 3D visibility complex and occlusion trees. The algorithm incrementally constructs a *regional occlusion tree* (ROT) for certain region R (without loss of generality we assume R is polygonal). The scene polygons are processed in an "approximate" front–to–back order with respect to R. At each step the ROT represents the set of lines blocked by the already processed polygons. Additionally the ROT contains a complete description of all visual events that may occur when viewing the scene from R.

5.2 Visibility from a Polygonal Region

Any scene can be decomposed into regions from which the scene has a topologically equivalent view [71]. Boundaries of these regions correspond to surfaces where *visual events* appear. Crossing such boundary induces a change in the topology of the view of the scene (visibility map). In polygonal scenes there are basically two types of visual events [71]: events involving an edge an vertex (VE) and events involving three edges (EEE) of some polygons. The VE events correspond to planes, the EEE events in general form quadratic surfaces.

We use plücker coordinates [131, 210] to map lines * to projective five–dimensional space [20]. The major advantage of this mapping is that it allows us to describe all *visual events* by means hyperplanes in plücker coordinates. This property greatly simplifies the approach since no explicit treatment of three dimensional quadratic surfaces is needed.

In this section we introduce the the plücker coordinates of lines. Further, we discuss visual events that occur in polygonal scenes. Finally, we outline the algorithm using the regional occlusion tree in plücker coordinates to describe visibility from a polygonal region.

5.2.1 Plücker Coordinates of Lines

An oriented line in three–dimensions can be expressed by homogeneous coordinates of its end–points. Let l be a line in $\mathcal{R}^3$ and let $a = (a_0, a_1, a_2, 1)$ and $b = (b_0, b_1, b_2, 1)$ be two distinct points in homogeneous coordinates lying on l. We can represent the line l oriented from a to b by the following matrix:

$$l = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \end{pmatrix} \tag{2}$$

Taking the determinants of the six two–by–two sub–matrices of the above two–by–four matrix, we obtain components of the *plücker coordinates* of the line:

$$p(l) = (\xi_{01}, \xi_{02}, \xi_{03}, \xi_{12}, \xi_{31}, \xi_{23}), \quad \text{with} \quad \xi_{ij} = det \begin{pmatrix} a_i & a_j \\ b_i & b_j \end{pmatrix}. \tag{3}$$

The six numbers ξ_{ij} are interpreted as homogeneous coordinates of a point in $\mathcal{R}^5$ or by duality, as coefficients of a hyperplane. For a given oriented line l these six numbers are unique and they do not depend on the particular distinct points a and b, that were chosen to describe the line l. We call $p(l)$ the *plücker point* of l in projective five–dimensional space $\mathcal{R}^5$.

Define the *plücker hyperplane* of the line l to be the hyperplane in $\mathcal{R}^5$ with vector of coefficients $v(l) = (\xi_{23}, \xi_{31}, \xi_{12}, \xi_{03}, \xi_{02}, \xi_{01})$. The plücker hyperplane is defined:

$$h(l) = \{p \in \mathcal{R}^5 : v(l) \cdot p = 0\} \tag{4}$$

The plücker hyperplane induces closed positive and negative halfspace given by:

$$\begin{aligned} h^+(l) &= \{p \in P^5 : v(l) \cdot p \geq 0\} \\ h^-(l) &= \{p \in P^5 : v(l) \cdot p \leq 0\} \end{aligned} \tag{5}$$

Somewhat asymmetric order in the specification of the plücker coordinates was adopted in [131] to produce positive signs in some identities about the plücker coordinates. Let us denote the new permuted coordinates by a six-tuple

$$\Pi(l) = (\pi_{l0}, \pi_{l1}, \pi_{l2}, \pi_{l3}, \pi_{l4}, \pi_{l5}). \tag{6}$$

The permutation from the original plücker coordinates is given by the following rules:

$$\begin{aligned} \xi_{01} &\rightarrow \pi_{l0} \\ \xi_{02} &\rightarrow \pi_{l1} \\ \xi_{03} &\rightarrow \pi_{l2} \\ \xi_{12} &\rightarrow \pi_{l3} \\ \xi_{31} &\rightarrow \pi_{l5} \\ \xi_{23} &\rightarrow \pi_{l4} \end{aligned} \tag{7}$$

*See [83] for an example of other line parametrization.

For an oriented line l defined by a pair of distinct points p, $q \in \mathcal{R}^3$ with homogeneous coordinates $p = (p_x, p_y, p_z, 1)$ and $q = (q_x, q_y, q_z, 1)$, the corresponding projective six–tuple $\Pi(l)$ is enumerated as follows:

$$\begin{aligned}
\pi_{l0} &= \xi_{01} = p_x q_y - q_x p_y \\
\pi_{l1} &= \xi_{02} = p_x q_z - q_x p_z \\
\pi_{l2} &= \xi_{03} = p_x - q_x \\
\pi_{l3} &= \xi_{12} = p_y q_z - q_y p_z \\
\pi_{l4} &= \xi_{31} = p_z - q_z \\
\pi_{l5} &= \xi_{23} = p_y - q_y
\end{aligned} \tag{8}$$

If a and b are two directed lines, and Π_a, Π_b their corresponding plücker mappings, the relation $side(a, b)$ is defined as the permuted inner product $\Pi_a \odot \Pi_b$:

$$\Pi_a \odot \Pi_b = (\pi_{a0}\pi_{b4} + \pi_{a1}\pi_{b5} + \pi_{a2}\pi_{b3} + \pi_{a4}\pi_{b0} + \pi_{a5}\pi_{b1} + \pi_{a3}\pi_{b2}) \tag{9}$$

This relation of *sidedness* can be interpreted with the right–hand rule (Figure 22). If the thumb of right–hand is directed along line a, then:

- ◇ $side(a, b) > 0$, if line b is oriented in the direction the fingers points,
- ◇ $side(a, b) = 0$, if lines a and b are coplanar (i.e., intersect or are parallel),
- ◇ $side(a, b) < 0$, if line b points against the fingers.

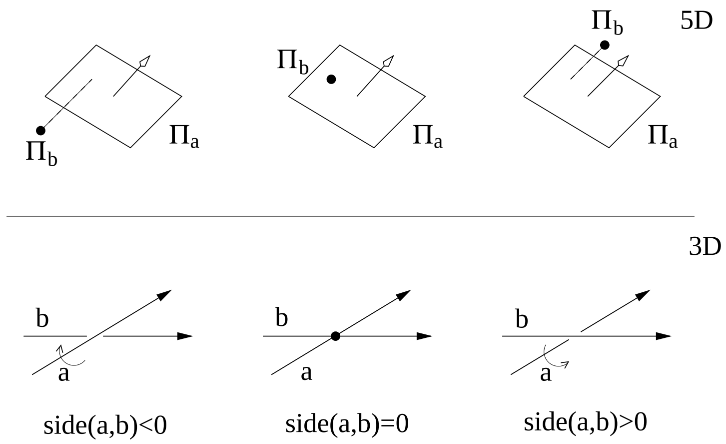


Figure 22: The $side(a, b)$, interpreted as the right–hand rule.

The main advantage of transforming lines into plücker coordinates is that detecting an incidence of lines in $\mathcal{R}^3$ is equivalent to computing the inner product of a homogeneous point (mapping of one line) with a hyperplane (mapping of the other). This can be expressed by the following lemma.

Lemma 1 *Lines l and l' intersect or are parallel (i.e. meet in infinity) if and only if $p(l) \in h(l')$*

Note that the condition 4 states in terms of plücker coordinates that any line always meets itself.

An important feature of the plücker coordinates is that although every directed line in $\mathcal{R}^3$ maps to a point in plücker coordinates, not every tuple of six real numbers corresponds to a line in three-dimensional space. Only the points Π satisfying the condition

$$\Pi \odot \Pi \equiv \xi_{01}\xi_{23} + \xi_{02}\xi_{31} + \xi_{03}\xi_{12} = 0 \tag{10}$$

correspond to real lines in $\mathcal{R}^3$. All other points map to lines which are said to be *imaginary*. The set of points in $\mathcal{R}^5$ satisfying the Equation 10 represents the *plücker quadric*, also known as *Klein quadric* or *Grassmanian manifold* (Figure 23). We can observe, that the six plücker coordinates of a real line are not independent. First they describe an oriented projective space and second they must satisfy the equation 10.

