UNIVERSITÉ DE LIMOGES

ÉCOLE DOCTORALE Sciences et Ingénieries pour l'Information

Année : 2012 Thèse N° 35-2012

Thèse

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE LIMOGES

Discipline : Informatique et Applications

présentée et soutenue par

Oana Livia APOSTU

le 25 Octobre 2012

Analytic visibility in Plücker space: From theory to practical applications

Thèse dirigée par Djamchid GHAZANFARPOUR,
Co-encadrée par Frédéric MORA

JURY :

Jean-Pierre JESSEL	Professeur, Université de Toulouse	Président
Kadi BOUATOUCH	Professeur, Université de Rennes 1	Rapporteur
Mateu SBERT	Professeur, University of Girona	Rapporteur
Djamchid GHAZANFARPOUR	Professeur, Université de Limoges	Examinateur
Frédéric MORA	Maître de conférences, Université de Limoges	Examinateur

"The truth is out there"
The X-Files

"I don't need sleep, I need answers.
I need to determine where,
in this swamp of unbalanced formulas,
squatted the toad of truth."
Shelodon Cooper

This thesis is dedicated to my parents, who filled my childhood with books and dreams.
And to Alex, who makes the sweetest cookies in the world.

Contents

List of Figures

List of Tables

Nomenclature

The following notations and assumptions are considered through out this thesis:

- A line refers to an oriented line, unless otherwise stated.
- A surface refers to a planar convex and oriented polygon, unless otherwise stated.
- A polygon refers to a planar convex and oriented polygon, unless otherwise stated.
- Although the term triangle is usually employed, all statements equally apply to planar convex polygons in general, unless otherwise stated.
- Since we are working with oriented primitives only, the terms *above* and *behind* are always defined with respect to the upper normal. Moreover, we consider that all the lines stabbing a polygon have a coherent orientation with respect to the polygon's orientation, and thus to its upper normal.
- $poly(S \rightarrow T)$ denotes the minimal polytope containing all the lines stabbing two convex polygons S and T, in this order, as described in Chapter 1, Section 1.4.14.
- $O(S,T)$ denotes all the occluders of two polygons S and T, as defined in Chapter 2.
- $O(S)$ denotes all the potential occluders of a polygon S, as defined in Chapter 3.
- xyz denotes a 3-dimensional point.
- $hemisphere(xyz,\delta)$ denotes the upper hemisphere, with respect to xyz's normal, centered at xyz and having a δ radius.

Introduction

IN image synthesis, determining visibility is a required step in many techniques. The visibility problems range from simple questions such as what is visible from a point, to more global issues describing the visibility between objects. For example, calculating the hard shadows due to a point light source is a classic point-to-point problem, since it requires to detect if the light is visible from the point to shade. On the other hand, the soft shadows generated by an area light source represent a point-to-surface problem. In this case, the visibility of the entire light source needs to be determined from the point to shade. A similar situation is that of an observer or a camera. In this case, the visibility of the surrounding environment must be described from the point representing the camera. More complex visibility issues are raised in the context of global illumination. In this case, the problem is extended to the visibility relation which exists between each polygon and its entire environment.

An important point in solving visibility is the nature of the solution. This has an impact both on the complexity of the calculations, as well as on the final results. Discrete approximations and sampling methods translate the various visibility problems to point-to-point visibility queries, which can be easily solved using ray tracing. Thus, they are robust and simple to implement. However, they cannot solve all visibility problems. The classic example is that of the mutual visibility or invisibility of two objects, which cannot be proven using a ray tracing approach. Also, their results are subject to noise. On the other hand, analytic methods provide a continuous and exact representation of visibility, which allows answering a wider range of problems. Moreover, the results are accurate and noise free. However, analytic methods are more complex to implement.

The efficiency of the visibility solution often relies on exploiting the visual coherence which exists between neighbor view rays. The notion of coherence can be translated by the fact that two rays which are "close" are more likely to intersect the same objects, since they travel close paths in the scene. In the context of discrete visibility, *packet tracing* [WSBW01, RSH05, WIK+06, Wal07, WK06] was introduced in order to take advantage of the coherence between the rays originating from the camera. The idea was to group rays together into packets, so the travel cost could be amortized over the entire packet. An analytic solution equally exists. More exactly, early research in image synthesis attempted to replace the conventional ray with volumetric structures, generally called *beams* [Ama84, HH84, STN87, GH98].

Both the discrete and the analytic methods exploit the coherence from a single view point. However, it is obvious that two close points also exhibit a strong visual coherence, since they share a similar view of the surrounding environment. For example, in the context of soft shadows, two neighbor points located on the same surface will most likely receive the same amount of light, since they have analogous views of the light source. Exploiting the visual coherence which exists between the view rays originating from a surface is a from-polygon visibility problem. This subject has received less attention, because of the its inherent complexity.

From-polygon visibility is by nature a four-dimensional problem [Dur99]. While it can be theoretically described in an elegant manner [Pel91, Pel04], few practical solutions are available, and they all lack robustness and efficiency.

In this thesis, we aim to fill this existing void between theory and practice by providing an analytic approach which exploits the visual coherence from a polygon in order to optimize from-point visibility queries, while ensuring high quality results.

Our contributions can be summarized as follows:

- First of all, we propose a method which solves from-polygon occlusion and which is used in the context of soft shadow generation. Our algorithm encodes an analytic representation of the occlusion information between an area light source and the polygon containing the points to shade. Then, for each one of these points to shade, we extract the exact visible parts of the light source and compute its analytic and exact soft shadow value. Our tests demonstrate that this solution is robust and efficient, and yields accurate and high quality results.
- Next, we build on our first algorithm and propose a method which takes into account an additional depth information and therefore computes from-polygon visibility. The applicative context is the calculation of ambient occlusion effects. Our second algorithm retains all the advantages of our first implementation. For each visible polygon from the viewpoint, we encode its exact visibility of the surrounding environment. Then, for each one of its points, we extract the visibility information and use it together with an analytic formulation to compute noise free and high quality ambient occlusion.

Overview

The first chapter starts with the definition of the necessary geometric preliminaries, and explains the basis of Plücker space, which is a necessary tool in the context of from-polygon visibility. We then describe the theoretical framework representing the starting point of our work. Next, we briefly present the methods which have proposed analytic solutions to from-point visibility problems, followed by a detailed description of from-polygon visibility techniques. We conclude the first chapter with a summary of the drawbacks associated with the current methods, and we announce the aims and motivations of this work with respect to the presented context.

The second chapter details our from-polygon occlusion method, as well as its application to the generation of soft shadows. The first section presents a brief overview of high-quality and analytic soft shadow generation methods. Then, we describe our occlusion algorithm from a theoretical point of view. Next, we present the details of our implementation and the soft shadows framework. The last two sections present the results we obtain and a global analysis of the method. This chapter corresponds to the first contribution previously presented.

The third chapter is dedicated to the presentation of our from-polygon visibility method, and to its application to the calculation of ambient occlusion. First of all, we present the ambient occlusion illumination model, and detail the methods which achieve high quality results. We then explain the main differences between this new approach and our from-polygon occlusion algorithm. Next, we present our visibility algorithm from a theoretical point of view, followed by the implementation details and a description of the ambient occlusion framework. Finally, we detail and analyze the results we obtained. This chapter corresponds to the second contribution previously presented.

Chapter 1 :

Analytic Visibility

THIS chapter provides a study of analytic visibility methods. Attempting to define visibility or providing a survey of existing visibility problems and algorithms is outside the scope of the work. In his PhD thesis, Durand [Dur99] provides a global and comprehensive overview on visibility problems in various research fields. We can also recall here the classic study on hidden-surface algorithms proposed by Sutherland *et al.* [SSS74], and the classification of visibility in computer graphics provided by Bittner and Wonka [BW03].

The definition of analytic visibility concerns the techniques which define visibility using a set of mathematical equations. These methods deal with continuous representations of lines and surfaces. Contrary to discrete solutions, the calculations involved are of higher complexity and the current implementations suffer from several drawbacks.

First of all, Section 1.1 defines some required geometric notions and explains the basis of Plücker space, which allows an elegant parametrization of real lines. In the context of from-polygon visibility, this parametrization represents a necessary tool. Next, Section 1.2 details a theoretical framework based on the Plücker parametrization. This framework provides a classification of lines according to the geometry they intersect. Thus, it can be used in the context of analytic visibility. Sections 1.3 and 1.4 analyze different methods which have attempted to provide analytical solutions to from-point and from-polygon visibility problems, respectively. All these techniques are then synthesized in a table contained in Section 1.5. Finally, Section 1.6 concludes by summarizing the drawbacks of the existing solutions. In this context, we define our motivation and our aims.

1.1 Geometric Preliminaries

This section summarizes some geometric concepts which are essential to this study. First of all, we review the notions of *polyhedron* and *polytope*, together with some adjacent definitions. Next, we present a few notions of projective geometry, which allow us to define the Plücker space, an oriented projective space providing an elegant parametrization of 3-dimensional lines.

1.1.1 Polyhedrons and Polytopes

The words *polyhedron* and *polytope* have various, and sometimes conflicting, definitions. A naive characterization defines a polytope as a geometric object with flat sides and which exists in any general number of dimensions [wik12c]. In order to avoid all ambiguities, we are summarizing here a list of definitions [HRGZ97].

We present two equivalent definitions for a polyhedron: algebraic (Definition 1) and a more intuitive one (Definition 2).

Definition 1. *A d-dimensional convex polyhedron is the set of points $x \in \mathbb{R}^d$ which are the solutions to a system of linear inequalities:*

$$mx \leq b$$

Where $m \in \mathbb{R}^{n \times d}$ is a real $n \times d$ matrix and $b \in \mathbb{R}^n$ is a real n vector.

Definition 2. *A d-dimensional convex polyhedron is an d-dimensional object having a boundary composed of a finite collection of polyhedral facets of dimension $d-1$.*

Definition 3. *A polytope is a bounded convex polyhedron.*

There are two distinct ways to represent polytopes: using a *half-space representation* or a *vertex representation.*

Definition 4. *H-polytope (Half-space representation, H-representation): a set of n half-spaces whose intersection in $\mathbb{R}^d$ gives the polytope.*

This characterization can be obtained from Definition 1: Let $m_i, 1 \leq i \leq n$ be the lines of matrix m, and b_i the elements of vector b. $m_i x - b_i = 0$ is the equation of a hyperplane, H_i. Thus, the set of solutions to the linear system in Definition 1 is the set of points $x \in \mathbb{R}^d$:

$$x \in \bigcap_{i=1}^{n} H_i^-$$

Definition 5. *V-polytope (Vertex representation, V-representation): the convex hull of a finite set $V = v^1, ..., v^k$ of points in $\mathbb{R}^d$:*

$$conv(V) = \{\sum_{i=1}^{k} \lambda_i x_i \mid \lambda \geq 0, \sum_{i=1}^{k} \lambda_i = 1\}$$

V is the set of the polytope's vertices. Figure 1.1 gives an illustration of the half-space and vertex representation of a $\mathbb{R}^3$ polytope.

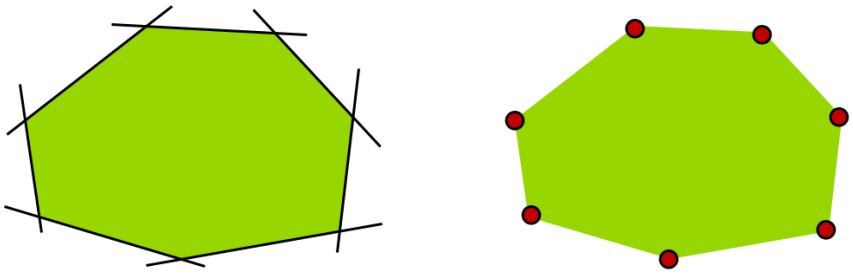


Figure 1.1: The Halfspace representation (left) and Vertex representation (right) of a $\mathbb{R}^3$ polytope.

Theorem 1. *The definitions of V-polytopes and of H-polytopes are equivalent. That is, every V-polytope has a description by a finite system of inequalities, and every H-polytope can be obtained as the convex hull of a finite set of points (its vertices).*

The *vertex enumeration problem* concerns the determination of an object's V-representation (its vertices) given the set of linear equations describing its H-representation. On the other hand, the construction of the H-representation is known as the *facet enumeration problem*. This is actually a particular case of the classical convex hull determination problem.

The dimension of the V-representation gives the dimension of the polytope.

Notation. *A d-polytope denotes a d-dimensional polytope.*

1.1.2 Oriented Projective Space

The need for a different system

Euclidean geometry is consistent with intuition: intersecting lines define angles, objects have sides with fixed lengths, parallel lines never meet. However, from some points of view, Euclidean geometry is either incomplete or not adapted to deal with certain problems.

A conclusive example is the case of railroad tracks. They represent parallel lines situated in the same plane, therefore they never intersect. However, for an observer, the two tracks intersect in a vanishing point on the horizon. Moreover, the same observer will perceive a square box as having angles smaller or larger than 90° and sides of different lengths. This *perspective view* is consistent with the behavior of a camera and is therefore very common in various areas of computer graphics and computer vision.

A second example which illustrates the limitations of Euclidean geometry is the case of exceptions. If we consider two lines in a plane, Euclidean geometry states that these lines intersect, *with the exception of* parallel lines.

Projective spaces have the advantage of providing definitions and properties which are more

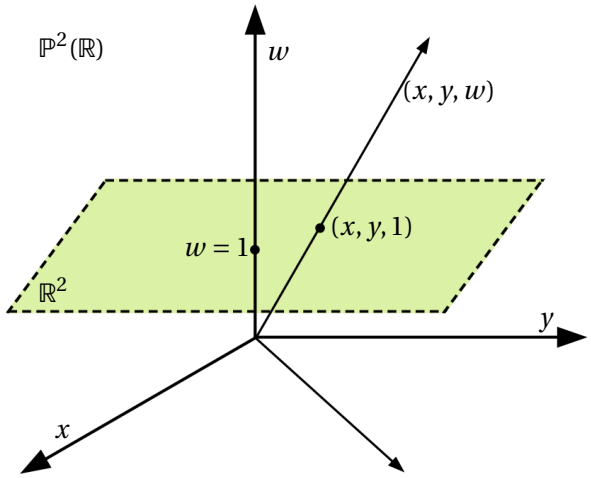


Figure 1.2: The projective plane $\mathbb{P}^2(\mathbb{R})$. Each point in $\mathbb{R}^2$ corresponds to a point in $\mathbb{P}^2(\mathbb{R})$. Geometrically, this can be seen as a vector in $\mathbb{R}^3$. If we consider the projection plane $w = 1$, the points belonging to the equivalence class $< X, Y, 0 >$ have no corresponding point in $\mathbb{R}^2$. These *ideal points* only exist in the projective plane.

consistent and without exceptions. Moreover, they are compatible with the perspective view of a camera.

This section discuses some aspects of projective geometry, which are required for a better understanding of this work. An extensive presentation can be found in [Sto91].

The Classic Projective Geometry

A n-dimensional projective space can be defined as follows:

Definition 6. *Let $\mathbb{K}$ be a field. The projective space of dimension n over $\mathbb{K}$ (noted $\mathbb{P}^n(\mathbb{K})$) is the set of lines passing through the origin in $\mathbb{K}^{n+1}$. More formally, let $\sim$ be an equivalence relation on the set of non-zero points $\mathbb{K}^* = \mathbb{K}^{n+1} \setminus \{0\}$, defined as follows:*

$$x \sim \lambda x, \quad x \in \mathbb{K}^{n+1} \setminus \{0\}, \quad \lambda \in \mathbb{K} \setminus \{0\}$$

$\mathbb{P}^n(\mathbb{K})$ represents the set of corresponding equivalence classes.

Let $x = (x_0, ..., x_{n-1}) \in \mathbb{K}^n$. In order to represent the same point in $\mathbb{P}^n(\mathbb{K})$, we use a system of coordinates called the *homogeneous coordinates*. x can be rewritten as $(x_0, ..., x_{n-1}, 1)$. This defines the equivalence class $< X_0, ..., X_n >$, where any point $(x_0, ..., x_n) \equiv (\lambda x_0, ..., \lambda x_n)$, $\forall \lambda \in \mathbb{K} \setminus \{0\}$. Note that $(x_0, ..., x_n) \in \mathbb{K}^{n+1}$. This equivalence class corresponds to a vector containing $(x_0, ..., x_n)$ in $\mathbb{K}^{n+1}$. Moreover, $\mathbb{P}^n(\mathbb{K})$ represents the set of all n-dimensional vector spaces over $\mathbb{K}^{n+1}$.

In order to better understand this definition, as well as the properties it implies, we give a succinct description of the *classic projective plane* $\mathbb{P}^2$.

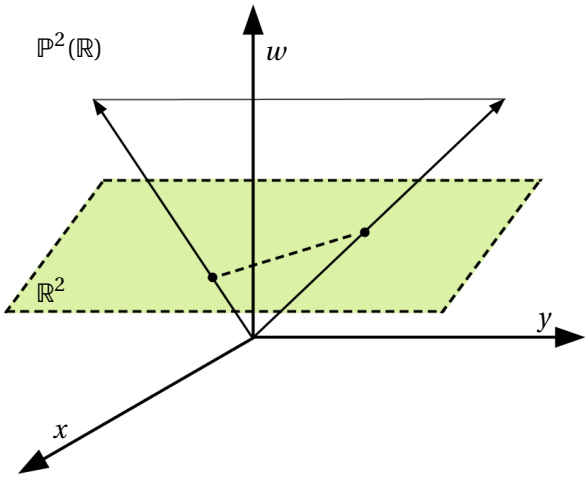


Figure 1.3: A line in $\mathbb{R}^2$ corresponds to a set of 2-dimensional vectors in $\mathbb{P}^2(\mathbb{R})$. Geometrically, this can be interpreted as a plane which contains the origin in $\mathbb{R}^3$.