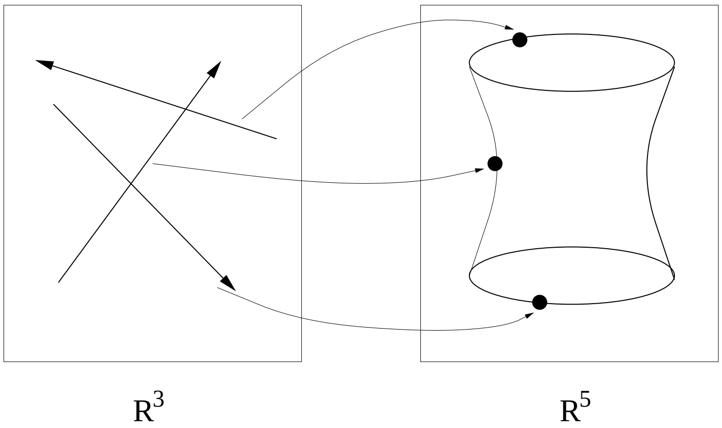


Figure 23: Real lines map on points on the Plücker quadric.

5.2.2 Visual Events and Extremal Lines

Let $\mathcal{H}$ be an arrangement of hyperplanes in $\mathcal{R}^5$. The hyperplanes of $\mathcal{H}$ correspond to plücker mappings of lines containing edges of the set of polygons $\mathcal{P}$. Denote $\mathcal{S}$ the four dimensional cross–section of the arrangement $\mathcal{A}$ and the plücker quadric. Then arrangement $\mathcal{S}$ yields all visual events [131].

The intersections visual event surfaces are bounded by *extremal lines*, i.e. lines incident on some four edges of scene polygons [131].

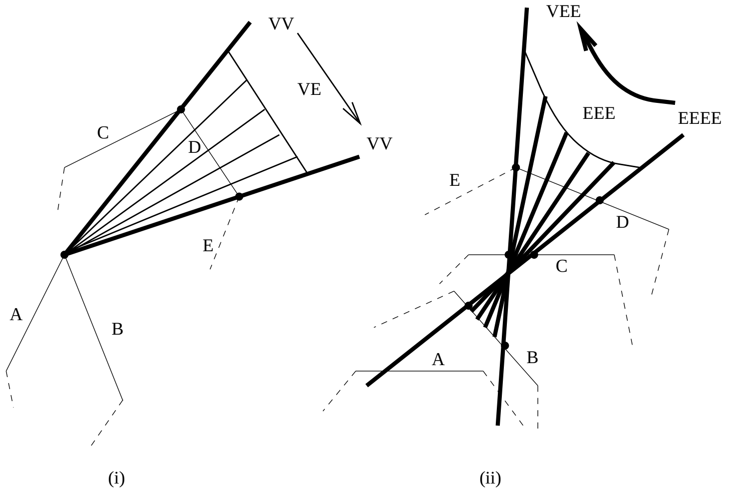


Figure 24: Event surfaces – VE plane (i) and EEE quadratic surface (ii).

Assume that an extremal line l is incident upon four of the generator edges e_k. Consequently, the plücker point Π_l must lie on four of the hyperplanes h_k in $\mathcal{R}^5$, and must therefore lie on a *1–face*, or edge, of the arrangement $\mathcal{H}$. Thus, we can find all extremal lines of a given set of polygons by examining the edges of $\mathcal{H}$ for intersections with the plücker quadric. The extremal line corresponding to each intersection can be determined from the four relevant generator edges e_k as shown by Teller and Hohmeyer [189].

There are three types of extremal lines: vertex-vertex (VV) lines, vertex-edge-edge (VEE) lines, or quadruple edge (4E) lines. Imagine "sliding" an extremal line (of any type) away from its initial position, by relaxing exactly one of the four edge constraints determining the line. The surface, or *swath*, swept out by the sliding line must either be planar (if it remains tight on a vertex) or a regulus, whose three generator lines embed three polygon edges. In the terminology of [71], swaths are VE or EEE *event surfaces* important in the construction of aspect graphs, since they define places at which visibility events occur.

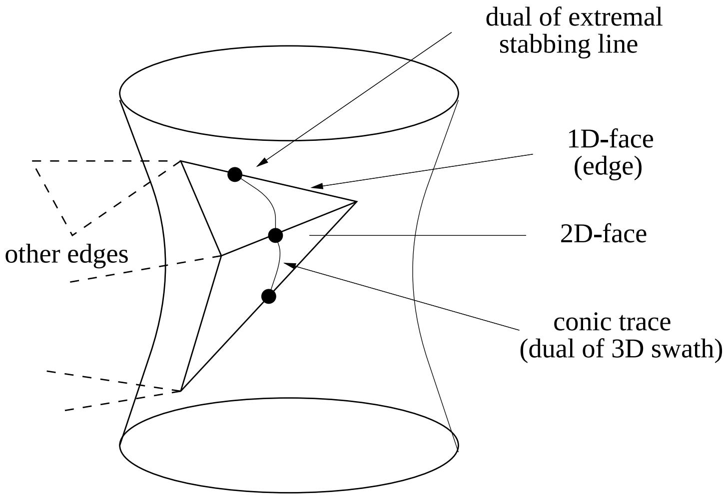


Figure 25: Duals of 3D swaths as a traces on the Plücker surface in 5D.

Figure 24-i shows an extremal VV line tight on four edges A,B,C, and D. Relaxing constraint C yields a VE (planar) event surface tight on A,B, and D. Eventually, the sliding line encounters an obstacle (in this case edge E), and terminates at a VV line tight on A,B,D, and E. Figure 24-ii depicts an extremal 4E line tight on the mutually skew edges A,B,C, and D. Relaxing constraint A produces a EEE (regulus) event surface tight on B,C and D. The sliding line eventually encounters edge E and induces an extremal VEE line, terminating the event surface.

Consider the same situation in the five dimensional projective space generated by the plücker mapping (see figure 25). Event surfaces map to particular points in the plücker space, namely, the intersections of the edges of the arrangement $\mathcal{H}$ with the plücker quadric. Since event surfaces are one–parameter entities, they correspond to curves in plücker coordinates. These curves are intersections of the 2–faces of $\mathcal{H}$ with the plücker surface and therefore conics (in 5D the 2D–faces of $\mathcal{H}$ are planar, and determined by the intersection of some three hyperplanes h_k).

5.2.3 Regional Occlusion Tree

Let the region R which we want to describe visibility from be a convex polygon. Similarly let all scene polygons be convex. For sake of simplicity assume that there is a front–to–back ordering of polygons with respect to the light–source [35]. The polygons are processed in this order and the dual of the line set occluded by the current polygon B is merged into the ROT. The ROT represents a union of all lines blocked by already processed polygons. In other words the ROT represents cells of the arrangement $\mathcal{H}$, that correspond to sets of lines that pierce R and are blocked by some scene polygon. A two dimensional illustration of the concept of ROT is depicted in Figure 26.

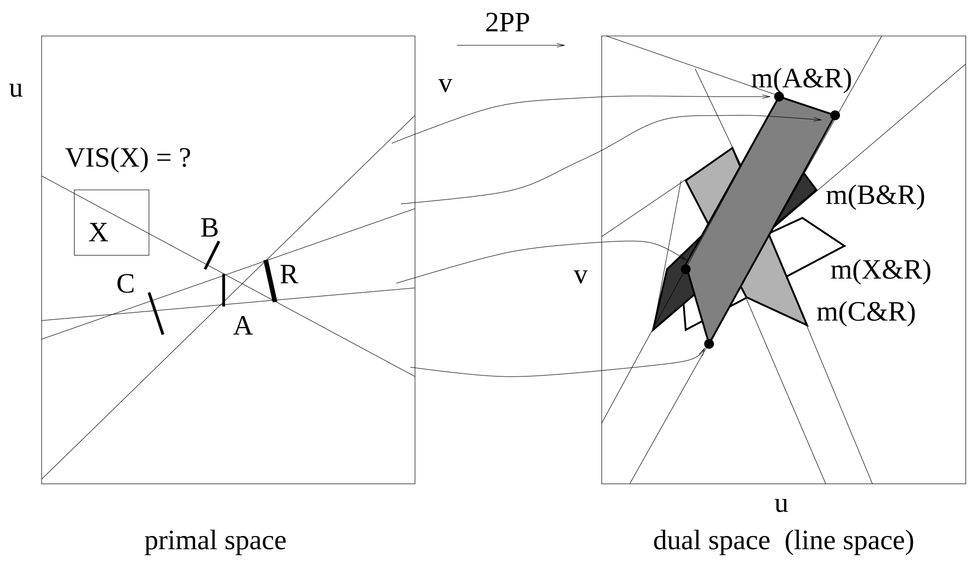


Figure 26: A two dimensional example the ROT. The binary space partitioning $\mathcal{Q}$ in line–space (right) represents a set of lines originating from the source line segment (R) that are blocked by some other scene line segment (occluder). Visibility of a region X in primal space can be determined by filtering the dual mapping (in 2D this is a polygon) of the lines passing through X and R $m(X\&R)$ down the ROT that corresponds to the partitioning $\mathcal{Q}$. On the example a 2D *two plane patrametrization* (2PP) of lines was used, that has similar properties as the plücker mapping for lines in $\mathcal{R}^3$.

First we build a five–dimensional polyhedron $\mathcal{P}$, that corresponds to all lines that are blocked by B. $\mathcal{P}$ is bounded by hyperplanes corresponding to edges of R and B (see Figure 27). Since $\mathcal{P}$ can be unbounded, we add two hyperplanes ("caps") to bound it. The positive halfspaces induced by the caps must contain the complete cross–section of $\mathcal{P}$ and the plücker quadric.

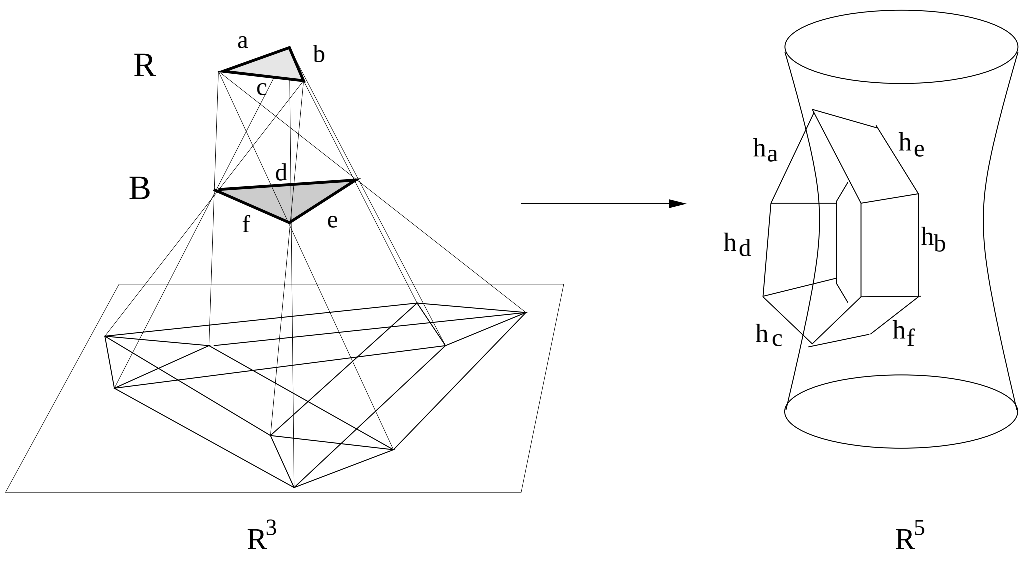


Figure 27: The extremal lines in 3D and the polyhedron $\mathcal{P}$ representing the set of occluded lines in plücker coordinates. The edges of R and B map to hyperplanes that bound $\mathcal{P}$. The visual event surfaces (shown as cross–sections with a plane in 3D) map to intersections of 2–faces of $\mathcal{P}$ with the plücker quadric.

Then the polyhedron is "filtered" down the ROT splitting it where appropriate. Reaching out–leaves the set $\mathcal{S}$ of polyhedra is obtained. These polyhedra correspond to sets of sight–lines between R and B. Also, the polyhedra yield a complete discontinuity mesh as well as the the corresponding backprojections. The ROT is enlarged by planes bounding polyhedra of $\mathcal{S}$ to merge lines blocked by P into the ROT.

If the set $\mathcal{S}$ is empty, B is invisible from R. If union of all polyhedra from $\mathcal{S}$ equals to $\mathcal{P}$, then the B is completely visible. Otherwise, it is partially visible since some lines between R and B are blocked.

5.3 Preliminary Results

We have implemented a preliminary version of the outlined algorithm. The most important operation of the ROT construction is the process of splitting a five dimensional polyhedron by a plane. We have used the approach proposed by Bajaj and Pascucci [10]. The algorithm is designed for floating point arithmetic and the authors claim that it is robust. Nevertheless, we have encountered problems in some degenerate cases since these are not solved consistently in this approach. Unfortunately we are not aware of any other implementation of the higher dimensional polyhedron splitting.

The preliminary results are presented to support our hope, that implementing the polyhedron splitting algorithm robustly leads to a robust out–sensitive algorithm for VFR problem.

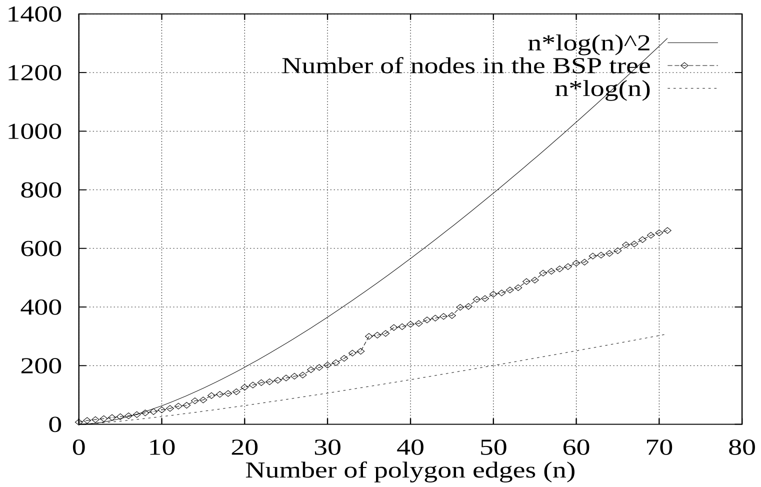


Figure 28: Dependence of the number of nodes in the 5D–occlusion tree on the number of polygon edges merged in the tree. The scene consisted of 72 randomly generated triangles. One of them was chosen as a region (light source) from which the visibility was described.

We have measured the dependence of the size of ROT on the number of edges of polygons added into the tree. The results are depicted in Figure 28. Although the worst case size of a BSP tree in $\mathcal{R}^5$ is $\Theta(n^5)$ it is obvious that in our application it performs much more promising.

5.4 Conclusion

In this chapter we have outlined the concept of regional occlusion trees. We have used mapping to plücker coordinates to simplify the representation of visibility from a polygonal area in the polygonal scene. The algorithm exploits spatial coherence of visibility by utilising a higher dimensional hierarchical data structure – the regional occlusion tree. The results presented are only preliminary since we currently lack a robust implementation of the algorithm for splitting a convex polyhedron by a hyperplane in $\mathcal{R}^5$.

6 Summary

In this report we have introduced a new classification of visibility algorithms that is based on the dimension of the line–space involved in the particular problem. In particular we have discussed the following three classes: visibility along a line, visibility from a point and visibility from a region. For each class we have presented an algorithm that extends previous results was presented.