The Projective Plane

Points. Let $(x, y) \in \mathbb{R}^2$ be a point in Euclidean plane. This corresponds to the equivalence class $< X, Y, W >$, where any point $(x, y, w) \equiv (\lambda x, \lambda y, \lambda w)$, $\forall \lambda \in \mathbb{R} \setminus \{0\}$. This is actually the real line $(\in \mathbb{R}^3)$ containing both the origin and the point (x, y, w) (the vector $(x, y, w) \in \mathbb{R}^3$).

To characterize the correspondence between $\mathbb{R}^2$ and the classic projective plane $\mathbb{P}^2(\mathbb{R})$, we use a *projection plane*. This can be any plane in $\mathbb{R}^3$, which does not contain the origin. Let $w = 1$ be the chosen projective plane. It can be easily demonstrated that there is a one-to-one correspondence between the points on this plane and $\mathbb{R}^2$. Indeed, any point $(x, y, 1)$ on the projection plane can be mapped to $(x, y) \in \mathbb{R}^2$. Moreover, each point $(x, y, 1)$ corresponds to a equivalence class $< X, Y, 1 > \in \mathbb{P}^2(\mathbb{R})$. Therefore, any point in $\mathbb{R}^2$ can be mapped to a point in $\mathbb{P}^2(\mathbb{R})$.

However, not all points $\in \mathbb{P}^2(\mathbb{R})$ can be mapped to $\mathbb{R}^2$. With respect to the chosen plane, the points corresponding to the equivalence class $< X, Y, 0 >$ cannot be represented in $\mathbb{R}^2$. These points are called *ideal points* or *points at infinity*. Figure 1.2 gives an illustration of the projective plane.

Lines. If a point in $\mathbb{R}^2$ corresponds to a point in $\mathbb{P}^2(\mathbb{R})$ (a vector in $(R)^3$), a line in $\mathbb{R}^2$ corresponds to a line in $\mathbb{P}^2(\mathbb{R})$ (a plane containing the origin in $\mathbb{R}^3$). The intersection of this plane with the projection plane $w = 1$ yields a set corresponding to a unique line in $\mathbb{R}^2$. Just like in the case of points, every line in $\mathbb{R}^2$ will correspond to a unique plane containing the origin in $\mathbb{R}^3$, and therefore to a line in $\mathbb{P}^2(\mathbb{R})$. And similarly, there is one line in $\mathbb{P}^2(\mathbb{R})$, called *ideal line* or *line at infinity*, which does not exist in $\mathbb{R}^2$. Moreover, all the ideal points lie on the ideal line. Figure 1.3 illustrates the representation of lines in $\mathbb{P}^2(\mathbb{R})$.

From this point of view, the projective plane can be seen as the standard plane $\mathbb{R}^2$ augmented

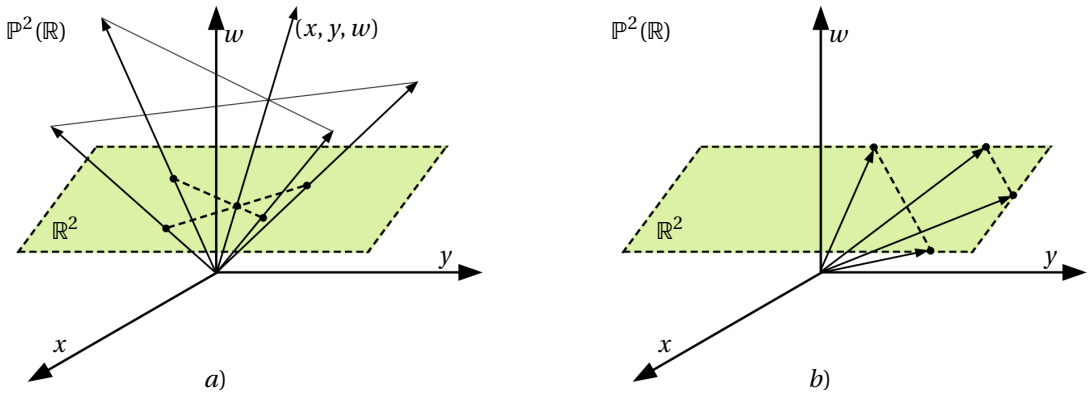


Figure 1.4: a) The intersection of two lines in $\mathbb{R}^2$ corresponds to the intersection of two lines in $\mathbb{P}^2(\mathbb{R})$ (ie. the intersection of two planes in $\mathbb{R}^3$, both containing the origin). b) Two parallel lines in $\mathbb{R}^2$ share an intersection in in $\mathbb{P}^2(\mathbb{R})$. This intersection corresponds to a point at infinity, $< x, y, 1 >$ in this case.

by a set of ideal points and an ideal line. Moreover, the ideal points and the ideal line are not distinguishable from regular points and lines, their definitions in the projective plane being identical. This is important because it allows uniform definitions and properties, without having to consider the particular cases taken into account by Euclidean geometry.

This simplification can be illustrated using the example of parallel lines. Two lines in $\mathbb{R}^2$ correspond to two lines in $\mathbb{P}^2(\mathbb{R})$, which can be interpreted as two planes going through the origin in $\mathbb{R}^3$. Since both these planes contain the origin, they are not parallel and thus they intersect. This intersection corresponds to an ideal point in $\mathbb{P}^2(\mathbb{R})$. Note that no assumption is made concerning the two initial lines. Therefore, since any line in $\mathbb{R}^2$ corresponds to a plane going through the origin in $\mathbb{R}^3$, and all these planes intersect in lines going through the origin, then all the lines in $\mathbb{P}^2(\mathbb{R})$ intersect in a point in $\mathbb{P}^2(\mathbb{R})$. Figure 1.4 offers a geometric illustration of arbitrary lines and their intersection in $\mathbb{P}^2(\mathbb{R})$.

Duality

The principle of duality represents an interesting and useful property of projective geometry. Its definition can be formulated as follows :

Definition 7. *Any theorem or definition of projective geometry has a dual theorem or definition, in which the terms "points" and "hyperplanes" have been interchanged.*

In a n-dimensional projective space, this can be demonstrated as follows: Let a and b be two points in $\mathbb{P}^n(\mathbb{K})$. Their homogeneous coordinates are $(a_0, ..., a_n)$ and $(b_0, ..., b_n)$, respectively. If the points satisfy the equation : $a_0b_0 + ... + a_nb_n = 0$, then the following statements are both true:

- The point a is contained in the hyperplane $b_0x_0 + ... + b_nx_n = 0$.
- The point b is contained in the hyperplane $a_0x_0 + ... + a_nx_n = 0$.

As with the previous definition, we illustrate this using the projective plane $\mathbb{P}^2(\mathbb{R})$.

Let (x_0, y_0) be a point in $\mathbb{R}^2$. $< X, Y, W >$ is the equivalence class corresponding to the mapping of this point to $\mathbb{P}^2(\mathbb{R})$.

Let $ax + by + c = 0$ be a line in $\mathbb{R}^2$. This line corresponds to the vector (a, b, c) in $\mathbb{R}^3$ (a line passing through the origin in $\mathbb{R}^3$). Thus, it can be mapped to a unique point $< a, b, c > \in \mathbb{P}^2(\mathbb{R})$.

If the point is on the line, then $ax_0 + by_0 + c = 0$, which is equivalent to $aX + bY + cW = 0$. This can be rewritten as $p \times l = l \times p = 0$, where $p =< X, Y, W > \in \mathbb{P}^2(\mathbb{R})$ and $l =< a, b, c > \in \mathbb{P}^2(\mathbb{R})$.

Therefore, in the projective plane $\mathbb{P}^2(\mathbb{R})$, saying that the point $< X, Y, W >$ is on the line $< a, b, c >$ is equivalent to stating that the point $< a, b, c >$ lies on the line $< X, Y, W >$.

Similarly, the dual statement of "Two points are concurrent" is "Two lines are incident". A demonstration, as well as additional dual properties, can be found in [Bir98].

The duality principle has several advantages. First of all, a demonstration in projective space is simultaneously a proof to two different theorems. Moreover, it is possible to use a more intuitive solution to a problem, in order to solve the problem's dual statement.

Oriented Projective Geometry

Geometric primitives are not oriented in the classic projective geometry. This is a major drawback, since many problems involve oriented lines or planes. In [Sto87] Jorge Stolfi discusses the problems which arise from the lack or orientation in classic projective geometry and describes how these inconveniences are solved by using *oriented projective geometry*. We only summarize here how the classic projective space can be modified in order to include orientation.

Roughly speaking, an oriented projective space is a double copy of a classic projective space, where each copy has a different orientation. This implies that $< +x_0, ..., +x_n >$ and $< -x_0, ..., -x_n >$ are treated as two different equivalence classes. For example, in $\mathbb{P}^2(\mathbb{R})$, each line of the classical projective plane is replaced by two lines which coincide, but are oppositely oriented. Therefore, they are distinct. By applying the principle of duality, we also obtain two oppositely oriented copies for each point.

1.1.3 Plücker Coordinates

Plücker space ($\mathbb{P}^5(\mathbb{R})$, or simply $\mathbb{P}^5$) is a 5-dimensional oriented projective space, which provides an efficient representation of lines. Plücker coordinates are a special case of *Grassmann coordinates*, which can be used to parametrize a k-dimensional affine sub-space embedded in an n-dimensional space as a point in a projective $\binom{n+1}{k+1} - 1$ dimensional space. In the context of parameterizing lines ($k = 1$) in $\mathbb{R}^3$ ($n = 3$), this yields the Plücker space.

For the rest of this work, the terms point, line and plane will refer to geometric elements in $\mathbb{R}^3$,

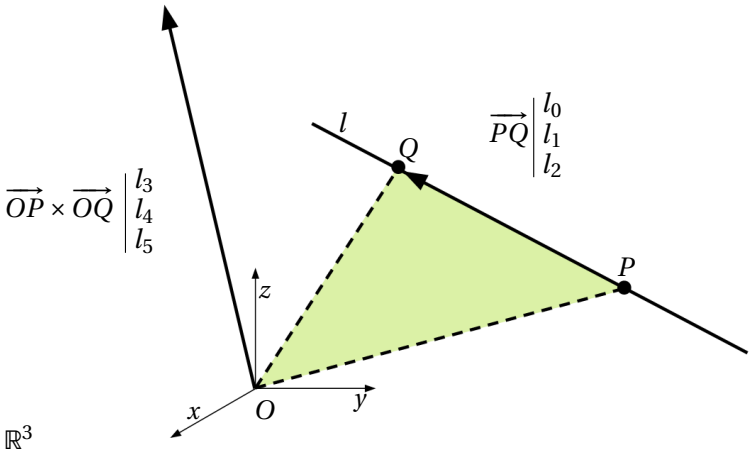


Figure 1.5: Geometrical interpretation of the Plücker coordinates: $(l_0, l_1, l_2, l_3, l_4, l_5)$ is the Plücker mapping of the line $l \in \mathbb{R}^3$. The first three coordinates (l_0, l_1, l_2) correspond to the direction of the line, while the last three (l_3, l_4, l_5) encode its position. (l_3, l_4, l_5) represents a vector which is perpendicular on the plane containing the line and going through the origin (OPQ).

unless otherwise stated.

Let $p = (p_x, p_y, p_z)$ and $q = (q_x, q_y, q_z)$ be two distinct points which define an oriented line l. In Plücker space, this line is represented by a set of six coefficients, called the Plücker coordinates, π_l, of the line l.

$$\pi_l = (l_0, l_1, l_2, l_3, l_4, l_5)$$

where

$$\begin{aligned} l_0 &= q_x - p_x \quad & l_3 &= q_z p_y - q_y p_z \\ l_1 &= q_y - p_y \quad & l_4 &= q_x p_z - q_z p_x \\ l_2 &= q_z - p_z \quad & l_5 &= q_y p_x - q_x p_y \end{aligned}$$

Although this parametrization is far from intuitive, a geometric interpretation can be given. The first three coordinates correspond to the direction of the line, while the last three coordinates describe the position of the line in space. Figure 1.5 gives an illustration.

Any Plücker point is in bijection with its dual hyperplane:

$$\begin{aligned} &h_l(x) = l_3 x_0 + l_4 x_1 + l_5 x_2 + l_0 x_3 + l_1 x_4 + l_2 x_5 = 0 \\ &with\ x(x_0, x_1, x_2, x_3, x_4, x_5) \in \mathbb{P}^5. \end{aligned}$$

Therefore, there are two geometrically dual ways to describe an oriented real line in Plücker space: either as a point or as its dual hyperplane.

An important property of the Plücker space is that we can characterize the relative orientation

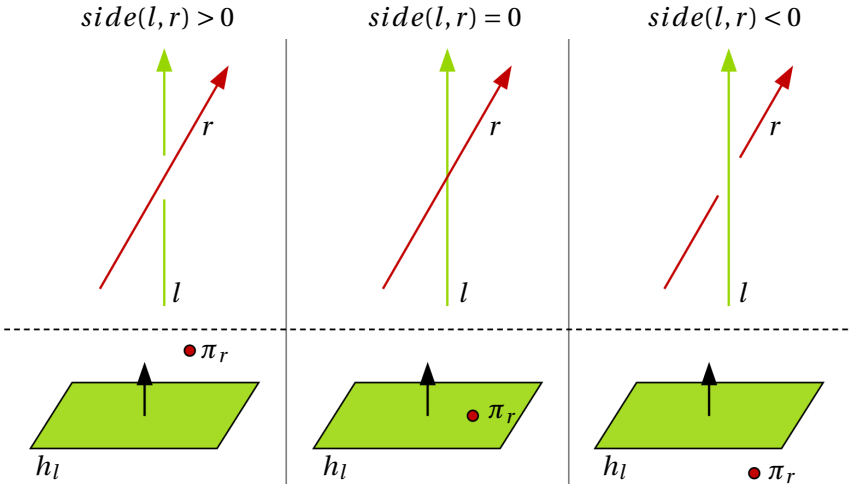


Figure 1.6: **Above**: The relative orientation of two lines is given by the sign of the side operator. **Below**: Since $side(l,r) = h_l(\pi_r)$, the side operator is equivalent to testing the position of the Plücker point of one line against the dual hyperplane of the other line.

of two lines. This can be done using the so-called *side operator*:

$$side(l,r) = l_3r_0 + l_4r_1 + l_5r_2 + l_0r_3 + l_1r_4 + l_2r_5$$

where l and r are two lines and $\pi_l(l_0,l_1,l_2,l_3,l_4,l_5)$ and $\pi_r(r_0,r_1,r_2,r_3,r_4,r_5)$ their Plücker coordinates. In particular, two lines are incident (or parallel) if their side operator equals zero. We can notice that $side(l,r) = h_l(\pi_r)$. From a geometric point of view, this translates to the fact that computing the relative position of two lines comes down to testing the position of the Plücker point of one line against the dual hyperplane of the second line (see Figure 1.6).

Although any line in $\mathbb{R}^3$ can be mapped to a point in $\mathbb{P}^5$, not all the points in $\mathbb{P}^5$ correspond to lines in $\mathbb{R}^3$. The set of real oriented lines corresponds in $\mathbb{P}^5$ to a particular four-dimensional quadric surface, known as the *Plücker hyper-surface* (also known as the *Grassmann manifold*, the *Klein quadric*, or the *Plücker quadric*). Formally, the *Plücker hyper-surface* can be defined as follows :

$$Q = \{x \in \mathbb{P}^5 | h_x(x) = 0\} \setminus \{0\}$$

Using the Plücker coordinates, the edges of polygons can be defined as directed lines, which correspond to hyperplanes in Plücker space. These hyperplanes define a 5-dimensional polyhedron. Therefore, the Plücker parametrization allows an easy characterization of continuous sets of lines using convex volumes.

1.2 Theoretical Framework

1.2.1 Ray Triangle Intersection Problem

A well-known application of the side operator is an easy and robust line-triangle intersection test. The solution is based on the following observation: An oriented line intersects a triangle if its relative orientation is consistent with respect to the lines spanning the triangle's edges. An illustration is given in Figure 1.7.

Let T be a triangle and l_0, l_1, l_2 the lines spanning its edges. If T is oriented, then l_0, l_1, l_2 have coherent orientations. These three lines correspond to three hyperplanes in $\mathbb{P}^5$, which split the Plücker space into cells. Any line intersecting the triangle will map to a Plücker point located in the same cell, which corresponds to the intersection of the three half spaces induced by the hyperplanes. Therefore, this cell contains all the lines (*i.e.* their Plücker points) stabbing the triangle, whereas the other cells contain all the lines missing the triangle. This provides an analytical representation of all the lines intersecting a triangle.

This property remains valid for any oriented convex polygon.

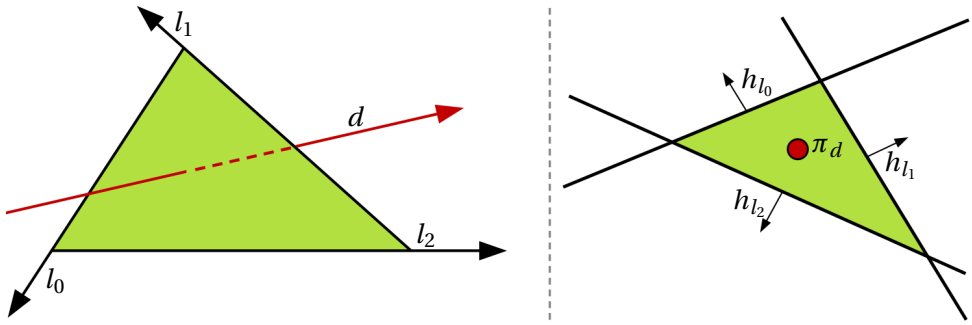


Figure 1.7: Left: a triangle in 3D space. l_0, l_1, l_2 are the lines spanning the triangle's edges. d is a line stabbing the triangle. **Right**: h_{l_0}, h_{l_1} and h_{l_2} are the Plücker hyperplanes mapped from l_0, l_1 and l_2 respectively. π_d is the Plücker point mapped from the stabbing line d. π_d has a consistent orientation in respect to l_0, l_1, l_2. From a geometrical point of view, π_d lies at the intersection of the half spaces induced by h_{l_0}, h_{l_1} and h_{l_2}. Thus, these 3 hyperplanes provide an analytical representation of all the lines stabbing the triangle.

1.2.2 Equivalence Classes of Oriented Lines

Description. The ray-triangle intersection property has been extended to any set of convex polygons by Pellegrini [Pel91, Pel04]. Let T_{set} be a set of oriented triangles (or convex polygons). The Plücker hyperplanes corresponding to the lines spanning the triangles' edges divide the Plücker space into cells. This builds an arrangement, having the following property: all the points belonging to the same cell have the same sign with respect to its bounding hyperplanes. Therefore, in Plücker space, all the lines (*i.e.* their Plucker points) belonging to the same cell intersect the same subset of triangles in T_{set}. The decomposition of Plücker space into cells allows to group lines together according to the

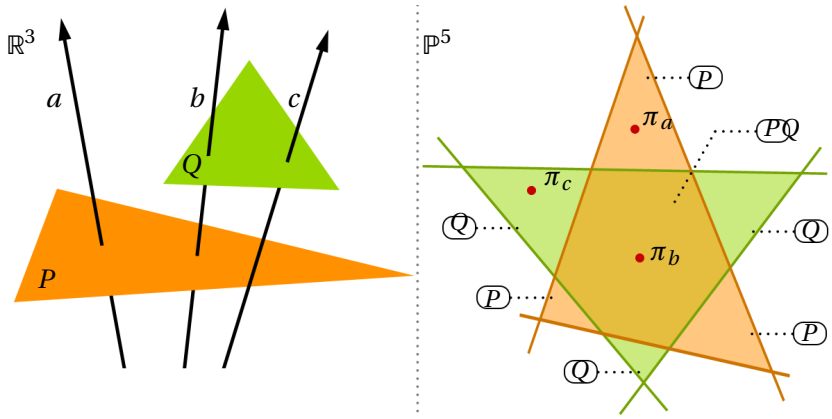


Figure 1.8: Equivalence classes according to Pellegrini. **Left**: Two triangles (P, Q) and three lines (a, b, c) in various configurations. **Right**: The arrangement of hyperplanes (illustrated by 6 2D lines) mapped from the triangles edges. Filled cells are set of lines intersecting at least one triangle. The intersected triangles are associated with the corresponding cells. π_a, π_b and π_c are the Plücker points mapped from a, b and c, respectively. They are located in the cells according to the triangles they stab. For example, π_b has a consistent orientation with respect to the 6 hyperplanes, since b intersects the two triangles.

subset of triangles they intersect. This defines an equivalence relation on lines. As a consequence, each cell corresponds to an equivalence class. To sum up, the Pellegrini approach allows an exact and analytical representation of all the sets of lines generated by a set of triangles. Figure 1.8 gives an illustration.

Theoretical complexity. An arrangement of n hyperplanes in $\mathbb{R}^d$ has a maximum of $O(n^d)$ cells [EOS86]. If this arrangement is restricted to a $(d-1)$-dimensional algebraic surface (*i.e.* the cells intersecting this surface), the maximum number of cells is $O(n^{d-1}\log n)$ [APS91]. In the above description, we are considering the arrangement of real lines (thus the Plücker points located on the Plücker quadric) in $\mathbb{P}^5$. Therefore, the equivalence classes generated by a set of real lines in Plücker space have a theoretical complexity of $O(n^4 \log n)$.

For a set of n disjoint triangles, Pellegrini [Pel91, Pel04] calculates a theoretical bound of $O(n^{4+\epsilon})$ for the considered arrangement.

Occlusion and Visibility. It is important to note that these equivalence classes correspond to an information of occlusions, rather than one of visibility. Pellegrini's definition provides the information of which lines intersect which subset of triangles, but not the order in which these triangles are intersected. In [Pel93], Pellegrini explains how the arrangement of lines can be used to answer the ray-shooting problem. More exactly, for each ray, the arrangement provides its equivalence class, and thus the triangles it intersects. The first intersection for the ray can be calculated using these triangles.

Therefore, speaking in terms of visibility, the definition of equivalence classes provides all the occlusion information for a view direction, without any particular order.

Other theoretical results. It is important to note that this framework has been extensively used to provide important theoretical results concerning oriented lines in space. Some of these problems involve grouping lines according to different types of geometry, answering various ray shooting problems, or dealing with moving lines among obstacles. A complete analysis of these problems, along with the established theoretical complexities has been provided by Pellegrini [Pel97].

The theoretical results provided by Pellegrini represent only a part of the research based on the Plücker space. The Plücker parametrization can be used to solve various problems involving lines [CEG+96], and has served as a framework for visibility computations in the field of computer graphics [TH99, NBG02, HMN05, Bit02, Mor06].

1.3 From-Point Analytic Visibility

This section provides a concise description of the analytic methods which calculate from-point visibility. We start with the most simple case, the point-to-point visibility. Then, we continue with the beam tracing methods, which were initially designed to replace rays with volumetric structures, in order to take advantage of spatial coherence and improve both the performance and the quality of the rendering. We also describe a method which encodes the conservative visibility from a point using a BSP tree, and a second one which calculates a partial representation of visibility in order to compute analytical irradiance in polygonal environments.

1.3.1 Point-to-Point Visibility

The most simple definition of visibility can be formulated as follows:

Definition 8. *Two points in space are mutually visible if the line containing them does not intersect any other objects between them.*

Let us consider the ray tracing algorithm, which uses rays to determine the visible surfaces from a viewpoint. For each ray, the algorithm determines the first visible surface by testing line-surface intersections between the ray and each object in the scene. Since the image is sampled, the output of the algorithm can be seen as a set of viewpoint-to-point visibility proofs. Thus, ray tracing can be considered as a point-to-point analytic visibility solution.

1.3.2 Beam Tracing

Ray Tracing with Cones

Amanatides [Ama84] proposed a method which replaces rays with cones, in order to gather sufficient information to perform anti-aliasing and achieve various effects such as fuzzy shadows and dull reflections. Instead of shooting one conventional ray through the center of each pixel, a cone is constructed which contains the pixel in question. Determining the intersection of this volume with a primitive provides a double answer: if an intersection exists, and the percentage of the cone which is blocked by the object. Three methods are given for calculating the intersection of the cone with a sphere, a polygon and a plane. A sorted list is maintained of the closest objects which intersect the cone, which are used for anti-aliasing. A restriction is imposed on the input geometry: although overlapping surfaces are calculated correctly, the algorithm fails when faced with adjacent elements.

The method achieves general anti-aliasing, and some auxiliary effects. Dull reflections are correctly obtained for all types of objects, but fuzzy shadows are limited to opaque surfaces only. Because of the nature of the volumetric structure, reflection and refraction calculations are more complex than in the case of convectional rays, and no sufficient information is provided on the construction of the reflected and refracted cones. Although the article claims to have improved the intersection calculations, no comparison is given with respect to a conventional ray tracer.

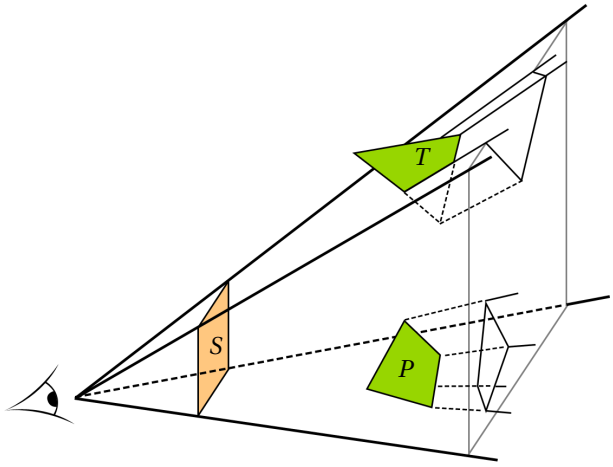


Figure 1.9: Geometry of beam tracing [HH84]. The primary beam is defined by the viewpoint and the screen (S). The beam is tested against the polygons in the scene, and each intersecting object is subtracted from the beam. This leads to arbitrary shapes for the primary beam, thus increasing the complexity of future intersections. Illustration from [HG98].

Beam Tracing Polygonal Objects

Heckbert and Hanrahan [HH84] proposed a method which replaces the conventional ray with a volumetric structure, called a beam. They aim to reduce the number of intersection tests and thus the total rendering time. Moreover, the beam should provide enough information to perform anti-aliasing and to obtain diffuse shading effects. The method is proposed for polygonal environments, and is composed of two parts. First of all, a beam tree is constructed, which is an object space representation of the entire final image. The second part consists of a rendering phase, where the color of each pixel is being calculated.

The beam tracer is a recursive polygon hidden surface algorithm. The procedure begins with an initial beam, having the viewpoint as apex and the screen as section. A depth ordered list of the polygons in the scene is maintained. The beam is intersected with the polygons in this list, in order to find the first visible surface. When such a polygon is found, it is added to the list of visible surfaces and then its shape is subtracted from the beam. This is illustrated in Figure 1.9. If the encountered polygon is reflecting or refracting, new beams will be created for the reflected and transmitted directions.

The main drawback of the beam tracer is that with each new subtraction, the shape of the beam becomes more and more complex. This can be seen in Figure 1.9. Thus, the complexity of each new intersection calculation is increased.

Similarly to [Ama84], secondary beams are not as simple to trace as primary beams. Correct secondary beams can be created for shadows and reflections, which is a linear transformation. However, reflection cannot be handled correctly and the authors propose an approximation.

Dadoun *et al.* [DKW85] construct on the initial method of Heckbert and Hanrahan [HH84], and describe some of its drawbacks and possible optimizations. They propose a hierarchical decomposition of the environment, based on a Binary Space Partitioning (BSP) tree, that will allow to further exploit the scene's coherence.

Pencil Tracing

Shinya *et al.* [STN87] propose a mathematical study and applications of *pencil tracing*. The conventional ray is replaced with a *pencil*, which is defined by an axial ray and several par-axial rays around it, all having a common origin. Thus, a pencil can be simply represented by its spread angle at the start point, which is its direction deviation from its axial ray. The main idea is to provide a mathematical tool which solves some of the limitations associated with cone and beam tracing, and to perform an error analysis in order to improve the image accuracy.

Pencil tracing can be used to accelerate image synthesis, by solving refraction and reflection calculations with matrix-vector operations. However, the algorithm cannot handle the edges of objects and their neighborhood. Thus, a conventional ray tracer is used to cover these areas. Also, small objects are not detected accurately, and may not appear at all in the final image.

The authors also propose a modification of Heckbert and Hanrahan's [HH84] beam tracer, with the purpose of handling refractive objects.

A Beam Tracing Method with Precise Antialiasing

Ghazanfarpour *et al.* [GH98] propose a hybrid method that combines a beam tracer with a ray tracer in order to provide a general solution to aliasing problems. The algorithm is decomposed into two main parts. First of all, a beam tracer is used to recursively detect all regions of the image that may contain aliasing problems. Secondly, a conventional ray tracer is used to calculate the color of each pixel on the screen.

The first part is based on an adaptive recursive subdivision of the image space. A view beam is traced through the entire screen and recursively subdivided until either a uniform region is obtained or the pixel subdivision limit is reached. The view beam is a pyramid defined by the viewpoint and the four rays passing through the corners of an image region. If the corresponding region is ambiguous, it is subdivided and four new sub pyramids are traced. The intersection calculations are replaced with a technique that uses separating planes to detect if an object is inside or outside the beam. When a uniform region has been detected, the next step is to create secondary beams for that region, in order to calculate shadows, reflections and refractions. Once all the ambiguous regions have been detected, a conventional ray tracer is used for the color computation. All the necessary ray-object intersection details have been provided in the first part, thus accelerating the rendering pass.

A Real-time Beam Tracer with Application to Exact Soft Shadows

Overbeck *et al.* [ORM07] propose a new beam tracing method which aims at solving two common drawbacks of earlier methods: the complexity of the intersection tests and the lack of adaptability to acceleration structures. Their first contribution is a robust beam-triangle intersection algorithm, which ensures sub-beams with convex cross-sections. The second contribution consists in developing a kd-traversal method for beam tracing.

The algorithm starts with one large pyramidal beam representing a volume of perspective parallel rays, which will be recursively split at triangle edges into a list of beams which represent the visible surface of the scene. In order to take advantage of parallel SIMD instructions, beams are represented by three or four corner rays having a common origin. The beams traverse the scene's kd-tree and are split into two lists of sub-beams: hit and miss beams. The latter ones continue the tree traversal until they either hit a triangle or exit the scene.

Since the method scales linearly with the visible triangles, the authors note that the algorithm is more efficient for shadow beams, than for calculating primary visibility. Also, if the visible triangles become smaller or equal to the considered pixel size, other ray tracing methods yield faster rendering times for both primary and shadow beams. The method provides accurate and high quality results, but is limited to scenes with moderate complexity and the images are rendered at a 512×512 pixels resolution in order to achieve interactive frame rates.

1.3.3 Other Methods

Occlusion Trees

Bittner *et al.* [BHS98] proposes a method which encodes the visibility from a point in a BSP data structure, called the *occlusion tree*. For the considered viewpoint a set of potential occluders is determined using a spatial hierarchy structure. These occluders are then processed in a depth first order, and their *occlusion volumes* are used to build the occlusion tree. For a convex polygon having e edges, its occlusion volume corresponds to the intersection of $(e+1)$ half-spaces. These half-spaces are formed by the support plane of the polygon and the planes passing through the viewpoint and each edge of the polygon.

Each inner node of the occlusion tree is associated with a plane passing through the viewpoint and an edge of an occluder. Thus, the nodes of the occlusion tree correspond to spatial regions. The visibility of such a region can be determined as visible, partially visible or invisible. Starting from the root of the occlusion tree, if a node is determined as visible or invisible, all its descendants are considered to be visible or invisible, respectively. On the other hand, if a node is marked as partially visible, its descendants must be further tested. When the visibility of all the leaves is known, the objects belonging to regions that are either visible or partially visible form a superset of polygons, which is then used by the graphics hardware (z-buffer) to solve exact visibility from the viewpoint.

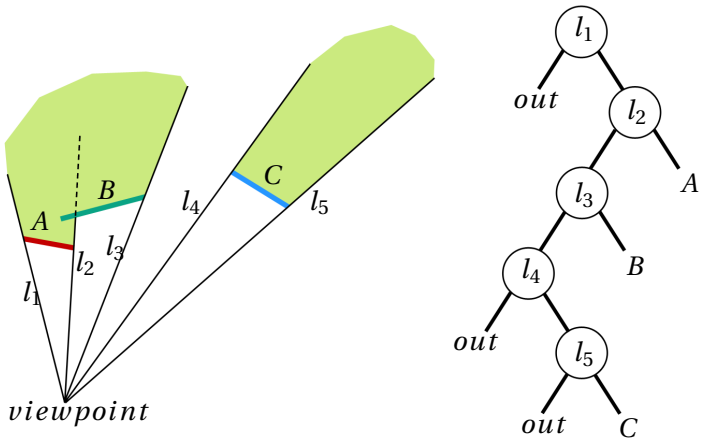


Figure 1.10: 2-dimensional illustration of an occlusion tree [BHS98]. The occluders are processed in a depth first order, and successively intersected with the hyperplanes in the inner nodes. The visible leaves are replaced by the BSP representation of the occluder reaching them. If the leaf is invisible, the fragment of occluder is discarded. This is the case of the fragment of polygon B which is behind A.

The visibility of a polygon is determined as follows: In each inner node of the occlusion tree, the position of the polygon is tested with respect to the associated plane. If the polygon lies completely in front or back of the plane, the visibility algorithm is applied on either the left or the right child of the current node. Otherwise, the polygon is split in two fragments and the algorithm continues in both children using the relevant fragments. The fragments which reach a leaf corresponding to an invisible region are deleted. Otherwise, the leaf is replaced with the occlusion volume representation of the fragment. Figure 1.10 gives an illustration.

It is important to note that the leaves of the occlusion tree correspond to view beams. The main advantage of this method, is that the algorithm only performs polygon–plane operations, instead of the complex intersections which need to be calculated for the majority of beam tracing methods. Note that Overbeck *et al.* [ORM07] proposes a similar technique (splitting beams at triangles' edges) in order to increase the robustness of their beam tracing algorithm.

The occlusion trees are extended to visibility from a region in 2-dimensional space [BP01] and in 2.5-dimensional space [BWW05]. In his PhD thesis [Bit02], Bittner provides an implementation of the occlusion tree to solve the visibility from a region in 3-dimensional space, based on the Plücker coordinates. This is detailed in Section 1.4.12.

Exact Illumination in Polygonal Environments using Vertex Tracing

In the context of exact computation of irradiance, Stark *et al.* [SR00, Sta02] propose a vertex tracing algorithm and an alternative analytical expression to Lambert's formula, which is adapted to partially occluded emitters.

In the case of an emitting polygonal light, the direct lighting calculated at a receiver point (irradiance term) is a surface integral that can be evaluated in closed-form using Lambert's formula [Lam60]. However, the drawback of Lambert's formula is that it only works for a polygonal light source which is entirely visible from the receiver point. In order to handle partially occluded environments, the exact fragments of the light source need to be calculated for each receiver point, and then Lambert's formula can be applied to each individual fragment.

The alternative formula provided by Stark *et al.* is valid in general environments where the light source can be partially occluded. Instead of summing over the edges of a polygon, the new expression is formulated in terms of vertices. Thus, all polygon clippings and fragment calculations are avoided.