In Chapter 3 we have described an algorithm for ray shooting that uses a hierarchical spatial subdivision (kD–tree) to accelerate the ray shooting queries. The kD–tree is constructed according to a surface area heuristics that adapts to the underlying geometry. The proposed algorithm uses neighbour links (trees) to eliminate certain overhead of traversing the hierarchical structure.

In Chapter 4 a hierarchical VFP algorithm was described. The algorithm uses the occlusion tree or the modified occlusion tree to represent an occlusion map of selected occluders. The visibility is determined using a kD–tree based spatial hierarchy. The proposed algorithm makes use of spatial coherence of visibility in object space as well as the spatial coherence of occluders. Further, we have described some general methods that increase the amount of temporal and spatial coherence exploited. The algorithm was used to achieve an output–sensitive hidden surface removal in large scenes with high depth complexity.

Finally, in Chapter 5 we have outlined an algorithm describing visibility from a polygonal region. It uses a regional occlusion tree to represent sets of lines blocked by a given set of occluders. The regional occlusion tree contains all information about visual events that may occur if the viewpoint is moving within the region the tree was built for.

7 Future Work

In the future we are going to create a unified framework for the algorithms presented in this report. Precomputing visibility for each region of the spatial hierarchy we can accelerate both the ray shooting and the hidden surface removal. Similarly, determining visibility from a point–light source can significantly accelerate ray shooting queries in the scope of ray tracing. We feel it important to study algorithms of various classes together, since they share many essential principles. Our aim is to create a robust, flexible, and output–sensitive algorithms that make use of various types of coherence [82].

7.1 Visibility along Line

First of all, we would like to improve the properties of the kD–tree based spatial subdivision for ray shooting purposes. Using the surface area heuristics the partitioning planes at the top of the kD–tree are placed in a position that reflects the global distribution of objects. Some planes can have rather unfavourable impact on the spatial subdivision locally. The problem can be solved by a data structure based on the kD–tree, that joins some leaf–cells together. The new cells create a *pseudo–cover* of the whole scene space.

The kD–tree implies that all cells of the spatial subdivision are parallelepipeds. A more general BSP tree that would improve the adaptability of the data structure to the underlying geometry. The surface–area heuristics used in the construction of the kD–tree assumes that rays are distributed uniformly. A better heuristic could consider the view–dependent distribution of rays and the visibility between regions of the scene.

The major problem of ray shooting algorithms is the lack of coherence exploited if more rays are cast. We are developing an algorithm that establishes *longest common traversal sequences* (LCTS) for BSP trees to exploit coherence in data structure traversal for certain sets of rays.

7.2 Visibility from Point

Many proposals for future work have been encountered during our research in the field of VFP problems. The hierarchical visibility algorithm assumes that the occluders are sufficiently large convex polygons. This assumption is often false in practice. A simplification of objects for the purposes of visibility algorithms is very desirable. The simplified polygonal objects should keep occlusion properties of objects they were generated from.

Another topic for future work concerns the occluder selection. It can be advantageous to determine the set of potential occluders in preprocessing [92, 45]. We currently develop an algorithm that computes potential occluders for a given region. The algorithm uses a more sophisticated sampling scheme to determine the asset of an occluder. Regarding the modified occlusion tree, we are going to give a probabilistic analysis of the conservative visibility algorithm. Based on this analysis, we want to increase the precision of the algorithm. This should be achieved by inserting additional nodes to the MOT, where the probability of visibility misclassification is high. Currently, the occlusion tree is built from scratch in each rendering frame. We study the possibility of exploiting the temporal coherence in the construction of the occlusion tree.

The conservative node visibility classification and the probabilistic modification of the visibility propagation algorithm currently use fixed probabilities. A more elaborate methods should be used [66]. For example an average "survival" time of visible nodes could be predicted and used in a sampling scheme ensuring that the visibility of a node is correctly updated within certain number of frames (such as russian roulette). This approach would eventually adapt to properties of the environment surrounding the viewpoint and the efficiency of the visibility classification algorithm. If the visibility algorithm is unsuccessful, most regions would be visible and their survival time would increase. These nodes would be tested with lower probability hence the wasted effort for their visibility classification would be reduced.

Currently the visibility propagation method determines only if a node is visible or invisible. Nevertheless, some heuristics could be used to determine that a node is partially visible with high probability and hence again avoid the visibility test itself. This modification would benefit mainly in sparsely occluded environments where many small regions (leaves of the hierarchy) are classified as partially visible.

7.3 Visibility from Region

The implementation of the algorithm proposed in Section 5 assumes a robust polyhedron splitting operation. Currently, the polyhedron splitting is the major limitation of the implementation. We will study the structure of regional occlusion trees and identify their possible applications. The solution of the VFR problem can be used to accelerate algorithms from both VAL and VFP classes. We want to apply the regional occlusion trees to assist global illumination techniques in determining regions of illumination discontinuities.

8 References

[1] P. Agarwal, T. Murali, and J. Vitter. Practical techniques for constructing binary space partitions for orthogonal rectangles. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 382–384, 1997.

[2] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In *1990 Symposium on Interactive 3D Graphics*, pages 41–50. ACM SIGGRAPH, Mar. 1990. Also available as Technical Report TR-90-001, Computer Science Department, University of North Carolina.

[3] T. Akimoto, K. Mase, and Y. Suenaga. Pixel-selected ray tracing. *IEEE Computer Graphics and Applications*, 11(4):14–22, July 1991.

[4] D. G. Aliaga and A. A. Lastra. Architectural walkthroughs using portal textures. In R. Yagel and H. Hagen, editors, *IEEE Visualization ́97*, pages 355–362. IEEE, Nov. 1997.

[5] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In G. Marechal, editor, *Eurographics '87*, pages 3–10. North-Holland, Aug. 1987.

[6] J. Arvo. Linear-time voxel walking for octrees. *Ray Tracing News (available at htpp://www.acm.org/tog/resources/RTNew s/html/rtnews2d.html)*, 1(5), 1988.

[7] J. Arvo and D. Kirk. Fast ray tracing by ray classification. In M. C. Stone, editor, *(SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64, July 1987.

[8] J. Arvo and D. Kirk. *A survey of ray tracing acceleration techniques*, pages 201–262. Academic Press, 1989.

[9] J. Arvo and D. Kirk. Particle transport and image synthesis. *Computer Graphics*, 24(4):63–66, Aug. 1990.

[10] C. L. Bajaj and V. Pascucci. Splitting a complex of convex polytopes in any dimension. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 88–97, 1996.

[11] D. Bartz, M. Meissner, and T. Hüttner. Extending graphics hardware for occlusion queries in opengl. In *Proceedings of the 1998 Workshop on Graphics Hardware, Lisbon, Portugal*, pages 97–104, 1998.

[12] M. D. Berg. Ray shooting, depth orders and hidden surface removal. In *Lecture Notes in Computer Science*, volume 703. Springer Verlag, New York, 1993.

[13] G. Bishop, H. Fuchs, L. McMillan, and E. J. Scher Zagier. Frameless rendering: Double buffering considered harmful. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 175–176. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[14] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz. Designing a PC game engine. *IEEE Computer Graphics and Applications*, 18(1):46–53, Jan./Feb. 1998.

[15] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 207–219, Hannover, Germany, June 1998. IEEE, NY. Also available as http://www.cgg.cvut.cz/~bittner/paper-copy.ps.gz.

[16] M. Blais and P. Poulin. Sampling visibility in three-space. In *Proc. of the 1998 Western Computer Graphics Symposium*, pages 45–52, Apr. 1998.

[17] J. Blinn. *Jim Blinn's Corner: A Trip Down the Graphics Pipeline*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1996. This book is a collection of columns from the journal IEEE Computer Graphics and Applications.

[18] J. Blinn. *Jim Blinn's Corner: Dirty Pixels*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1998. This book is a collection of columns from the journal IEEE Computer Graphics and Applications.

[19] J. Blinn. Jim blinn's corner: Ten more unsolved problems in computer graphics. *IEEE Computer Graphics & Applications*, 18(5), Sept. – Oct. 1998. ISSN 0272-1716.

[20] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. cambridge University Press, 1998.

[21] L. S. Brotman and N. I. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(10):5–12, Oct. 1984.

[22] W. C, C. A, S. S, F. D, and H. F. Multiresolution bsp trees. In *Graphics Interface 97*, pages ??–??, 1997. University of Texas.

[23] A. T. Campbell, III. *Modeling Global Diffuse Illumination for Image Synthesis*. PhD thesis, CS Dept, University of Texas at Austin, Dec. 1991. Tech. Report TR-91-39.

[24] A. T. Campbell, III and D. S. Fussell. Adaptive mesh generation for global diffuse illumination. In F. Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 155–164, Aug. 1990.

[25] F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: A new solution for ray-tracing complex scenes. *Computer Graphics Forum*, 14(3):C/371–382, 1995.

[26] F. Cazals and C. Puech. Bucket-like space partitioning data-structures with applications to ray-tracing. In *13th ACM Symposium on Computational Geometry*, page To Appear, Nice, 1997.

[27] F. Cazals and M. Sbert. Some integral geometry tools to estimate the complexity of 3d scenes. Technical Report RR-3204, The French National Institue for Research in Computer Science and Control (INRIA), July 1997.

[28] B. Chamberlain, T. DeRose, D. Lischinski, D. Salesin, and J. Snyder. Fast rendering of complex environments using a spatial hierarchy. In *Graphics Interface*, pages 132–141, May 1996.

[29] J. Chapman, T. W. Calvert, and J. Dill. Exploiting temporal coherence in ray tracing. In *Proceedings of Graphics Interface '90*, pages 196–204, Toronto, Ontario, May 1990. Canadian Information Processing Society.

[30] B. Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., Apr. 1996.

[31] H. Chen and W. Wang. The feudal priority algorithm on hidden-surface removal. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 55–64. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[32] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 99–106, July 1989.

[33] N. Chin and S. Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. In D. Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 21–30, Mar. 1992.

[34] Y. Chrysanthou. A fast and accurate shadow algorithm for dynamic scenes. In *A symposium of the Framework for Immersive Virtual Environments Working Group*, pages ??–??, Geneva, Switzerland, Oct. 1996.

[35] Y. Chrysanthou. *Shadow Computation for 3D Interaction and Animation*. PhD thesis, QMW, Dept of Computer Science, Jan. 1996.

[36] Y. Chrysanthou, D. Cohen-Or, and D. Lischinski. Fast approximate quantitative visibility for complex scenes. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 23–31, Hannover, Germany, June 1998. IEEE, NY.

[37] Y. Chrysanthou, D. Cohen-Or, and E. Zadicario. Viewspace partitioning of densely occluded scenes. Abstract of a video presentation, at the 13th Annual ACM Symposium on Computational Geometry, Minnesota, pages 413–414, June 1998.

[38] Y. Chrysanthou and M. Slater. Computing dynamic changes to bsp trees. In A. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, volume 11, pages 321–332, Sept. 1992.

[39] Y. Chrysanthou and M. Slater. Incremental updates to scenes illuminated by area light sources. In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 103–114, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[40] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, Oct. 1976.

[41] J. G. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

[42] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. In *EUROGRAPHICS'98*, 1998.

[43] D. Cohen-Or and A. Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum*, 14(3):C/171–C/180, Sept. 1995.

[44] D. Cohen-Or and E. Zadicario. Visibility streaming for network-based walkthroughs. In *Graphics Interface*, pages 1–7, June 1998.

[45] S. Coorg and S. Teller. A spatially and temporally coherent object space visibility algorithm. Technical Report TM-546, Department of Computer Graphics, MIT, Feb. 1996.

[46] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proceedings of the Twelfth Annual ACM Symposium on Computational Geometry*, Philadelphia, PA, May 1996.

[47] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 83–90, New York, Apr.27–30 1997. ACM Press.

[48] K. Daubert, H. Schirmacher, F. X. Sillion, and G. Drettakis. Hierarchical lighting simulation for outdoor scenes. In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 229–238, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[49] D.Cohen and Z.Sheffer. Proximity clouds - an acceleration technique for 3d grid traversal. *The Visual Computer*, 11:27–38, 1994.

[50] M. de Berg. Generalized hidden surface removal. In A.-S. ACM-SIGGRAPH, editor, *Proceedings of the 9th Annual Symposium on Computational Geometry (SCG '93)*, pages 1–10, San Diego, CA, USA, May 1993. ACM Press.

[51] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[52] F. Dèvai. On the computational requirements of virtual reality systems. In *State of Art, Eurographics'97*, 1997.

[53] O. Devillers. The macro-regions: an efficient space subdivision structure for ray tracing. In W. Hansmann, F. R. A. Hopgood, and W. Strasser, editors, *Eurographics '89*, pages 27–38. Elsevier / North-Holland, Sept. 1989.

[54] G. Drettakis. *Structured Sampling and Reconstruction of Illumination for Image Synthesis.* CSRI Technical Report, Department of Computer Science, University of Toronto, Toronto, Ontario, Jan. 1994.

[55] G. Drettakis and E. Fiume. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 223–230, 1994.

[56] G. Drettakis and F. Sillion. Interactive update of global illumination using A line-space hierarchy. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 57–64. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[57] F. Durand, G. Drettakis, and C. Puech. The 3D visibility complex: A new approach to the problems of accurate visibility. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 245–256, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4.

[58] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 89–100. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[59] D. Eggert, K. Bowyer, C. Dyer, H. Christensen, and D. Goldgof. The scale space aspect graph. *PAMI*, 15(11):1114–1130, November 1993.

[60] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Co., Reading, MA, 2nd edition, 1990. T 385.C587.

[61] A. Fournier and P. Poulin. A ray tracing accelerator based on a hierarchy of 1D sorted lists. In *Proceedings of Graphics Interface '93*, pages 53–61, Toronto, Ontario, May 1993. Canadian Information Processing Society.

[62] W. R. Franklin and M. S. Kankanhalli. Parallel object-space hidden surface removal. *Computer Graphics*, 24(4):87–94, Aug. 1990.

[63] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 124–133, July 1980.

[64] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

[65] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West. A beam tracing approach to acoustic modeling for interactive virtual environments. In *Computer Graphics Proceedings, Annual Conference Series, 1998 (SIGGRAPH '98)*, page ???, July 1998.

[66] T. A. Funkhouser. *Database and Display Algorithms for Interactive Visualization of Architectural Models.* PhD thesis, CS Division, UC Berkeley, 1993.

[67] D. Fussell and K. R. Subramanian. Fast ray tracing using K-D trees. Technical Report TR-88-07, U. of Texas, Austin, Dept. Of Computer Science, Mar. 1988.

[68] W. F. Garrett, H. Fuchs, M. C. Whitton, and A. State. Real-time incremental visualization of dynamic ultrasound volumes using parallel BSP trees. In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.

[69] D. W. George, F. X. Sillion, and D. P. Greenberg. Radiosity redistribution for dynamic environments. *IEEE Computer Graphics and Applications*, 10(4):26–34, July 1990.

[70] C. Georges. Obscuration culling on parallel graphics architectures. Technical Report TR95-017, Department of Computer Science, University of N. Carolina, Chapel Hill, 1995.

[71] Z. Gigus and J. Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, Feb. 1990.

[72] A. S. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, Mar. 1988.

[73] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[74] D. Gordon and S. Chen. Front-to-back display of BSP trees. *IEEE Computer Graphics and Applications*, 11(5):79–85, Sept. 1991.

[75] C. W. Grant. Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-space. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 79–84, July 1985.

[76] C. W. Grant. *Visibility Algorithms in Image Synthesis.* PhD thesis, U. of California, Davis, 1992.

[77] N. Greene. Detecting intersection of a rectangular solid and a convex polyhedron. In P. Heckbert, editor, *Graphics Gems IV*, pages 74–82. Academic Press, Boston, MA, 1994.