First of all, the emitting polygons are projected onto an image plane, parallel to the surface containing the receiver point. Three types of vertices can be distinguished: intrinsic, which are the vertices of the original polygons, apparent, which are formed by the apparent intersection of two edges, and conjunctive, which may appear to coincide as viewed from the received point. The visibility of these vertices is determined using ray tracing. A ray is traced from the receiver point and through each projected vertex, and all the intersection information is collected and used to calculate the final irradiance value.

Although correct from a mathematical point of view, the method proposed by Stark *et al.* suffers from numerical instabilities. More exactly, theoretical conjunctive vertices cannot be determined exactly when using finite precision floating-point implementations. A solution proposed by the authors is to trace a cone, instead of a ray, as described by Amanatides [Ama84]. This allows for detecting conjunctive vertices more accurately. However, cone tracing is not enough to avoid all errors and furthermore, it slows the computations by a factor of three.

1.4 From-Polygon Analytic Visibility

This section outlines the techniques which attempt to calculate exact and analytic from-polygon visibility. We divide these techniques into two categories. The first one contains the methods which provide a description of 3-dimensional visibility independently of any parametrization. The second category concerns the methods which rely on the Plücker parametrization. By analyzing these techniques we remarked that they all represent visibility using a sub-part of a line arrangement, as described by Pellegrini (see Section 1.2.2).

The first part summarizes some definitions and concepts which are of importance in this context, followed by a description of the first category of visibility methods. Then we provide a general overview of how the Plücker parametrization can be used to solve visibility problems. The methods relying on this theoretical framework are detailed next.

1.4.1 Preliminary Definitions

Stabbing Lines

Definition 8 characterizes the visibility between two points. If we consider two objects, we can translate their mutual visibility using the set of lines intersecting them. Such a line is referred to as a *stabbing line*.

Definition 9. *Let $A = \{A_i, 1 \leq i \leq n\}$ be a set of objects. A line corresponds to a stabbing line if it intersects (or it is tangent to) all the objects in the set.*

Definition 10. *$stab(A_1, ..., A_n)$ denotes the set of all the lines stabbing all the objects in A.*

Degrees of Freedom

The *degrees of freedom (DOF)* associated with an object denote the number of independent parameters that define the displacements which are available for the object within the considered space [wik12a]. The degree of freedom can also be deducted using the minimum number of parameters required to specify a position. In 3-dimensional space a line can have maximum four degrees of freedom.

Visual Events

The visual events (or visibility events) represent topological changes in visibility. This occurs when an object disappears or appears as seen by a mobile observer which moves through the scene. An example of visual events in image synthesis is the boundaries of hard shadows, or the limits of the light, penumbra and umbra in the case of soft shadows. It is important to underline that the visual events can be represented using the sets of lines which are incident to the given geometry.

Several types of visual events can be characterized. An *EEE (Edge-Edge-Edge)* visual event is defined by three distinct edges, in a non degenerate position. The lines incident on these three edges

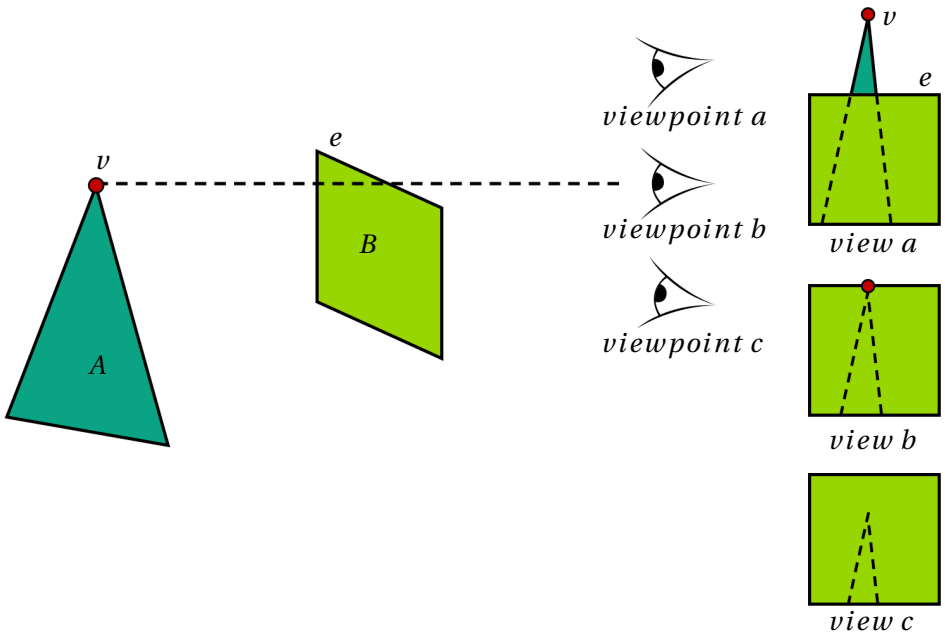


Figure 1.11: A *vertex-edge* visual event, defined by a vertex (v) of polygon A and an edge (e) of polygon B. **Left**: Three viewpoint configurations. **Right**: The views from the viewpoints. View a: vertex v is completely visible. As the viewpoint moves downwards, vertex v becomes hidden (view c). View b is the limit viewpoint and it lies on a visual event.

have two degrees of freedom. A *VE (Vertex-Edge)* visual event is defined by a vertex of one polygon and an edge of a second distinct polygon. Thus, the incident lines have one degree of freedom. A *VV (Vertex-Vertex)* visual event is defined by a vertex of one polygon and a vertex of a second polygon. The line incident on these two vertices has no degrees of freedom. Figure 1.12 and 1.11 illustrate the EEE and VE visual events.

Extremal Stabbing Lines

Definition 11. *A critical line is tangent to at least one object. Therefore, it has at least one less degree of freedom.*

Definition 12. *An extremal stabbing line has no degrees of freedom.*

Figure 1.13 illustrates the possible configurations of extremal stabbing lines in 3-dimensional space.

All these definitions indicate that it may be more natural to describe visibility problems in line space, rather than in the classical 3-dimensional Euclidean space. However, the continuous sets of lines which represent the visual events are generated by the 3-dimensional geometry of the scene. Therefore, the following question arises: Is line space or object space more suitable to represent and solve from-polygon visibility problems ? The following sections present the two categories of methods, those who represent from-polygon visibility in 3-dimensional object space, and those who chose a line space representation.

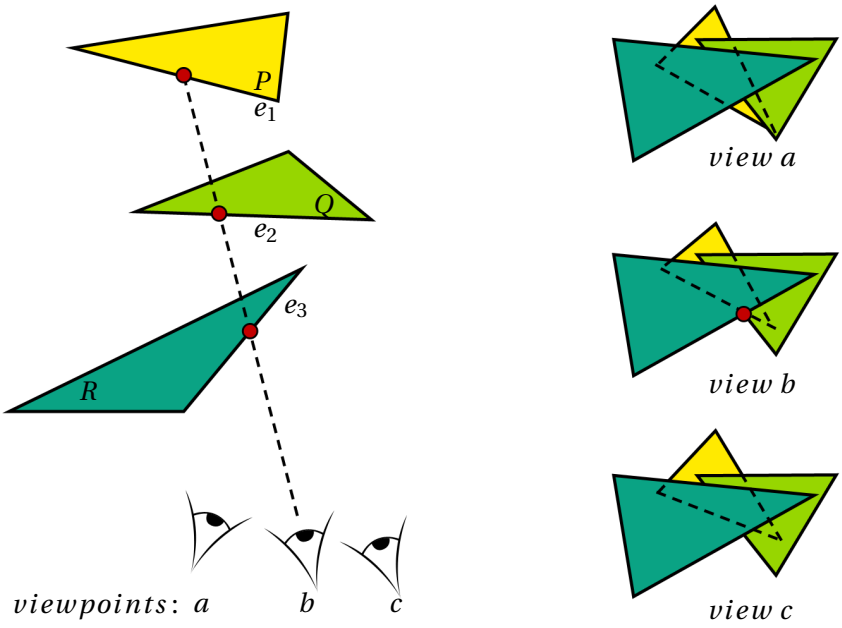


Figure 1.12: An *edge-edge-edge* visual event, defined by three distinct edges (e_1, e_2, e_3) of polygons P, Q, R respectively. **Left**: Three viewpoint configurations. **Right**: The views from the viewpoints. View a: vertex v is completely visible. As the viewpoint moves, polygon P becomes hidden more and more by Q and R. View b is the limit viewpoint, where the three edges appear to be intersecting. This view lies on a visual event.

1.4.2 The Aspect Graph

The Aspect Graph is a representation of all the possible views of a an object. It was first introduced in computer vision for model-based recognition [KD76, KD79, EBD92]. The idea was to define all the possible views of an object in order to identify it, whatever the viewpoint. In order to obtain such an information, the viewing space needs to be partitioned into regions where the view is qualitatively invariant. Thus, a robot can compare its view of a particular object with the views stored in the aspect graph. If one of the views corresponds, then the robot is able to identify the object.

The aspect graph is directly linked to the notion of visual event. More exactly, each node of the aspect graph represents a general view of the considered object, and each arc is a visual event describing a transition between two neighboring general views. This organization translates the fact that the view of an object changes as the observer is traversing a visual event. Figure 1.14 illustrates the aspect graph of a convex cube, in orthographic projection.

Building the aspect graph is a complex task. Calculating the visual events is not sufficient, and their arrangement (*i.e.* the decomposition of the viewpoint space into cells) needs to be computed also. More exactly, this involves calculating the intersections between the visual events in order to obtain the segments of events which form the boundaries of the cells. An arrangement of n hyperplanes in $\mathbb{R}^d$ has a maximum of $O(n^d)$ cells [EOS86]. For convex polyhedrons, this gives $O(n^2)$ cells for an arrangement of n lines in $\mathbb{R}^2$ and $O(n^3)$ for $\mathbb{R}^3$. For general polyhedrons, the visual events to calculate are much more complex, which yields a maximum size for the aspect graph of $O(n^6)$ for orthographic projection and $O(n^9)$ for perspective projection.

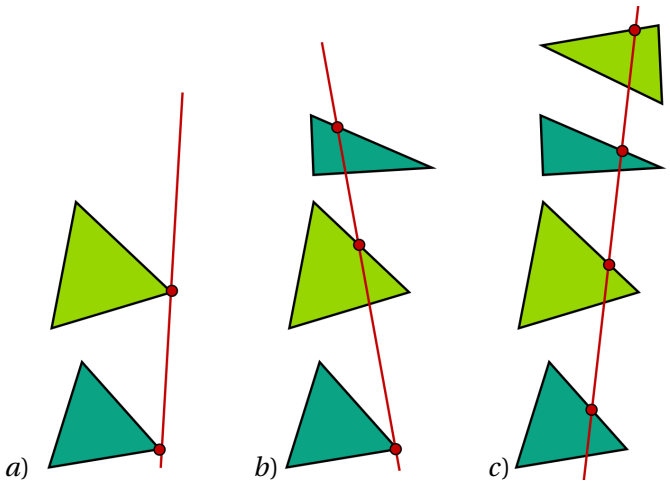


Figure 1.13: Possible configurations for an extremal stabbing line in 3-dimensional space. a) The line is incident on two vertices. b) The line is incident on one vertex and two edges. c) The line is incident on four edges. The first two cases are just special cases of the third, since a vertex is incident on two edges.

From a practical point of view, the majority of the construction algorithms approximate the aspect graph. The viewpoint space is sampled and similar views are merged to obtain the graph's nodes [HK85]. The drawback of such methods is that their accuracy is dependent of the sampling density and thus some important views may be missed. This motivated a series of research which aimed to compute exact construction methods [GMM90, GCS91]. However, no exact and complete solution to the construction problem has been successfully implemented.

In his PhD thesis, Durand [Dur99] provides a synthesis of the existing construction algorithms. Moreover, a state of the art has been provided by Eggert and Dyer [EBD92].

1.4.3 The Asp

The *Aspect representation (asp)* is a intermediate data-structure to build the aspect graph of polyhedrons. For a convex polygon, the asp is the set of lines intersecting it. Thus, visibility and occlusion can be described using the asp [PD90]. More exactly, the occlusion created by two polygons corresponds to the union of their asp. Similarly, the occlusion of a polygon by another polygon can be obtained by subtracting their asp. The definition of the asp depends on the viewing space considered. In the case of orthographic projection, the asp is defined in the 4-dimensional space of lines, while for perspective projection, the 5-dimensional space of rays is used.

No full implementation of the *asp* exists, although research exist for model-based recognition, or view maintenance [PSD90].

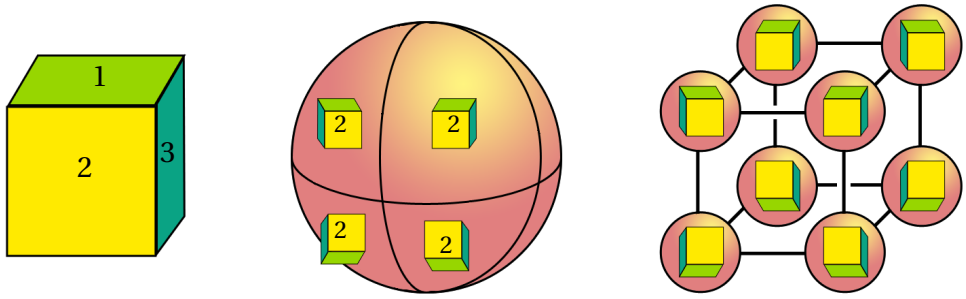


Figure 1.14: The Aspect graph of a convex cube in orthographic projection [Dur99]. *a*) The initial cube with numbered faces. *b*) The partition of the viewpoint space for orthographic projection with some representative aspects. *c*) Corresponding aspect graphs.

1.4.4 2D Visibility Complex

Pocchiola and Vegter [PV93] define the 2-dimensional visibility complex. The intuitive concept behind their data structure is to group all the rays which see the same objects. These rays are actually *maximal free segments*, which are plane segments which do not intersect the interior of any object, and whose extremities lie on the objects of the scene (or are at the infinity). Using this notion, the 2-dimensional visibility complex is defined as a partition of the plane segments according to the objects at their extremities. They use a parametrization of directed lines in the plane based on the polar coordinates, and define the *free space* containing an object at infinity, which is a basically a disk sufficiently large to enclose all the objects in the scene. This last detail allows an uniform description of the visibility complex, and reminds of the similar description of projective spaces.

The authors give a $O(n \log n + m)$ complexity for building the 2-dimensional visibility complex, where n is the total number of objects and m is the size of the corresponding visibility graph. An optimal time and space construction algorithm is developed by Pocchiola and Vegter [PV96] for curved objects, and by Riviere [Riv95] for polygonal objects.

The initial aim of the 2-dimensional visibility complex was to provide solution for problems where objects act as obstacles, by either blocking the view from other objects, or in motion planning where they may interfere with the movement of an object along a straight path. In practice, the 2-dimensional visibility complex was used for 2D global illumination simulations [DORP96, ORDP96], and for calculating and maintaining views [Riv97a, Riv97b].

This analytical 2-dimensional approach to describing visibility has inspired the 3D visibility complex study.

1.4.5 3D Visibility Complex

Inspired by the work of Pocchiola and Vegter, Durand *et al.* [DDP96, DDP02, Dur99] propose a theoretical study of global visibility in 3-dimensional space. They describe the 3D visibility complex, a data structure which contains all the visibility events corresponding to a scene composed of

polygons and convex smooth surfaces.

Similarly to the 2D Visibility Complex, their analysis is based on the notion of free maximal segment, which is interpreted as a ray being able to see the two objects at its extremities, without intersecting their interior. A distinction is made between line and ray visibility. In the case of line visibility, all the objects intersected by the line are considered (as if they were transparent). On the other hand, if occlusion is taken into account, only the first object intersected by a ray is reported (objects are not transparent).

The main concept is to group the lines of the 3D space into connected components according to the objects they intersect. This forms a *visibility arrangement*. The 3D visibility complex is defined as the partition of the maximal free segments of the 3D space into connected components according to the objects they touch. The underlying idea is that for each set of rays the visibility properties are invariant, and the boundaries of these sets correspond to changes in visibility. This is connected to the notion of *visual events*.

The 0-dimensional faces of the 3D complex are segments which are tangent to four objects, the 1-faces are tri-tangent segments, the 2-faces are bi-tangent segments and finally the 3-faces are segments which are tangent to one object. More exactly, the set of lines related to a k-face have k degrees of freedom.

The 3D visibility complex is described in line space, or more exactly in maximal free segment space. They use a parametrization which maps 3-dimensional lines to points in a 4D space. This parametrization is based on the spherical coordinates of the director vector for the lines and a projection which retains the position of the line. However, the authors note that their concepts do not rely on a particular parametrization of lines and the same notions can be described using different parametrization. For example, the visibility arrangement is equivalent to the arrangement proposed by Pellegrini [Pel91, Pel04] and described in Section 1.2.2.

The theoretical size of the 3D visibility complex is $O(n^4)$, where n is the number of considered objects. Moreover, a $O((k+n^3)\log n)$ theoretical build time is given, where n retains its previous definition and k is the number of 0-faces of the complex. No complete implementation of the 3D visibility complex exists and the authors note that its direct practical interest is questionable. This is due to the fact that implementing a 4D arrangement as the one described is not straightforward and it is prone to numerical instabilities and robustness issues. Moreover, as compared to the 2D visibility complex, the 3D data structure is not a cell complex (the faces can have holes) and some of the properties which are valid in 2-dimensional space do not hold in 3D.

1.4.6 The Visibility Skeleton

The Visibility Complex encodes all the visibility information for a scene in 3-dimensional space. In order to avoid calculating a 4D cell-complex, Durand *et al.* [DDP97] propose the Visibility Skeleton,

which is partial representation of the Visibility Complex. The main idea is to create a complete catalog of visual events, which encodes all the possible changes in visibility in a considered scene.