[78] N. Greene. Hierarchical polygon tiling with coverage masks. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 65–74. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[79] N. Greene and M. Kass. Error-bounded antialiased rendering of complex environments. *Computer Graphics*, 28(Annual Conference Series):59–66, July 1994.

[80] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Proceedings of SIGGRAPH '93*, pages 231–238. ACM Press, Aug. 1993.

[81] E. Groeller. Oct-tracing animation sequences. In *Summer school in computer graphics in Bratislava (SCCG93)*, pages 96–101, June 1993.

[82] E. Gröller and W. Purgathofer. Coherence in computer graphics. Research paper, Institute for Computer Graphics, Technical University Vienna, 1995.

[83] X. Gu, S. J. Gortier, and M. F. Cohen. Polyhedral geometry and the two-plane parameterization. In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 1–12, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[84] E. A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, Nov. 1987. also in SIGGRAPH '87, '88, '89 Introduction to Ray Tracing course notes, code available via FTP from princeton.edu:/pub/Graphics.

[85] E. A. Haines and D. P. Greenberg. The light buffer: A ray tracer shadow testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, Sept. 1986.

[86] E. A. Haines and J. R. Wallace. Shaft culling for efficient ray-traced radiosity. In P. Brunet and F. W. Jansen, editors, *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, New York, NY, 1994. Springer-Verlag. also available via FTP from princeton.edu:/pub/Graphics/Papers.

[87] V. Havran. Spatial data structures for visibility computation. Technical Report DC-PSR-97-05, CTU–FEE Prague, May 1997. Postgraduate Study Report, 34 pages, also available as ftp://www.cgg.cvut.cz/outgoing/havran/minimum/dc-psr-97-04.ps.gz.

[88] V. Havran, J. Bittner, and J. Žára. Ray tracing with rope trees. In *Proceedings of 13th Spring Conference on Computer Graphics*, pages 130–139, Budmerice, 1998. Also available as http://www.cgg.cvut.cz/~bittner/sccg98.ps.gz.

[89] P. S. Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pages 203–216, Bristol, UK, May 1992.

[90] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *Computer Graphics (SIGGRAPH'84 Proceedings)*, 18(3):119–127, July 1984.

[91] A. Hinkenjann and H. Müller. Hierarchical blocker trees for global visibility calculation. Research Report 621/1996, University of Dortmund, Universität Dortmund, 44221 Dortmund, Germany, Aug. 1996.

[92] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1997.

[93] J. Huerta, M. Chover, J. Ribelles, and R. Quiros. Constructing and rendering of multiresolution binary space partitioning trees. In H. P. Santo, editor, *Edugraphics + Compugraphics Proceedings*, pages 212–221, P.O. Box 4076, Massama, 2745 Queluz, Portugal, Dec. 15, 1997. GRASP- Graphic Science Promotions & Publications.

[94] A. James and A. Day. The priority face determination tree for hidden surface removal. In *Computer Graphics Forum*, volume 17, pages 55–71, march 1998.

[95] F. W. Jansen. Data structures for ray tracing. In L. R. A. Kessener, F. J. Peters, and M. L. P. van Lierop, editors, *Data Structures for Raster Graphics*, pages 57–73. Springer-Verlag, New York, 1986. Eurographic Seminar.

[96] H. W. Jensen and N. J. Christensen. Efficiently Rendering Shadows Using the Photon Map. In H. P. Santo, editor, *Edugraphics + Compugraphics Proceedings*, pages 285–291, P.O. Box 4076, Massama, 2745 Queluz, Portugal, Dec. 12, 1995. GRASP- Graphic Science Promotions & Publications.

[97] D. Jevans. Object space temporal coherence for ray tracing. In *Proceedings of Graphics Interface '92*, pages 176–183, Toronto, Ontario, May 1992. Canadian Information Processing Society.

[98] S. B. Jr. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132, Sept. 1988.

[99] J. T. Kajiya. The rendering equation. In D. C. Evans and R. J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 143–150, Aug. 1986.

[100] M. Kaplan. Space-tracing: A constant time ray-tracer. In *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*, pages 149–158. Addison Wesley, July 1985.

[101] A. Keller. Instant radiosity. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 49–56. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[102] K. S. Klimaszewski and T. W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, Jan./Feb. 1997.

[103] A. J. F. Kok and F. W. Jansen. Adaptive sampling of area light sources in ray tracing including diffuse interreflection. *Computer Graphics Forum (Eurographics '92)*, 11(3):289–298, Sept. 1992.

[104] A. J. F. Kok, F. W. Jansen, and C. Woodward. Efficient, complete radiosity ray tracing using a shadow-coherence method. *The Visual Computer*, 10:19–33, oct 1993.

[105] I. Kolingerov. Convex polyhedron-line intersection detection using dual representation. *The Visual Computer*, 13(1):42–49, 1997. ISSN 0178-2789.

[106] S. Kumar and D. Manocha. Hierarcical visibility culling for spline models. In W. A. Davis and R. Bartels, editors, *Graphics Interface '96*, pages 142–150. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996. ISBN 0-9695338-5-3.

[107] S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical back-face computation. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 235–244, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4.

[108] B. Kwon, D. S. Kim, K.-Y. Chwa, and S. Y. Shin. Memory-efficient ray classification for visibility operations. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):193–201, jul–sep 1998.

[109] M. Levoy and P. Hanrahan. Light field rendering. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[110] D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, Nov. 1992.

[111] N. Loofbourrow. Optimizing ray tracing with visual coherence. Technical Report CMU-CS-93-209, Carnegie-Mellon University, Department of Computer Science, 1993.

[112] C. Loscos and G. Drettakis. Interactive high-quality soft shadows in scenes with moving objects. *Computer Graphics Forum*, 16(3):C219–C230, Sept. 4–8 1997.

[113] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, Apr. 1995. ISBN 0-89791-736-7.

[114] A. Lukaszewski and A. Formella. Fast penumbra calculation in ray tracing. In *Proceedings of WSCG'98, the 6th International Conference in Central Europe on Computer Graphics and Visualization '98*, pages 238–245, Feb. 1998.

[115] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, 6(6):153–65, 1990. criteria for building octree (actually BSP) efficiency structures.

[116] D. Marshall, D. S. Fussell, and I. A.T. Campbell. Multiresolution rendering of complex botanical scenes. In *Graphics Interface '97*, pages 97–104, May 1997.

[117] T. Miller. Hidden-surfaces: Combining bsp trees with graph-based algorithms. Technical Report CS-96-15, Department of Computer Graphics, Brown University, Apr. 1996.

[118] H. P. Moravec. 3D graphics and the wave theory. *Computer Graphics*, 15(3):289–296, Aug. 1981.

[119] P. Morer, A. M. Garcia-Alonso, and J. Flaquer. Optimization of a priority list algorithm for 3-D rendering of buildings. *Computer Graphics Forum*, 14(4):217–227, Oct. 1995.

[120] K. Mulmuley. An efficient algorithm for hidden surface removal. *Computer Graphics*, 23(3):379–388, July 1989.

[121] T. M. Murali and T. A. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 155–162. ACM SIGGRAPH, Apr. 1997. ISBN 0-89791-884-3.

[122] B. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Computer–Aided Design*, pages 250–252, 1990.

[123] B. Naylor. Constructing good partition trees. In *Proceedings of Graphics Interface '93*, pages 181–191, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[124] B. Naylor, J. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *Computer Graphics*, 24(4):115–124, Aug. 1990.

[125] B. F. Naylor. Interactive solid geometry via partitioning trees. In *Proceedings of Graphics Interface '92*, pages 11–18, May 1992.

[126] B. F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, May 1992.

[127] K. Nechvile and J. Sochor. Form-factor evaluation with regional BSP trees. In *Winter School of Computer Graphics 1996*, Feb. 1996. held at University of West Bohemia, Plzen, Czech Republic, 12-16 February 1996.

[128] K. Nechvíle and P. Tobola. Local approach to dynamic visibility in a plane. In *Proceedings of WSCG'99, the 7th International Conference in Central Europe on Computer Graphics and Visualization '99*, pages 202–208, 1999.

[129] T. Nishita and E. Nakamae. Continuous tone representation of 3-D objects taking account of shadows and interreflection. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):23–30, July 1985.

[130] R. Orti, S. Riviere, F. Durand, and C. Puech. Using the Visibility Complex for Radiosity Computation. In *Lecture Notes in Computer Science (Applied Computational Geometry: Towards Geometric Engineering)*, volume 1148, pages 177–190, Berlin, Germany, May 1996. Springer-Verlag.

[131] M. Pellegrini. Ray shooting and lines in space. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 32, pages 599–614. CRC Press LLC, Boca Raton, FL, 1997.

[132] Q. Peng, Y. Zhu, and Y. Liang. A fast ray tracing algorithm using space indexing techniques. In G. Marechal, editor, *Eurographics '87*, pages 11–23. North-Holland, Aug. 1987.

[133] H. Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3D scenes. In *Proceedings of Graphics Interface '93*, pages 166–173, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[134] H. Plantinga, C. R. Dyer, and W. B. Seales. Real-time hidden-line elimination for a rotating polyhedral scene using the aspect representation. In *Proceedings of Graphics Interface '90*, pages 9–16, May 1990.

[135] M. Pocchiola and G. Vegter. The visibility complex. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 328–337, 1993.

[136] F. P. Preparata and M. I. Shamos. *Computational Geometry : An Introduction*. Springer-Verlag, 1985.

[137] A. Rajkumar, B. Naylor, F. Feisullin, and L. Rogers. Predicting rf coverage in large environments using ray–beam tracing and partitioning tree represented geometry. *Wireless Netwoks*, 2:143–154, 1996.

[138] S. Rivière. Topologically sweeping the visibility complex of polygonal scenes. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C36–C37, 1995.

[139] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 421–423, 1997.

[140] S. Rivière. Walking in the visibility complex with applications to visibility polygons and dynamic visibility. In *Proc. 9th Canad. Conf. Comput. Geom.*, pages 147–152, 1997.

[141] J. Rohlf and J. Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In *Proceedings of SIGGRAPH '94*, pages 381–395, July 1994.

[142] S. S and K. H. Directional safe zones & dual extent algorithms for efficient grid traversal. In *Graphics Interface 97*, page to appear, 1997. University of Colorado.

[143] A. Sagadic. An algorithm for invariant polygon visibility orderings. Downloaded from the WWW.

[144] H. Samet. Implementing ray tracing with octrees and neighbor finding. *Computers and Graphics*, 13(4):445–60, 1989. includes code.

[145] H. J. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods.* Addison–Wesley, Redding, MA, 1989.

[146] G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 151–162, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[147] G. Schaufler and W. Sturzlinger. A three-dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):C227–C235, C471–C472, Sept. 1996.

[148] A. Schilling. Toward real-time photorealistic rendering: Challenges and solutions. In S. Molnar and B.-O. Schneider, editors, *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 7–16, New York City, NY, Aug. 1997. ACM SIGGRAPH / Eurographics, ACM Press. ISBN 0-89791-961-0.

[149] D. Schmalstieg. A survey of advanced interactive 3-D graphics techniques. Research paper, Institute of Computer Graphics, Vienna University of Technology, 1997.

[150] P. Schroeder and P. Hanrahan. On the form factor between two polygons. *Computer Graphics*, 27(Annual Conference Series):163–164, 1993.

[151] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[152] M. Sharir and M. H. Overmars. A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11(1):1–11, Jan. 1992.

[153] I. Shimshoni and J. Ponce. Finite resolution aspect graphs of polyhedral objects. In *WQV93*, pages 140–150, 1993.

[154] P. Shirley, B. Wade, P. M. Hubbard, D. Zareski, B. Walter, and D. P. Greenberg. Global Illumination via Density Estimation. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 219–230, New York, NY, 1995. Springer-Verlag.

[155] F. Sillion and G. Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 145–152. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[156] F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 335–344, July 1989.

[157] C. T. Silva and J. S. B. Mitchell. The lazy sweep ray casting algorithm for rendering irregular grids. *IEEE Transaction on Visualization and Computer Graphics*, 3(2):142–157, Apr. 1997.

[158] G. Simiakakis. *Accelerating RayTracing with Directional Subdivision and Parallel Processing.* PhD thesis, University of East Anglia, october 1995. available at ftp://ftp.sys.uea.ac.uk/pub/ah/G.Simiakakis_PhD.ps.

[159] G. Simiakakis and A. M. Day. Five-dimensional adaptive subdivision for ray tracing. *Computer Graphics Forum*, 13(2):133–140, June 1994.

[160] S. S. Skiena. *The Algorithm Design Manual.* Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1998.

[161] M. Slater. A comparison of three shadow volume algorithms. *The Visual Computer*, 9(1):25–38, 1992.

[162] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic caching scheme. In *ACM Virtual Reality Software and Technology VRST'97*, pages ??–??, Lausanne, Switzerland, Sept. 1997.

[163] P. Slusallek. Photo-realistic rendering – recent trends and developments. In *EUROGRAPHICS '97 State-of-the-Art-Report.* Eurographics Association, 1997.

[164] J. Snyder and J. Lengyel. Visibility sorting and compositing without splitting for image layer decomposition. In M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 219–230. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[165] E. Sojka. Aspect graphs of three dimensional scenes. In *Winter School of Computer Graphics 1995*, pages 289–299, Feb. 1995. held at University of West Bohemia, Plzen, Czech Republic, 14-18 February 1995.

[166] C. Soler and F. Sillion. Accurate error bounds for multi-resolution visibility. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 133–142, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4.

[167] C. Soler and F. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '98 (Orlando, FL), pages ??–?? ACM SIGGRAPH, New York, July 1998.

[168] M. Stamminger, W. Nitsch, and P. Slusallek. Isotropic clustering for hierarchical radiosity – implementation and experiences. In *Proc. Fifth International Conference in Central Europe on Computer Graphics and Visualization '97*, 1997.

[169] M. Stamminger, P. Slusallek, and H.-P. Seidel. Bounded clustering – finding good bounds on clustered light transport. Technical Report 2, Universität Erlangen-Nürnberg, 1997. submitted for publication.

[170] A. J. Stewart. Hierarchical visibility in terrains. In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 217–228, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[171] A. J. Stewart and S. Ghali. An Output Sensitive Algorithm for the Computation of Shadow Boundaries. In *Canadian Conference on Computational Geometry*, pages 291–296, Aug. 1993.

[172] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 231–238. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[173] A. J. Stewart and T. Karkanis. Computing the approximate visibility map, with applications to form factors and discontinuity meshing. In *Proceedings of the Ninth Eurographics Workshop on Rendering*, pages 57–68, 1998.

[174] K. Subramanian and D. Fussel. Factors affecting performance of ray tracing hierarchies. Technical Report Tx 78712, The University of Texas at Austin, July 1990.

[175] K. Subramanian and D. Fussel. A search structure based on k-d trees for efficient ray tracing. Technical Report Tx 78712-1188, The University of Texas at Austin, 1992.

[176] K. R. Subramanian and D. S. Fussell. Automatic termination criteria for ray tracing hierarchies. In *Proceedings of Graphics Interface '91*, pages 93–100, June 1991.

[177] K. R. Subramanian and B. F. Naylor. Converting Discrete Images to Partitioning Trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):273–?, July 1997.

[178] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):C249–C258, Sept. 1996.

[179] K. Sung and P. Shirley. Ray tracing with the BSP tree. In D. Kirk, editor, *Graphics Gems III*, pages 271–274. Academic Press, San Diego, 1992. includes code.

[180] L. Szirmay-Kalos. Global ray-bundle tracing. Technical Report, TR-186-2-98-18, Vienna University of Technology, Vienna, 1998.

[181] L. Szirmay-Kalos and G. Marton. On the limitations of worst–case optimal ray shooting algorithms. In *Winter School of Computer Graphics 1997*, pages 562–571, Feb. 1997. held at University of West Bohemia, Plzen, Czech Republic, 14-18 February 1997.