With respect to the Visibility Complex, only the 0 and 1 -dimensional faces are kept. More exactly, the Visibility skeleton is a graph structure where each node corresponds to an extremal stabbing line and each arc translates a visual event (VE or EEE). An arc connects two nodes if the visual event representing the arc is delimited by the two extreme stabbing lines stored in the considered nodes.

A construction process is presented for scenes containing oriented convex polygons. A catalog is created, which encodes all the possible geometric configurations which can define extremal stabbing lines and visual events. The scene is then processed and the occurrence of such configurations is detected.

A series of constraints are imposed on the input scene. The algorithm cannot handle a number of geometric degeneracies, such as intersecting edges or vertices which are almost aligned or coplanar. All these geometric degeneracies multiply the number of possible configurations which create visual events. This can lead to an infinite catalog and cannot be dealt with correctly. Moreover, if a model is inaccurate and its polygons are lacking connectivity information, a correct catalog cannot be build. All these drawbacks are also demonstrated by the small test scenes (less than 1500 polygons). In order to process larger and more complex environments, the authors propose a lazy on-demand version of the construction algorithm, adapted to global illumination. This has a positive impact on the computation times, since less data is calculated. However, it does not seem to solve the limitations concerning the size of the scene because of the geometric degeneracies.

The authors describe the Visibility Skeleton as a multi-purpose tool, which can solve different problems. An extension of the Visibility Skeleton has been used for global illumination computations [DDP98, DDP99]. Although the resulting images are of high quality, the implementation suffers from important drawbacks. First of all the memory usage is high, limiting the applications to really small scenes (maximum 500 polygons). Secondly, the method suffers from numerical robustness and cannot handle all the degenerate cases that arise.

Duguet *et al.* [DD02, Dug04] propose a more robust visibility skeleton, used in the context of shadow computation. In order to correctly handle geometric degeneracies, they no longer rely on a visual event catalog, but use a flexible definition of extremal stabbing lines based on a ϵ precision. The tests show that scenes up to 120 000 polygons can be handled. Although many of the previous limitations are solved, the detection of skeleton's elements are dependent of the ϵ parameter. Moreover, if this value is to large, the shadows are no longer correct or even visible. As with the original Visibility Skeleton, the input scene must have a complete and correct connectivity information.

1.4.7 Plücker space and Visibility

In order to provide an accurate description of visibility, a line parametrization corresponding to a line space is necessary. The 2D and 3D Visibility Complexes, as well as the Visibility Skeleton, construct a representation of visibility which is independent of the chosen line parametrization. Their characterizations of coherent sets of lines prove to be either too complex to implement (3D visibility complex), or not sufficient enough to represent a complete information of visibility (visibility skeleton). This underlines even further the need for a solid mathematical framework which allows working with continuous sets of lines.

Therefore, the majority of from-polygon visibility solutions are based on the Plücker parametrization and the concepts described by the theoretical framework, in Section 1.2.2. More exactly, they formulate the visibility problems in terms of sub-parts of the arrangement of lines described by Pellegrini. Before presenting these techniques, we summarize some relevant notions and definitions.

Definition 13. *Let A and B be two convex polygons. A third convex polygon O is called occluder of A and B if it intersects the convex hull of A and B.*

To determine if A and B are mutually visible, or if B is visible from A, analytic methods study the set of stabbing lines of A and B. These lines are represented in Plücker space as a polyhedron (we note this $P(A,B)$).

For an occluder O, the set of lines stabbing both O and B (or both O and A) can also be represented as polyhedron in dual space, $P(O,B)$ ($P(O,A)$, respectively).

Two polygons are mutually visible if at least one of their stabbing lines does not intersect any of their occluders. More formally:

Definition 14. *Let A and B be two convex polygons and let O(A,B) = $\{O_i, 1 \leq i \leq n\}$ be the set of their occluders. A and B are visible if and only if there is at least one stabbing line $l \in stab(A,B)$, such as $l \notin \bigcup_{i=1}^{n} stab(O_i)$.*

Calculating the set of lines which stab only A and B and none of their occluders can be done by subtracting from $P(A,B)$ the parts representing the lines stabbing each occluder. This can be done using CSG operations in Plücker space. An example is given in Figure 1.15. After each split, the remaining volume is not necessarily convex. In order to deal with convex polyhedrons only, this volume is split into convex parts, forming a *complex of convex polyhedrons*.

A few remarks can be made concerning these operations.

First of all, in order to perform the required 5D CSG operations, the polyhedron $P(A,B)$ needs to be bounded [Nir03, NBG02]. The solution was to add two hyperplanes, one on each side of the Plücker quadric, in order to obtain a complete V-representation of $P(A,B)$. The position of

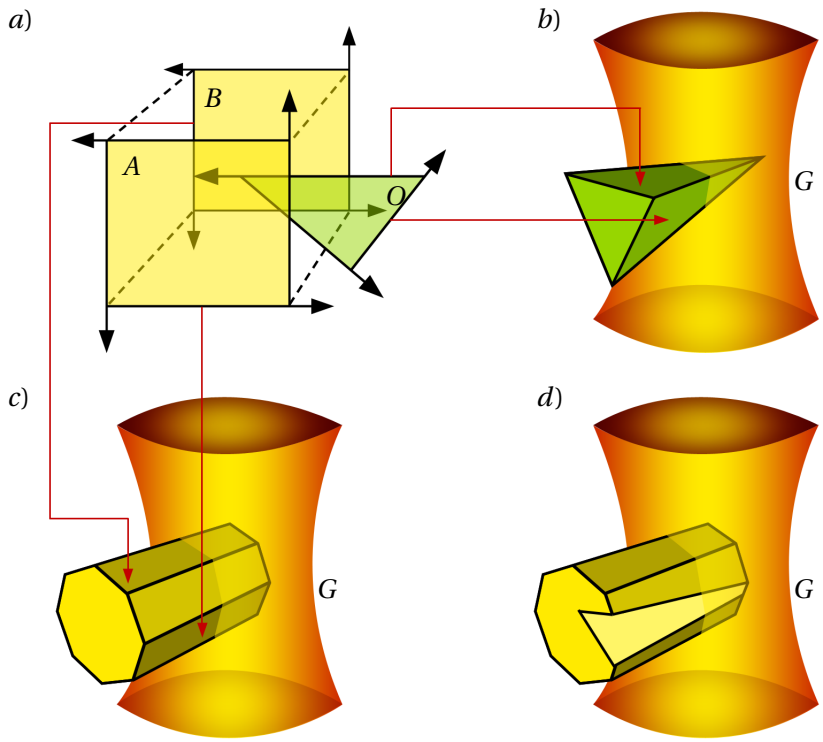


Figure 1.15: Calculating occlusion for two polygons A and B, with occluder O [Nir03, NBG02]. a) The given configuration. b) A visualization of the mapping of the edges of O to a volume in Plücker space. G is the Plücker quadric. c) A visualization of the mapping of each edge of A and B which form a volume in Plücker space. d) The subtraction of the occluder volume from the initial volume. What remains represents the set of lines between A and B, which miss O.

these hyperplanes has an important impact on the computations. Each time $P(A,B)$ is cut using a hyperplane from an occluder, the resulted complex may contain polytopes which represent no real lines (*i.e.* no intersection with the Plücker quadric). Thus, the closer the two supplementary hyperplanes are to the quadric, the less redundant operations are made. This method has been used by all works on from-polygon visibility relying on the Plücker parametrization. However, none of these methods obtain the minimal polytope which contains all the stabbing lines of two convex polygons. The existence of the minimal polytope, as well as its characterization has been proved by Charneau [Cha07] and will be detailed in Section 1.4.14.

Second of all, since all operations are performed in Plücker space, the final complex of polytopes needs to be intersected with the Plücker quadric in order to determine the existence of a real solution.

1.4.8 Cells and Portals

Teller *et al.* [TH92] extends the ray-triangle intersection problem to a set of oriented convex polygons. More exactly, for a given collection of oriented convex polygons, he wants to determine the existence of a line which simultaneously intersects them. Let $\{l_i, 1 \leq i \leq n\}$ be the lines spanning all the edges of the considered polygons. As explained in Section 1.2.1, a line intersects a triangle if its relative orientation is consistent with respect to the lines spanning the triangle's edges. Therefore, any line d intersecting all the polygons verifies

$$\pi_d \in \bigcap_{i=1}^{n} h_{l_i}.$$

Where π_d is the Plücker point mapped from the line d, and $h_{l_i}, 1 \leq i \leq n$ are the Plücker hyperplanes corresponding to the lines spanning the polygons' edges.

Each hyperplane splits the Plücker space into two half-spaces. The intersection of the half-spaces induced by all the hyperplanes h_{l_i} is a convex polyhedron in $\mathbb{P}^5$. Teller remarks that this polyhedron is not necessarily bounded. However, its intersection with the Plücker quadric is. The intersection of the polyhedron with the quadric corresponds to the Plücker points representing all the lines stabbing the polygons, $\bigcap_{i=1}^{n} stab(P_i)$. Therefore, in order to decide if a real line exists which stabs all the polygons, an intersection test must be performed with the Plücker quadric. Teller *et al.* [TH92] provide an algorithm which determines if the related polyhedron exists (*i.e.* non empty), and if so, a stabbing line is calculated. The time complexity is $O(n^2)$, where n is the total number of edges of the considered polygons.

If the convex polygons correspond to portals which connect different cells, the above problem can be formulated in terms of visibility. More exactly, for a set of portals, we want to know if there is a sight line allowing one cell to be visible from another.

Teller *et al.* [Tel92, TS91] propose a cell to portal method in global illumination application. Although the visibility through a sequence of portals theory is valid for arbitrary sets of convex polygons, the proposed application is adapted to architectural environments which are axis-aligned. Teller *et al.* [TH93] solve this limitation, by proposing a conservative approach for which the result is a superset containing all partially and totally visible elements. Concerning the implementation, Teller notes in his PhD thesis that a robust package for CSG operations is required. Moreover, these complex operations are prone to numerical instability and degeneracy problems. [TH93] is a simplified method, based only on point-hyperplane comparisons in Plücker space.

1.4.9 PSP Tree

Mount and Pu [MP99] make one of the first attempts to represent the exact from-region visibility. They encode this information in a PSP tree (*Plücker Space Partition*), which is actually a BSP tree (*Binary Space Partition*) in Plücker space. Each node of the tree represents a region in Plücker space. Each inner node of the data structure contains a Plücker hyperplane corresponding to a directed line

in 3-dimensional space. This hyperplane subdivides the region associate with the node in two. Each leaf represents a coherent set of lines which intersect the same objects. The algorithm calculates the complete arrangement of hyperplanes, as described by Pellegrini (see Section 1.2.2). Thus, for each set of lines (leaf), we know which polygons are intersected, but not in which order.

An optimized version of the algorithm is also presented. Two trimming operations are applied to the PSP tree in order to reduce its size.

Although the construction of the PSP tree starts with a specific view region, the algorithm calculates and stores the complete arrangement of hyperplanes generated by the scene's geometry. Therefore, they are bounded by the theoretical complexity of an arrangement of real lines in Plücker space, which is $O(n^4 \log n)$ (see Section 1.2.2). After a series of tests, Mount and Pu conclude that the practical complexity of their algorithm is $O(n^{5.17})$ for the base version, and $O(n^{4.31})$ for the optimized version.

However, the algorithm of Mount and Pu seems impracticable. The tests presented by the authors were done on scenes of maximum 15 random triangles. And even for such small number of polygons, the time and memory consumption are a clear drawback.

1.4.10 Exact From-Region Visibility Culling

Nirenstein *et al.* [NBG02] propose an algorithm which answers if two polygons are mutually visible. The aim of the method is to provide a pre-process step for computing Potentially Visible Sets (PVS) for visibility culling during walkthroughs. The algorithm only tests if the pair polygons are mutually visible or invisible, and does not aim to calculate a full representation of the visibility information. Moreover, a single proof of visibility is sufficient (*i.e.* a single line stabbing the two polygons, which does not intersect any other geometry between them) and the calculated data is dropped as soon as such a visibility/invisibility answer has been found.

The principle of the method is to represent the polygons and their stabbing lines in Plücker space, and then solve the visibility by subtracting the sets of occluded lines from the global set of stabbing lines, as explained in Section 1.4.7. Let P_S and P_T be the two polygons for which visibility needs to be answered. The first step is to clip all the scene's geometry to the interior of the convex hull defined by P_T and P_S. All the occluders intersecting this convex hull are blocking a set of lines between P_S and P_T. The occluders are tested successively, and for each one, the set of lines it blocks is removed. If the remaining (after all occluders have been processed) set of lines is empty, then the two initial polygons are mutually invisible. If, on the other hand, the remaining set is not empty, the resulting structure needs to be intersected with the Plücker quadric. If no intersection is found, then again there are no real lines which miss all the occluders, and thus P_S and P_T are mutually invisible. Only if the intersection of the final structure with the Plücker quadric is not empty, the initial polygons are considered to be visible.

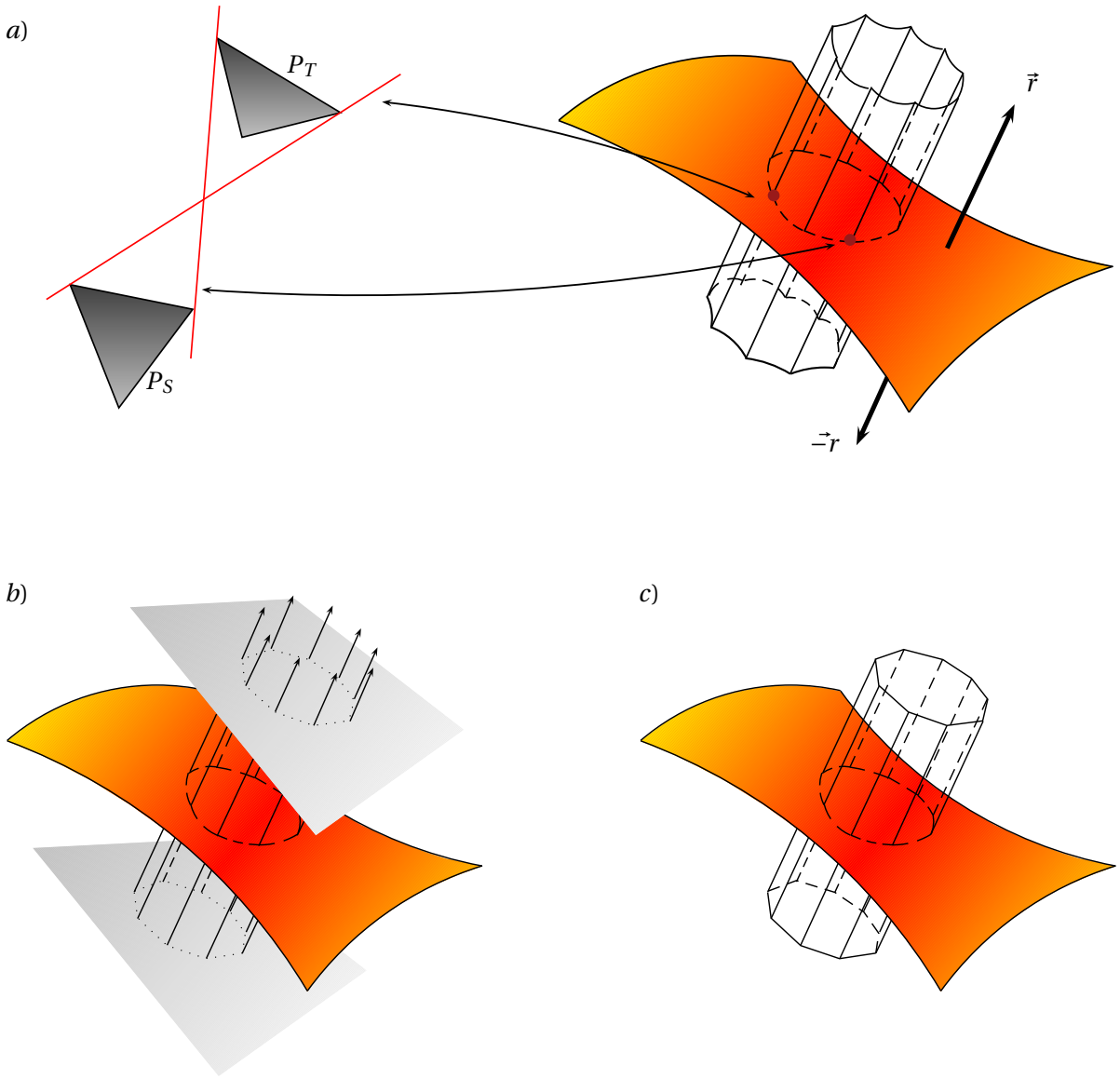


Figure 1.16: The Plücker convex polyhedron containing all the lines stabbing two convex polygons (P_S and P_T) is unbounded. Nirenstein's solution [NBG02] consists in adding two planes, one on each side of the Plücker quadric, in order to cap the polyhedron. a) The polyhedron is open in directions $\vec{r}$ and $-\vec{r}$. The intersection of each edge with the Plücker quadric is a line containing one vertex of P_S and one vertex of P_T. b) In order to bound the polyhedron, two planes orthogonal with $\vec{r}$ are added on each side of the Plücker quadric. c) The vertices of the resulted polytope are the projections of the vertex-to-vertex lines on the two planes.

In his PhD thesis, Nirenstein [Nir03] details the *selective stabbing problem*, a generalization of Teller's stabbing problem [TH92]. This can be summarized as follows: For two sets of convex polygons, compute a representation of all the lines stabbing all the polygons in one set and missing all the polygons is the second set. The case where the first set has only two elements (P_S, P_T) corresponds to the visibility problem in the above described algorithm. For this case, Nirenstein provides a $O(mn)$ worst case optimal algorithm which constructs a polytope in Plücker space containing all the lines stabbing both P_S and P_T (m and n are the number of edges of P_S and P_T,

respectively). The constructed polytope is not minimal and the calculations require that one of the polygons is embedded in the $x = 0$ plane. Figure 1.16 illustrates the construction of this polytope.

Once the stabbing polytope has been constructed, the lines blocked by each occluder are removed. Let O be an occluder of P_S and P_T. The hyperplanes corresponding to the support lines of O's edges define a convex volume in Plücker space. Subtracting this volume from the initial stabbing polytope is equivalent to calculating the set of lines between P_S and P_T which are not blocked by O.

The subtractions are done using 5D CSG operations, based on the algorithm provided by Bajaj and Pascucci [BP96]. This method solves the problem of intersecting a complex of convex polytopes with a hyperplane and is valid in any dimension. The advantage of the algorithm is that numerical operations are reduced to the bare minimum. However, it requires calculating the incidence graph for the polytope $P(P_S, P_T)$.

In order to avoid as much as possible the 5D CSG operations which are both time consuming and prone to numerical instabilities, Nirenstein *et al.* [NBG02] constructed a framework which implements early termination mechanisms and organizes both geometry and visibility queries.

A ray casting mechanism is used to decide if two polygons are mutually visible without using the complex mechanism described above. More exactly, a number of rays is cast from one polygon to another. If at least one ray reaches the second polygon, without intersecting any occluder, then the faces are considered visible. On the other hand, if all the rays are blocked, then the visibility algorithm is applied.

The data resulted from this trivial acceptance step is further used to optimize the visibility algorithm. The authors remark that 5D CSG is more efficient if a large number of stabbing lines can be removed as soon as possible. Thus, the occluders can be sorted according to the number of rays that intersected them during the test step. The one having the largest number of stabbing rays will be subtracted first. Each new choice is made by removing the rays that have already been accounted for.

Moreover, the scene is hierarchically subdivided, so that visibility queries could be organized by view cells. Thus if a region is detected as invisible, there is no point in testing its individual polygons. The faces of a cell are used as virtual occluders. If a face has been found as invisible, it will also hide all the objects located behind it.

The complex computations associated with 5D CSG operations have an impact on execution time, memory consumption and robustness of the solution. For scenes of about 1.45 millions triangles, the algorithm processes the scene in approximately 70 hours, on a 1.7Ghz Pentium 4. Despite these drawbacks, Nirenstein's algorithm has the merit of being the first practical solution to the polygon-to-polygon visibility problem.

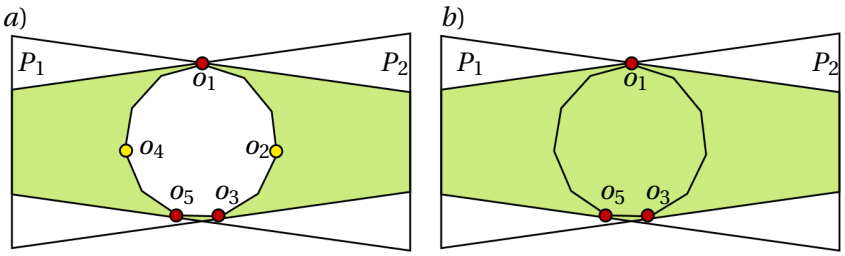


Figure 1.17: Not all the edges of an occluder need to be used when splitting a stabbing complex. In this example [HMN05], P_1 and P_2 are two query polygons, and $\{o_1,...,o_5\}$ are the edges of one of their occluders. a) o_2 and o_5 are internal edges, while o_1, o_3 and o_4 represent from-region silhouette edges. This translates the fact that they are from-point silhouette edges simultaneously for at least one point of the polygon P_1 and at least one point of the polygon P_2. Thus, they define visual events. b) The internal edges are redundant they are not used when calculating the set of blocked rays.

1.4.11 A Low Dimensional Framework for Exact Polygon-to-Polygon Occlusion Queries

Haumont *et al.* [HMN05] build on Nirenstein's method, and provide an improved algorithm for answering polygon-to-polygon occlusion queries.

First of all, they generalize Nirenstein's algorithm for determining a stabbing polytope. Geometric transformations are no longer required to achieve a valid configuration. Moreover, they demonstrate that maintaining occlusion in Plücker space requires only the 1-skeleton of the polytopes: the 0-faces (vertices) and 1-faces (edges). However, additional combinatorial operations are required in order to identify the new edges. Also, the obtained polytope is still not minimal.

Contrary to Nirenstein's approach, the computations are restricted to the silhouette edges, instead of individual polygon edges. The silhouette of an object from a view point can be seen as the set of edges which form the visual boundaries of the object. More exactly, each boundary defines a visual event. This is used to avoid redundant 5D CSG operations, by excluding the internal edges when splitting a stabbing complex of polytopes. Figure 1.17 gives an illustration.

All this is embedded in a new framework, which aims to find an answer to the polygon-to-polygon occlusion queries as fast as possible. Early termination mechanisms are implemented, based on ray casting into the apertures created by the occluders which have already been removed. The same rays are used for occluder selection, thus taking advantage of previous computations. Similarly to Nirenstein's approach, a single proof of visibility is enough to terminate the query.

The practical complexity for answer a visibility query for two polygons is of $O(n^{1.44})$, where n is the number of tested occluders. Compared to Nirenstein's approach, Haumont's framework can process scenes of as large as 4.6 million triangles. Although the early termination mechanism is improved, in the case of mutually invisible polygons, Haumont's method would have done more calculations than Nirenstein's algorithm. As noted by the authors, the most costly operations remain the CSG calculations in Plücker space.

1.4.12 Exact From-Region Visibility - Occlusion Tree

Contrary to Nirenstein and Haumont, Bittner [Bit02] proposes to encode the visibility information in a data structure, called the *occlusion tree*. Similarly to the PSP tree proposed by Mount and Pu (see Section 1.4.9), he builds a BSP tree in Plücker space. However, the encoded information is different, since Bittner also takes into account a depth information, and does not compute the same arrangement of hyperplanes. The method is proposed in the context of calculating exact PVS sets.

Section 1.3.3 has presented the from-point visibility algorithm, based on occlusion trees. The exact from-region visibility algorithm in 3-dimensional space can be viewed as a generalization of the from-point method. The occlusion tree becomes a BSP tree in Plücker space which encodes visibility using a set of 5-dimensional polyhedrons. Let P_S be the source polygon for which visibility needs to be evaluated. This visibility can be described by using all the rays emerging from P_S. For a polygon O, the occlusion volume is the polyhedron obtained by calculating the intersection of the half-spaces induced by the Plücker hyperplanes corresponding to the support lines of the edges of P_S and O. As previously noted by Nirenstein and Haumont, this polyhedron is unbounded. Thus, Bittner adds two cap planes, located one on each side of the Plücker quadric and aligned with it. The intersection of this resulting polytope with the Plücker quadric yields all the extremal stabbing lines of P_S and O.

As in the from-point approach, an approximate occlusion sweep is used to order the occluders in the scene. The algorithm processes incrementally each occluder, by inserting it in the occlusion tree. At each step the tree contains a representation of the lines blocked by the already processed geometry. The root of the occlusion tree represents the problem relevant line set (*i.e.* all the lines emerging from P_S). Each node represents a subset of lines. Also, a leaf represents a subset of lines which are either occluded or unoccluded. Each inner node is associated with a hyperplane, which is tested against each occlusion volume reaching the node. The occlusion volumes are split if they intersect the hyperplanes.

In the end, the occlusion tree represents all the lines emerging from P_S and grouped according to the first geometry they intersect.

Concerning the intersection of polytope with a hyperplane, Bittner's implementation is similar to the enumeration algorithm provided by [AF93]. Basically, if an intersection is detected, the hyperplane is added (twice, a second time with a reversed orientation) to the H-representation of the polytope. Then the vertices of the polytope are calculated, which yields the V-representation. This solution has the advantage of a simpler implementation when compared to the solutions provided by Nirenstein and Haumont. However, the computation time involved is more important, since all the vertices (not only the new ones) need to be re-calculated at each new step. Also, the method is less robust.

As with previous methods, all the polytopes obtained in the end need to be intersected with

the Plücker quadric in order to obtain only the real lines describing the visibility.

The complexity of the occlusion tree is up-bounded by the same theoretical $O(n^4 \log n)$. However, when compared with the PSP tree (see Section 1.4.9), the methods performs better, because the computation are limited to the region concerned by an occluder.

However, the method does not seem to behave very well in practice. The tests are limited to scenes composed of a few thousand random triangles, and the authors note that the splitting method is prone to numerical inaccuracies. Since the hyperplane-polytope intersection is a crucial point in the construction of the occlusion tree, several methods have been tested, included the algorithm of Bajaj and Pascucci [BP96], previously used by Nirenstein. However, all tests showed problems with numerical stability.

1.4.13 Exact Polygon-to-Polygon Visibility

Mora *et al.* [MAM05] constructs on Nirenstein and Bittner's approaches in order to propose a new method which describes the visibility between two convex polygons. They retain the CSG computations present in Nirenstein's algorithm, but output a data structure similar to the one proposed by Bittner. This information is used in image synthesis to render analytic soft shadows [MA05], and in electromagnetic waves propagation [Mor06].

Nirenstein's method is optimized in order to improve the visibility coherence. Mora [Mor06] remarks that some of the split operations performed by Nirenstein's algorithm are redundant and result in a high degree of visibility fragmentation. This has an impact on both the calculation times and the robustness of the solution. Thus, he proposes an enforced intersection test which limits this problems.

In order to encode the visibility information between two polygons, a BSP tree in Plücker space is proposed (noted *VBSP*). The authors note that this is however different from the occlusion tree proposed by Bittner because no depth information is taken into account. Thus, the order in which the occluders are processed can be optimized to improve the computations. Each inner node, n_i, of the VBSP tree contains a hyperplane and a polytope which represents the region of space partitioned by the sub-tree having n_i as root. Each leaf can be either visible or invisible. A visible leaf contains a polytope representing a set of lines not blocked by any occluder. Each occluder is hierarchically inserted into the tree and all the intersection calculations are done in the visible leaves.

The complexity of the method is inferior to the theoretical $O(n^4 \log n)$ bound. For two polygons and a set of about 9 000 random occluders, they achieve a memory complexity of $O(n^{1.76})$. In the case of 2.5-dimensional environment, the VBSP construction needs to be applied to each set of two polygons in the scene. The authors note that the complexity is bounded (*i.e.* there are two polygons for which the complexity is maximal).

For the two proposed applications (soft shadows and electromagnetic waves propagation), the occlusion algorithm is used as a pre-process step. The authors note that its two main drawbacks are the required calculation time and the robustness issues resulting from the 5D CSG operations.

The advantage of this method is that it avoids many redundant CSG operations, thus improving robustness. However, as the authors note their algorithm remains subject to numerical instability. The split operations are performed on a polytope similar to the one used by Nirenstein, under the same assumptions (one polygon needs to be contained in the $x = 0$ plane). Moreover, as with all the other 5D CSG operation-based methods, in the end of each split procedure, the intersection with the Plücker quadric needs to be tested. The authors note that it would be interesting to start with a minimal polytope and use it for all computations.

The proposed data structure encodes all the information between two polygons and the occluders contained in their convex hull. No depth information is being considered, since no distinction is being made between the sets of blocked lines according to the blocking geometry. Thus, the VBSP encodes the occlusion information between two polygons.

1.4.14 nD Visibility

In his PhD thesis [Cha07], Charneau studies the application of geometric algebras to the global visibility problem, in a projective n-dimensional space. His research provides a theoretical generalization of the visibility between surface elements of any dimension. The main difference between his approach and the previous works [Nir03, HMN05, Mor06] is the fact that he describes polygon-to-polygon visibility using an algebraic framework, valid in all dimensions. This allows him to define the geometric operations independently of the data representation or the dimension of this representation. As noted by the author, these theoretical definitions should allow the possibility to define robust and valid visibility algorithms in all dimensions, capable of treating all exceptions and degenerate cases in an exact and accurate manner. Moreover, in the 3-dimensional case, the line parametrization in this algebraic framework is equivalent to the Plücker parametrization.

An implementation is provided for the 3-dimensional case, and compared to the works of Nirenstein [NBG02, Nir03], Haumont [HMN05] and Mora [Mor06].

The most important (for the purpose of this study) result demonstrated by Charneau concerns the minimal polytope containing all the lines stabbing two convex polygons. As shown in Section 1.4.7, having an accurate and optimal representation of the lines in $stab(A,B)$ is crucial to the visibility calculations in Plücker space. Nirenstein [NBG02, Nir03], Haumont [HMN05], Bittner [Bit02] and Mora [Mor06] constructed a conservative polytope containing the relevant set of stabbing lines by bounding a polyhedron with two hyperplanes, on each side of the Plücker quadric. Charneau provides the solution to the minimal polytope problem and demonstrates that the solution cannot be achieved using two hyperplanes only. Moreover, he studies the possible degeneracies which can occur and provides exact and accurate solutions for each case.

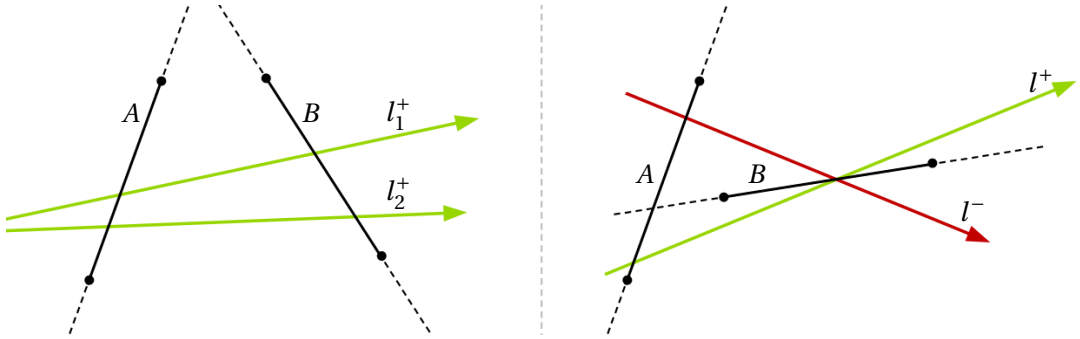


Figure 1.18: Left: If the support plane of polygon A does not intersect the polygon B (and vice versa), then all the lines stabbing both A and B have a coherent orientation. **Right**: A degenerate case. The supporting plane of polygon B intersects A. Thus, it is not possible to group together the lines stabbing both A and B in one convex polytope. The lines l^+ and l^- have opposite orientations.

We summarize here this important result, as it is crucial for our own implementations. All the concerned demonstrations, as well as further details can be found in [ACFM11] or [Cha07].

Let A and B be two convex polygons, and let H_A and H_B be their supporting planes respectively.

Theorem 2. *If H_A and H_B do not intersect the faces B or A respectively, then all the lines intersecting both A and B are contained in a minimal convex polyhedron defined in Plücker space (we note this $poly(A \rightarrow B)$). The vertices of this polyhedron are the Plücker points corresponding to the lines defined by one vertex of A and one vertex of B. The bounding hyperplanes are the Plücker hyperplanes corresponding to the support lines of the edges of A and B.*

Corollary 1. *Testing the position of $poly(A \rightarrow B)$ with respect to a hyperplane is equivalent to calculating the orientation of the lines stabbing A and B with respect to the hyperplane. If all the vertices of $poly(A \rightarrow B)$ have the same sign with respect to the hyperplane, the polyhedron lies completely in the positive or negative half-space defined by the hyperplane. Thus, all the stabbing lines have the same orientation. Otherwise, $poly(A \rightarrow B)$ is intersected by the hyperplane and the orientation of the lines stabbing A and B is no longer coherent.*

If one of the faces has an intersection with the support plane of the second face, it is not possible to group together the lines stabbing both faces. An illustration of both a valid and a degenerate case is provided in Figure 1.18. However, this particular case can always be handled by splitting the intersected face along the intersection with the support hyperplane of the second face.

1.5 Summary

Tables 1.1 and 1.2 summarize the various analytic methods which aim at computing from-point and from-polygon visibility.

Algorithm	Application	Domain	Space	Data structure
Amanatides [Ama84]	ray-set tracing	from-point visibility	3D	cone
Heckbert and Hanrahan [HH84]	ray-set tracing	from-point visibility	3D	beam
Shinya [STN87]	ray-set tracing	from-point visibility	3D	pencil
Ghazanfarpour [GH98]	ray-tracing anti-aliasing	from-point visibility	3D	beam
Bittner [BHS98]	PVS	from-point visibility	3D	occlusion tree
Stark [SR00]	irradiance calculation	from-point visibility	3D	none
Overbeck [ORM07]	GPU beam tracing soft shadows	from-point visibility	3D	beam KD-tree
Pocchiola and Vegter [PV93]	2D visibility complex	2D visibility	line space	2D visibility complex
Durand [DDP02]	3D visibility complex	3D visibility	3D to line	3D visibility complex
Durand [DDP97] Duguet [DD02]	3D visibility skeleton	3D visibility	line space	3D visibility skeleton
Teller [Tel92, TS91]	portal antipenumbra	portal visibility	Plücker	none
Mount and Pu [MP99]	visibility maps	from-poly occlusion	Plücker	PSP tree
Nirensetein [NBG02]	PVS	poly-to-poly occlusion	Plücker	none
Haumont [HMN05]	PVS	poly-to-poly occlusion	Plücker	none
Bittner [Bit02]	PVS	from-poly visibility	Plücker	occlusion tree
Mora [MAM05, Mor06]	soft shadows electromagnetic wave propagation	poly-to-poly occlusion	Plücker	VBSP tree

Table 1.1: Summary of analytic visibility algorithms.