[182] L. Szirmay-Kalos and W. Purgathofer. Global ray-bundle tracing with hardware acceleration. In *Ninth Eurographics Workshop on Rendering*, Vienna, Austria, June 1998.

[183] T. Tanaka and T. Takahashi. Shading with area light sources. In *Eurographics '91*, pages 235–246, 535–537. North-Holland, Sept. 1991.

[184] M. Teichmann and S. Teller. A weak visibility algorithm with an application to an interactive walkthrough. Downloaded from the WWW.

[185] S. Teller and J. Alex. Frustum casting for progressive, interactive rendering. Technical Report MIT LCS TR–740, MIT, January 1998.

[186] S. Teller, K. Bala, and J. Dorsey. Conservative interpolants for ray tracing. Technical Report MIT/LCS/TM-549, MIT, Apr. 1996.

[187] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan. Partitioning and ordering large radiosity computations. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 443–450. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[188] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. In *Proceedings of SIGGRAPH '93*, pages 239–246, 1993.

[189] S. Teller and M. Hohmeyer. Computing the lines piercing four lines. Technical Report UCB/CSD 93/161, UC Berkeley, Apr. 1993.

[190] S. J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, 26(2):139–148, July 1992.

[191] S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992. Also available as Technical Report UCB//CSD-92-708.

[192] S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, CS Division, UC Berkeley, Oct. 1992. Tech. Report UCB/CSD-92-708.

[193] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of SIGGRAPH '91*, pages 61–69, July 1991.

[194] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of SIGGRAPH '87*, volume 21, pages 153–162, July 1987.

[195] R. F. Tobler, A. Wilkie, M. Feda, and W. Purgathofer. A hierarchical subdivision algorithm for stochastic radiosity methods. In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 193–204, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[196] P. Tobola and K. Nechvíle. Linear size bsp trees for scenes with low directional density. In *Proceedings of WSCG'99, the 7th International Conference in Central Europe on Computer Graphics and Visualization '99*, pages 297–304, 1999.

[197] E. Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 507–518. North-Holland, Sept. 1990.

[198] M. van der Zwaan, E. Reinhard, and F. W. Jansen. Pyramid clipping for efficient ray traversal. In *Proceedings of the Sixth Eurographics Rendering Workshop*, Dublin, Ireland, 1995.

[199] E. Veach and L. Guibas. Bidirectional Estimators for Light Transport. In *Fifth Eurographics Workshop on Rendering*, pages 147–162, Darmstadt, Germany, June 1994.

[200] E. Veach and L. J. Guibas. Metropolis light transport. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 65–76. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[201] J. R. Wallace, K. A. Elmquist, and E. A. Haines. A ray tracing algorithm for progressive radiosity. In J. Lane, editor, *SIGGRAPH '89 Proceedings*, volume 23, pages 315–324, July 1989.

[202] Y. Wang, H. Bao, and Q. Peng. Accelerated walkthroughs of virtual environments based on visibility preprocessing and simplification. In *Eurographics '98*, volume 17, pages 187–194, 1998.

[203] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, Jan. 1984.

[204] R. Westermann and T. Ertl. The VSBUFFER: Visibility ordering of unstructured volume primitives by polygon drawing. In R. Yagel and H. Hagen, editors, *IEEE Visualization ́97*, pages 35–42. IEEE, Nov. 1997.

[205] M. Wimmer, M. Giegl, and D. Schmalstieg. Fast walkthroughs with image caches and ray casting. Technical Report TR-186-2-98-30, Institut für Computergraphik, TU Wien, December 1998.

[206] P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. Technical Report TR-186-2-99-03, Institut für Computergraphik, TU Wien, January 1999.

[207] A. Woo and J. Amanatides. Voxel occlusion testing: A shadow determination accelerator for ray tracing. In *Proceedings of Graphics Interface '90*, pages 213–220, May 1990.

[208] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, Nov. 1990.

[209] R. Yagel and W. Ray. Visibility computation for efficient walkthrough of complex environments. *Presence: Teleoperators and Virtual Environments*, 5(1), 1995.

[210] F. Yamaguchi and M. Niizeki. Some basic geometric test conditions in terms of pluecker coordinates and pluecker coefficients. *Visual Comput.*, 13(1):29–41, 1997.

[211] H. Zhang and K. E. Hoff III. Fast backface culling using normal masks. In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 103–106. ACM SIGGRAPH, Apr. 1997. ISBN 0-89791-884-3.

[212] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

9 Dissertation thesis

Title: Hierarchical Techniques for Output Sensitive Visibility Determination

Abstract

We are going to study visibility problems in three dimensions belonging to the VAL, VFP, and VFR classes. For each problem an output–sensitive solution based on hierarchical data structures will be proposed. We will analyse the methods theoretically and experimentally.

Further, we are going to study the relationships between the proposed methods and provide a unified framework. The ultimate goal is to accelerate visibility algorithms using results of algorithms belonging the higher–dimensional class of visibility problems. In particular we want to concentrate on the VFR problems, where no robust, output–sensitive and simple–to–implement solution has been proposed.

Keywords

Visibility, occlusion, ray casting, hidden surface removal, real–time rendering, discontinuity meshing, potentially visible sets, conservative algorithms, hierarchical algorithms, output sensitive algorithms, kD–tree, BSP tree.

10 Publications of the author

[1] J. Bittner and V. Havran. Exploiting temporal and spatial coherence in hierarchical visibility algorithms. Submitted to the Eurographics'99 conference in Milano, January 1999.

[2] V. Havran, J. Bittner, and J. Žára. Constructing rectilinear BSP trees for preferred ray sets. Poster presentation at the WSCG'99 conference, Pilsen, February 1999.

[3] J. Bittner, V. Havran, and T. Kopal. Visibility computations in large 3D scenes. Poster presentation at the Workshop'99, CTU Prague, January 1999.

[4] V. Havran, T. Kopal, J. Bittner, and J. Žára. Fast robust BSP tree traversal algorithm for ray tracing. *Journal of Graphics Tools*, December 1998.

[5] J. Bittner, V. Havran, and P. Slavík. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 207–219, Hannover, Germany, June 1998. IEEE, New York. Also available as http://www.cgg.cvut.cz/~bittner/paper-copy.ps.gz.

[6] V. Havran, J. Bittner, and J. Žára. Ray tracing with rope trees. In *Proceedings of 13th Spring Conference on Computer Graphics*, pages 130–139, Budmerice, 1998. Also available as http://www.cgg.cvut.cz/~bittner/sccg98.ps.gz.

[7] J. Bittner. Conservative hierarchical visibility with occlusion trees. In *Proceedings of Poster'98*, pages 2 pages, IC 03, Prague, Czech Republic, May 1998. CTU–FEE.

[8] T. Kopal, V. Havran, J. Bittner, and J. Žára. Directionality in ray tracing. In *Proceedings of Poster'98*, pages 2 pages, IC 24, Prague, Czech Republic, May 1998. CTU–FEE. 2nd International Student Conference on Electrical Engineering.

[9] J. Bittner. Hierarchical visibility algorithms. In A. Strejc, editor, *Proceedings of Workshop'98*, pages 195–196, Prague, Czech Republic, February 1998. CTU–FEE.

[10] J. Bittner. Global visibility computations. Master's thesis, Department of Computer Science and Engineering, Czech Technical University Prague, January 1997. Also available as http://www.cgg.cvut.cz/~bittner/masters.ps.gz.

[11] J. Bittner. Global visibility computations. In *Graphics and WEB '97*. HTW Dresden, Germany, 1997.

[12] J. Bittner. Global visibility. In *Proceedings of the CESCG '97, April 97, Bratislava and Vienna*, pages 105–111, 1997.

[13] M. Trefný and J. Bittner. Global visibility. In *Proceedings of The Fifth International Conference in Central Europe on Computer Graphics and Visualization '97*, volume IV, pages 691–694, February 1997.

[14] J. Bittner. Recnet - the speech recognition system. Technical Report TR 96–160, Technical University, Delft, June 1996.