Algorithm	Drawbacks
Pocchiola and Vegter [PV93]	limited to 2D case, incomplete implementation
Durand [DDP02]	no practical implementation
Durand [DDP97]	robustness, degeneracies
Duguet [DD02]	ϵ precission
Teller [Tel92, TS91]	cannot handle occluders
Mount and Pu [MP99]	5D CSG, complete arrangement, worst complexity
Nirensetein [NBG02]	5D CSG, computation time, proof of visibility only
Haumont [HMN05]	5D CSG proof of visibility only
Bittner [Bit02]	5D CSG, computation time
Mora [MAM05, Mor06]	5D CSG, computation time

Table 1.2: Summary of the main drawbacks of the analytic from-polygon visibility algorithms.

1.6 Conclusion

Exact from-polygon visibility is a complex, four dimensional problem, which can be expressed elegantly in terms of lines. The continuous sets of lines which represent the visual events in a scene, are generated by the 3-dimensional geometry of the scene. Therefore, we can distinguish between the methods which represent and solve visibility in object space and in line space.

The first class of methods concerns the visibility complex [DDP02] and the visibility skeleton [DDP97]. However, the first one is to complex to allow an implementation, and the second one represents only a partial information of visibility. From a theoretical point of view, these two techniques are intuitive because they provide an explicit representation of the relation between the geometry and the visibility events generated by this geometry. However, in practice, they use a 3-dimensional description of objects in order to represent 4-dimensional relations. Therefore, to our knowledge, a complete and accurate implementation does not exist.

These two solutions underline even further the need of a solid mathematical framework allowing working with continuous sets of lines.

Such a representation is provided by the Plücker parametrization. In Plücker space, continuous sets of lines can be described using convex volumes, which are easier to manipulate. Moreover, using the Plücker parametrization, both the geometry and the visibility relations it generates can be represented as continuous sets of lines. This motivated a series of algorithms which encode from-polygon visibility in $\mathbb{P}^5$.

Using the Plücker parametrization, Pellegrini [Pel91, Pel04] describes a theoretical framework which allows an exact and analytical representation of all the sets of lines generated by a set of triangles. He groups lines into equivalence classes and provides theoretical bounds for the calculated arrangement. This theoretical framework, together with the Plücker parametrization, represent a powerful tool which can be used to solve global visibility problems. However, from a practical point of view, the existing implementations are limited in their application and suffer from important drawbacks.

Although the existing methods are not directly based on the theoretical framework described by Pellegrini, they all calculate a sub-part of an arrangement of lines in Plücker space.

Teller provides a solution to the line-polygons intersection problem and applies it to portal visibility and global illumination calculation (see 1.4.8). His algorithm determines if a sight line exists for a set of portals. This translates to the calculation of one single cell from an arrangement of lines: the cell corresponding to the equivalence class containing all the sight lines which traverse the given sequence of portals.

Mount and Pu attempt to use the Plücker parametrization to calculate the classes of blocked lines from a source polygon. Their algorithm computes and stores the complete arrangement of hyperplanes generated by the scene's geometry. Therefore, their method is too close to the theoretical complexity bound and fails in practice (see 1.4.9).

The methods proposed by Nirenstein and Haumont focus on polygon-to-polygon occlusion and provide algorithms which are used in the context of PVS determination (see 1.4.10 and 1.4.11). Both methods output a boolean result. Contrary to Mount and Pu, Nirenstein and Haumont limit their calculations to a sub-part of theoretical arrangement of lines. They only calculate those cells corresponding to the sets of lines stabbing the two polygons considered for each query.

Contrary to Nirenstein and Haumont, Bittner encodes the visibility from a source polygon into a BSP tree in Plücker space (see 1.4.12). Although bounded by the same theoretical complexity, the method performs better than the one proposed by Mount and Pu, due to a better implementation.

In order to take advantage of both Nirenstein and Bittner's algorithms, Mora proposes a method which calculates and stores all the occlusion information between two polygons (see 1.4.13). Again, the calculation is limited to a sub-part of the theoretical arrangement of lines.

All these methods rely on 5D CSG operations in Plücker space. As noted by the authors themselves, the results are thus subject to robustness issues and numerical instabilities. Moreover, the complexity of the operations has a negative impact on the computation times. Thus, all these methods are only used as pre-process steps.

Another problem concerns the definition of the 5-dimensional polytope used to characterize the lines between two polygons. As underlined by Charneau [Cha07], none of these methods use the minimal polytope which contains all the lines stabbing two convex polygons. This results in redundant 5D CSG operations and an intersection test with the Plücker quadric is always necessary.

In conclusion, the Plücker parametrization together with the 5-dimensional arrangement of lines described by Pellegrini provide an elegant theoretical framework. However, there is an important gap between the theoretical results and the existing implementations. The existing methods suffer from important drawbacks and thus are unable to fully exploit the potential of the theoretical framework.

Therefore, we start from Pellegrini's framework and intend to propose a method which fills the existing void between the theoretical tools and the existing implementations. Our objective is to develop a robust and accurate algorithm which calculates exact from-polygon visibility in a given applicative context. Although we are bounded by an important theoretical complexity, we aim to demonstrate that we can achieve practical and efficient algorithms.

The two main contributions of this thesis can be outlined as follows:

First of all, we want to provide a robust and accurate solution to the *from-polygon occlusion* problem. Our method considers a simplification of Pellegrini's framework, and analytically distinguishes between the lines which are blocked by some geometry and those which are not. In order to implement a numerically stable algorithm, we need to avoid all 5D CSG operations. This is achieved by a conservative insertion process, and a 5-dimensional solution operating only on real lines, which is based on Charneau [Cha07]'s study on the minimal polytope. In order to test the robustness and the accuracy of our method, we apply it in the context of soft shadows generation. This is presented in Chapter 2.

Secondly, we want to provide an answer to the *from-polygon visibility* problem. More exactly, we build on the same concept of equivalence classes and develop a new definition which analytically groups the rays issued from a polygon according to the first triangle they intersect. This allows us to propose an exact and robust solution to the *from-polygon visibility* problem. The applicative context for this second algorithm is the calculation of analytic ambient occlusion. This second algorithm retains all the advantages of the first implementation. This is presented in Chapter 3.

Both algorithms encode the calculated information using a BSP tree in Plücker space. These data structures are build on demand at run time, when and where the information is required, thus avoiding all pre-process calculations.

It is important to note that our algorithms encode the exact from-polygon occlusion (first solution) and visibility (second solution), and use this information to efficiently solve from-point individual queries, by taking advantage of the visual coherence which exists between neighbor points.

Chapter 2 :

From-Polygon Occlusion Application to Soft Shadows

IN this chapter we present our new method which calculates the exact and analytic visibility of a polygon, as seen through a set of occluders. This information is used to take advantage of the visual coherence which exists between neighbor points and generate accurate and noise-free soft shadows of high quality.

Calculating soft shadows is essentially a point-surface problem, because it requires the visibility of the light source, as seen from each point in the scene. However, in order to take advantage of the visual coherence which exists between neighbor points, we are faced with a surface-to-surface occlusion problem. Therefore, the aim of our method is to calculate the occlusion information between surfaces and use it to answer point-to-surface occlusion queries. This will allow the algorithm to take advantage of the visual coherence.

From a theoretical point of view, the starting point of our method are the equivalence classes described by Pellegrini, which allow grouping lines according to the geometry they intersect. Contrary to this theoretical framework, our new algorithm only distinguishes between occluded and unoccluded lines, thus representing a simplification of the initial concept.

With respect to the previous works which rely on calculating an arrangement of lines in Plücker space, our new technique avoids all 5D CSG operations, thus being accurate and robust.

In the context of off-line rendering, we use our algorithm to analytically calculate the visibility of the light source as seen from every point to shade, and compute the direct illumination for these points. Since the soft shadows are calculated only for the visible points from the camera, we design our algorithm to perform the computations lazily at run time, when and where the visibility information is required.

The first part of this chapter details some related work on high quality soft shadows (Section 2.1), followed by a description of our new method, from a theoretical point of view (Section 2.2). Next, we present the practical aspects of our implementation (Section 2.3), and the soft shadows framework we use (Section 2.4). The final sections focus on the results we obtained (Section 2.5) and a discussion of our method (Section 2.6).

The work presented in this chapter has been published as

Lazy Visibility Evaluation for Exact Soft Shadows
F. Mora, L. Aveneau, O. Apostu, D. Ghazanfarpour
in
Computer Graphics Forum,
Volume 31, Issue 1, pages 132–145, February 2012

2.1 Soft Shadow Generation

Generating soft shadows is a topic that has received increased attention over the years. An extensive survey on real time soft shadows algorithms has been provided by Hasenfratz *et al.* [HLHS03], and a more recent synthesis can be found in [EASW09] and [ESAW11].

Soft shadows are the result of objects partially blocking the light coming from area light sources. Figure 2.1 illustrates the difference between hard and soft shadows, which appear in the presence of point and area light sources, respectively.

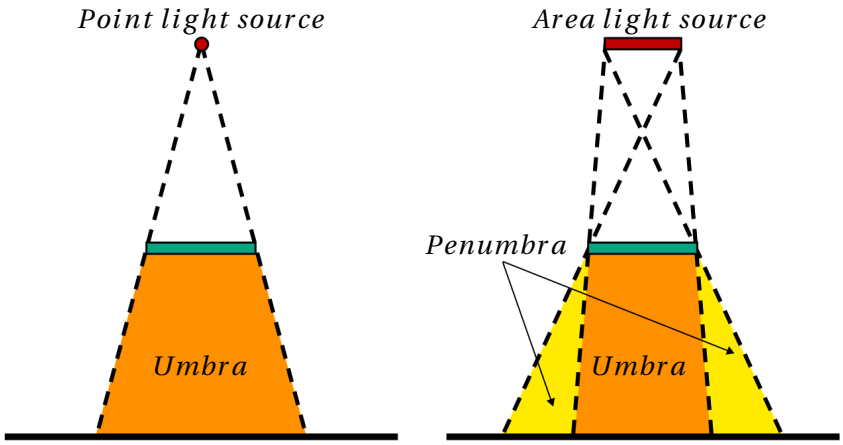


Figure 2.1: Left: Point light source generate hard shadows that consist of umbra. **Right**: Area light sources (or any volumetric light source) generate soft shadows which consist of an umbra (light source is invisible), and a smooth transition, the penumbra (light source is partially visible).

Calculating the amount of soft shadow for a point is equivalent to determining the direct illumination value for the point, which is represented by the radiance reaching the point directly from the light sources. Radiance can be seen as the color perceived by the human eye, and is defined as the radiant flux per unit solid angle per projected area. The direct illumination can be derived from the *rendering equation* [Kaj86], and is expressed as follows:

$$L_{direct} = \int_{\Omega} vis(x,x') f_r(x,\omega_i,\omega_r) E(x,\omega_r) \frac{\cos\theta_i \cos\theta_r}{||x-x'||^2} d\omega_i \tag{2.1}$$

Where:

- x is the point of interest
- x' is a point on the area light source
- ω_i is the direction of the incident radiance
- ω_r is the direction of the emitted/reflected radiance

- $||x - x'||^2$ is the distance between x' and x
- Ω is the integration domain
- θ_i is the angle between the normal vector and the direction of an incident light
- θ_r is the angle between the normal vector and the direction of an emitted light
- $f_r(x, \omega_i, \omega_r)$ is the proportion of light reflected at x (from inward direction to outward direction)
- $E(x, \omega_r)$ is the emitted radiance in direction ω_r
- $vis(x, x')$ is the visibility function which equals 1, if x' is directly visible from x, and 0 otherwise.

In order to compute an accurate direct illumination, the integral needs to be solved for the surfaces of the light sources. The main step consists in integrating over all the visible fragments of these light sources, as seen from the point to shade. In the case of uniformly emitting diffuse surfaces, a closed form evaluation can be achieved using Lambert's formula [Lam60].

In practice, the majority of soft shadow algorithms separate the visibility and the lighting calculations. Moreover, they usually approximate the visibility of the light sources and/or the amount of direct illumination. In this section, we distinguish mainly between sampling-based, analytic and approximated methods. From a mathematical and physical point of view, solving the direct illumination integral using a sampling based technique is an exact solution, when the number of samples tends towards infinity. However, from a practical point of view, this is an impossible scenario, since the number of samples is always finite. Thus, in practice, we can say that the complete visibility is approximated using the visibility of the selected samples. Because of this, the resulted images are subject to noise. On the other hand, analytic methods deal with continuous representations of lines and surfaces, and thus avoid noise. However, any approximation in the visibility calculations usually results in visual artifacts. It is important to note that in the majority of the current methods there is a fine line between all these distinctions. Some methods use analytic representations of the shadow volumes, but sample the light in the end in order to solve the direct illumination integral, while other techniques are based entirely on either sampling or analytic and exact calculations.

In this section we focus on various methods which compute high quality or exact soft shadows. We start by a short analysis of the ray traced soft shadows, followed by a description of some analytical methods. We then present silhouette based techniques, which can be seen as a crossing point between sampling-based methods and analytical approaches. Finally, we mention briefly some of the recent algorithms capable of producing high quality soft shadows, despite the approximations made to reach interactive frame rates.

2.1.1 Sampling-based Methods

Soft shadows can be easily generated using ray tracing. This consists of two main steps. First of all, the light source is sampled. Next, rays are traced for each point and all the light samples. These rays are

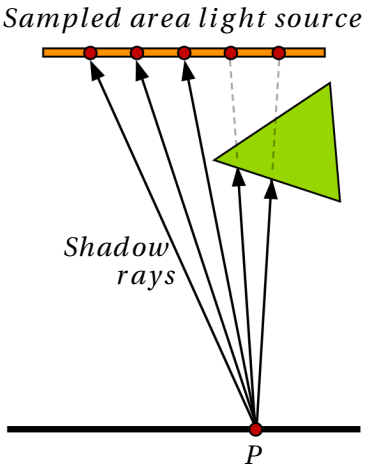


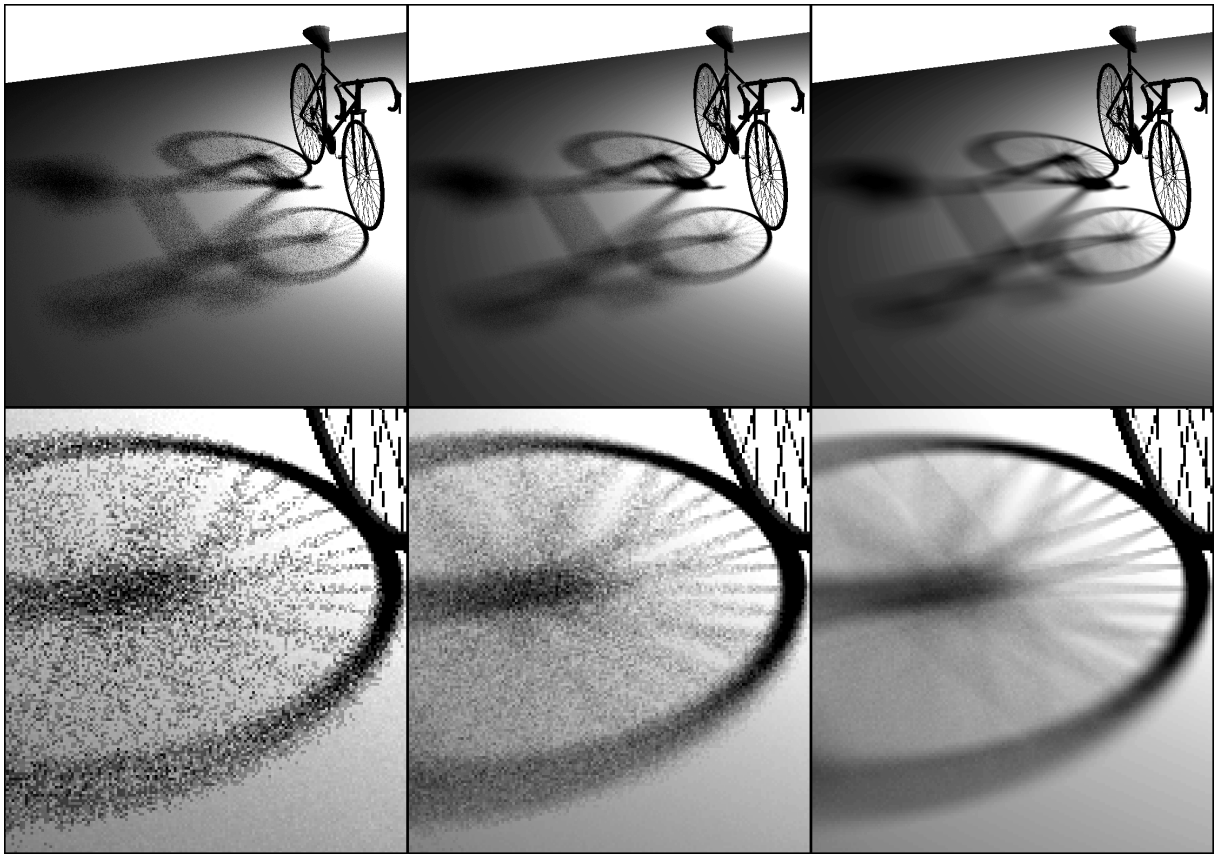
Figure 2.2: 2D illustration of a sampling based method. P is the surface point to shade. The shadow rays emerge from P and have as directions the sample points on the area light source. The rays reaching the light without intersecting any object are counted as visible.

usually called *shadow rays*, and their purpose is to test if the incoming light is blocked by an object. Figure 2.2 provides an illustration. Summing all the shadow rays reaching the light without being blocked yields an approximation of the visible percentage of the source, as seen from the point. In order to calculate the soft shadows value for the point, this value needs to be weighted by the intensity of the light source. Thus, both the visibility and the incoming intensity are approximated. Another solution consists in considering each sample as an independent point light source. In this case, the soft shadows calculation uses the cosine-weighted intensity of each visible sample.

Ray traced soft shadows is a robust solution, which can be applied to all types of light sources. Moreover, it can be easily integrated in any ray tracing based rendering engine. Therefore, it represents today one of the most common solution in production rendering.

On the other hand, ray traced soft shadows have the same disadvantages as any other sampling-based method. The quality of the final images is dependent on the number of rays and the results are sensitive to noise. In order to attenuate this, the number of samples needs to be increased. However, this has a negative impact on performance, by drastically slowing down the rendering process. Figure 2.3 provides an illustration of how the number of samples impacts on the quality of the final result.

All these drawbacks are common to ray tracing in general, both for primary and secondary rays. Therefore, an important amount of research has been done in order to propose various solutions which may improve both the rendering performance and the quality of the results.



4 *rays / pixel* 16 *rays / pixel* 256 *rays / pixel*

Figure 2.3: When generating soft shadows using ray tracing, the number of samples has an significant impact on the quality of the result. An important number of samples is necessary to attenuate the noise in the final results.

Several research have attempted to exploit spatial coherence by grouping rays into *packets* and thus amortizing computational costs over the entire packet. Wald *et al.* were the first to propose a coherent ray tracing method [WSBW01], which traces groups of four rays through a kd-tree and uses SIMD instructions to process the rays in parallel. Another optimization was the use of frustum or interval arithmetic based methods. The rays are grouped into packets of variable sizes and the bounds of the packet are used to avoid traversal steps or object intersections. The concept was introduced by Reshetov *et al.* [RSH05], who applied it to a kd-tree, and was later extended to grids [WIK+06] and BVHs [Wal07, WK06]. These ray aggregation techniques can be successfully implemented for shadow rays, because these sets of secondary rays have a high spatial coherence.

Another optimization consists in improving the sample distribution. Interleaved sampling [KH01] is a hybrid method between regular and irregular sampling, which uses an arbitrary irregular sampling pattern, repeated to form a regular grid. Keeping regular grids allows an easy exploitation by raster graphics hardware, while aliasing is reduced by the fact that neighbor pixels have different sampling patterns. A second technique, importance sampling [SWZ96, SSSK04, wik12b], is based on

sampling some parts of the light source more than others, according to their importance, which is defined based on previous knowledge of the visibility function. A comparison of different sampling techniques used in the context of soft shadows can be found in [FBP08].

Depending on the optimization technique and the type of scene, we can obtain the same quality results while tracing fewer shadow rays. However, the images remain sensitive to noise. Therefore, a common noise reduction technique consists in applying a post-process filter to the results. Durand *et al.* [DHS+05] propose a theoretical framework designed to provide a complex frequency analysis of the radiance function. The authors state that a better understanding of the light transport would allow for more efficient sampling approaches. They describe an algorithm which computes visibility using a low number of samples per pixel and then determines the parts of the image which need to be reconstructed using filtering. A similar method is proposed by Egan *et al.* [EHDR11]. They target the generation of soft shadows produced by intricate geometries, which usually require a prohibitive number of samples to attenuate noise. A Fourier analysis of the shadow signal is used to create a filter which is customized to the frequency content of the shadow. The algorithm can be divided into three main steps. First of all, a small number of rays is used to roughly sample the occlusion in a 4D light field. The 4D parametrization of each ray is stored into a database. Next, for each receiver point, a programmable shader construct the shape of the appropriate filter, and uses the database to find all the samples inside the filter's 4D footprint. These samples are then weighted using the filter, in order to reconstruct the shadow value.

Ramamoorthi *et al.* [RAMN12] provide a comprehensive analysis of the effectiveness of different non-adaptive sampling patterns for rendering soft shadows. They consider linear and area light sources and focus mainly on visibility sampling. For linear light sources, the analytic analysis indicates that uniform sampling has the lowest maximum error rate, although it produces banding artifacts. In order to avoid them, an uniform jitter sampling can be used instead. This applies a constant jitter to all samples in a uniform pattern. In the case of area light sources, the authors show that the best sampling method depends on the type of light source (circular, gaussian, or square/rectangular). The theoretical analyses are completed by practical tests, executed on two separate platforms, an off-line and a real-time engine: RenderMan, and NVIDIA's Optix GPU ray-tracing API, respectively. The authors note that the guidelines resulting from their analysis may not yield the best results in the case of very complex geometry. In this cases, a method such as the one proposed by Egan *et al.* [EHDR11] would give better results.

Despite various improvements, rendering high quality soft shadows using a ray traced approach remains a challenging task.

2.1.2 Analytic Methods

Beam tracing methods (see Chapter 1, Section 1.3.2) represent an analytic solution to the point-to-surface visibility problem. Most of these methods date from a period (1984 – 1998) when the computers' performances limited the interest towards sampling solutions. Todays' hardware makes

possible tracing several hundreds rays per pixel in order to attenuate the noise in the final images. However, this was not always the case. Thus, the researchers have oriented their efforts towards the beam tracing methods, in order to remove noise, and solve some aliasing problems.

In the context of soft shadow generation the main advantage of beam tracing is that it achieve noise free results, due to the analytic representations of the visibility of the light sources. However, beam tracing have received limited attention from the rendering community for many years, because of their inherent limitations. More exactly, they suffer from two main drawbacks. First of all, the geometry of the beam is more complex than the simple ray, and thus more complicate to manipulate. The basic geometry intersections become significantly more complicated when the classic rays are replaced with beams. Also, many algorithms subtract the encountered polygons from the beam, whose shape becomes more and more degenerate. Therefore, beam tracing algorithms generally have an increased computational cost. Secondly, the classic ray tracing acceleration structures cannot be easily applied to beams, which limits the possible optimizations.

The method proposed by Amanatides [Ama84] (see Chapter 1, Section 1.3.2) replaces each ray with a cone, which contains exactly one pixel. Soft shadows (or *fuzzy shadows*, as called by the author) are calculated by replacing the shadow rays with cones which contain the entire light source. The method is restricted to spherical lights, non transparent surfaces and does not handle multiple occlusions. Therefore, the visibility is approximated.

The *beam tracer* proposed by Heckbert and Hanrahan [HH84] (see Chapter 1, Section 1.3.2) considers a view beam defined by the viewpoint and the projection screen, which is successively intersected with the objects in the scene. The authors apply the same principle to the generation of soft shadows, with the difference that the *light beams* are traced from the light sources instead of the viewpoint. Each beam of light will therefore contain the entire scene. Although the visibility of the light source is analytically determined, no information is given on how the direct illumination values are calculated. The authors note that if the light sources are infinitely distant, a constant diffuse intensity is associated with each face.

Ghazanfarpour *et al.* [GH98] (see Chapter 1, Section 1.3.2) provide a hybrid method, based on an adaptive recursive subdivision of the image space. A vision beam is traced through the entire screen and recursively subdivided until either a uniform region is obtained or the pixel subdivision limit is reached. The method handles point and spherical light sources. Shadows are generated by constructing beams from the light sources, as illustrated in Figure 2.4. In order to compute soft shadows, the algorithm detects the regions which are either uniform (entirely in shadow or entirely lighted), or located in the soft shadow zone. In this latter case, the beam is recursively subdivided until an uniform region is obtained or the pixel subdivision limit has been reached. If the region is greater than one pixel, four corner rays are traced for each pixel. The total illumination for each pixel is then calculated as the sum of its four sub-pixels. If the region is smaller than one pixel, a single ray is traced through the middle of the sub-pixel. The direct illumination of a sub-pixel is calculated

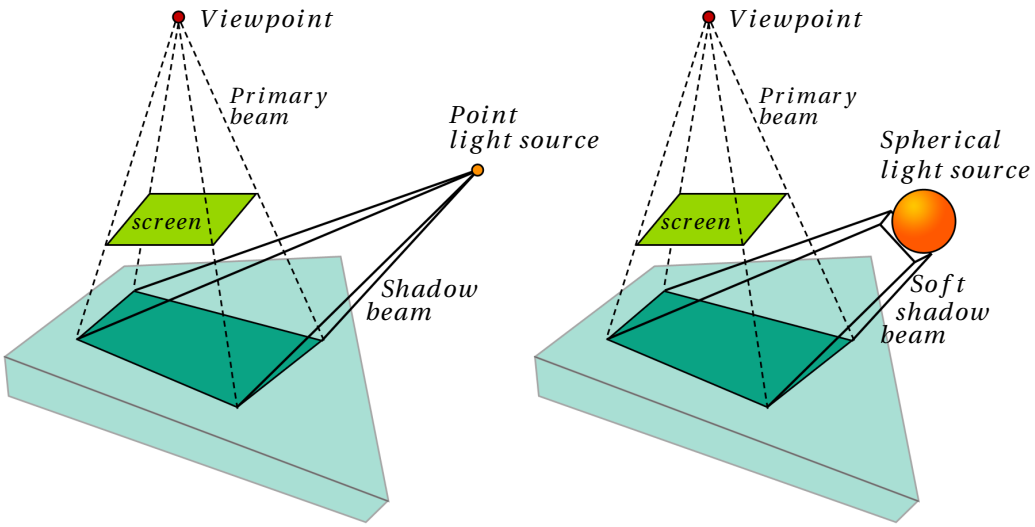


Figure 2.4: Hard and soft shadow generation using beam tracing [GH98]. **Left**: Shadow beam for a point light source. **Right**: Shadow beam from a spherical light source.

using a point light source placed at the center of the initial spherical source. Therefore, the initial light source is only used to detect the different shadow regions, and the final illumination value is approximated. As noted by the authors, the obtained soft shadows are realistic, but not exact.

Overbeck *et al.* [ORM07] (see Chapter 1, Section 1.3.2) propose a method which applies kd-traversal for beam tracing and improves beam - triangle intersection calculations. The shadow beams are defined by the points to shade and the light area source. They are used to solve exact visibility of the light from the point to shade. The irradiance from the visible portion of the light is then analytically calculated for the point to shade. The method provides accurate and high quality results, but is limited to scenes with moderate complexity. In the case of soft shadows, the performed tests only deal with less than 10 visible triangles for each pixel to shade, and thus for each shadow beam. Beam tracing methods scale linearly with the visible triangles. Thus, the authors note that their algorithm is more efficient for shadow beams, than for primary visibility, because there are less triangles to be taken into account.

In his PhD thesis, Stark [Sta02] provides a comprehensive study of analytic techniques for evaluating partially-occluded irradiance in polygonal environments. The first results concern the calculation of exact soft shadows, by representing emitters and occluders as generalized perspective prisms in 4D. The method, detailed in [SCLR99], is limited to single occluders or to multiple occluders whose shadows do not interact. A more general approach is provided in [Sta02], but the authors note that its interest is rather theoretical than practical. Their third analytic contribution [SR01] is the use of the vertex tracing algorithm (see Chapter 1, Section 1.3.3) to calculate exact soft shadows in arbitrary polygonal environments. The method computes the exact visible parts of an area light source as seen from the point to shade, and then analytically calculates the irradiance of each visible fragment, using

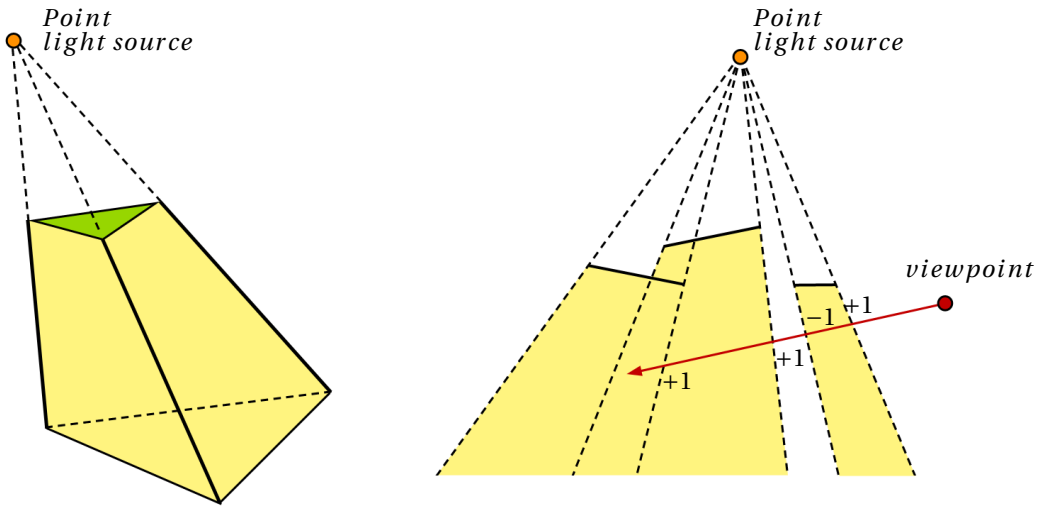


Figure 2.5: Shadow volumes according to Crow [Cro77]. **Left**: The shadow volume for a given triangle. **Right**: Each time a viewpoint ray enters a shadow volume, a counter value is incremented. Exiting a shadow volume is marked by a decrementing the same value. Therefore, deciding if the hit point is in shadow or not is equivalent to checking the value of the counter.

a formulation derived from Lambert's formula [Lam60]. Aside the numerical instability described in 1.3.3, the performance of the algorithm also suffers when there is a lot of shadow interaction.

Mora [Mor06] provides a method which calculates accurate and high quality soft shadows, based on an algorithm which computes the exact occlusion information between the area light source and the surfaces in the scene. More exactly, this information is encoded in a 5D BSP tree (see Chapter 1, Section 1.4.13) for each set composed of a polygon and the light source. After this pre-process step, the exact visibility of the area light source is extracted for each point to shade. Since all the exact occlusion information is calculated and stored for the entire scene, independently of the viewpoint, the required times are an important limitation. Moreover, this exact visibility is calculated using complex 5D CSG operations, which subject the results to numerical instabilities and robustness issues.

It is important to note that our algorithm represents a continuation of the work of Mora [Mor06]. We also compute the exact visibility information between a polygon and a light source, in order to exploit the visual coherence and accelerate from-point queries. However, our aim is to provide a solution to all the robustness issues which are characteristic of all the previous work on from-polygon visibility, and in particular of the soft shadow method provided by Mora. Thus, our new is designed to avoids all 5D CSG operations, and to computes only the necessary information, during run time, without any additional pre-process steps.

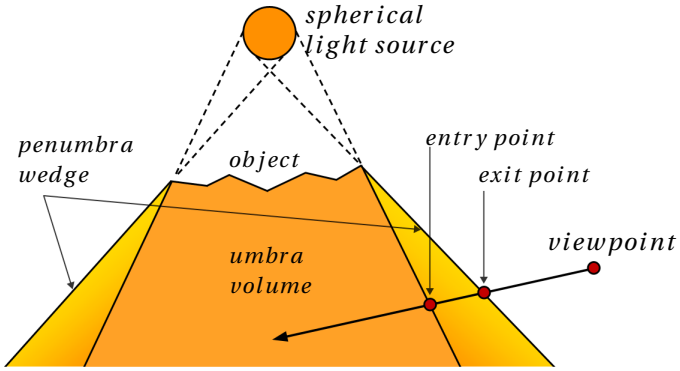


Figure 2.6: Soft shadow volume with penumbra wedges [AMA02].

2.1.3 Silhouette Based Soft Shadows

The shadow volume and *soft shadow volume* techniques propose a crossing point between analytic and sampling based methods. These method use an analytic description of the shadow and the penumbra volumes. Also, the majority of these algorithms are object based, and therefore the visibility computations are accurate. However, the light source is still sampled and thus aliasing problems remain an issue. Moreover, the various approximations and simplifications which are made in order to achieve real time rendering compromise even further the quality of the results.

In the context of hard shadows, Crow [Cro77] defines the shadow volume of a point light source and an object. The proposed algorithm constructs this volume by considering the planes formed by the light and the silhouette edges of the polygons in the object. A silhouette edge is an edge shared by two polygons, one oriented towards the light, and a second one oriented away from the light. During rendering, the algorithm counts for each ray if it has entered and exited a shadow volume, thus determining if the hit point needs to be shaded or not. Figure 2.5 gives an illustration.

Akenine-Möller and Assarsson [AMA02] propose a soft shadow algorithm, which builds on the shadow volume method, proposed by Crow [Cro77]. They consider light sources to be spheres instead of points, and each silhouette edge as seen from the light source defines a *penumbra wedge*. The penumbra volume created by these wedges represents an approximation of the soft shadow volume. Figure 2.6 provides an illustration. The light intensity computation for each point is also approximated, using linear interpolation. As stated by the authors, all these approximations are motivated by the aim to achieve the best compromise between quality and rendering speed. The method is suited for objects which have really simple silhouette edges, and robustness issues arise in some particular scenarios (for example, silhouette edges which are parallel with the direction of the incoming light). These cases are treated by removing the edges causing problems. However, this solution has a negative impact on the quality, since the accuracy of the geometry is no longer preserved and visual artifacts arise.

Assarsson and Akenine-Möller [AAM03] address these limitations in a GPU method, intended for real time rendering of dynamic scenes. Although the penumbra wedge construction algorithm is improved and some robustness issues are solved, the method continues to sacrifice accuracy for speed. In order to construct soft shadow volume, they consider the silhouette edges as seen from a single point on the area light source, which results in visual artifacts. The penumbra volumes are an approximation of the real soft shadow volumes, but the points inside the wedge but outside the real penumbra do not affect the visibility calculations. Another visual artifact results from overlapping objects. All objects are treated independently and therefore, two objects which overlap as seen from the light source will cause an overestimation of the shadow. The authors note that these issues are acceptable in contexts such as games, or other real time applications.

The soft shadow volume algorithm is extended to off-line ray-tracing by Laine *et al.* [LAA+05]. The authors propose a two step method which constructs a conservative and analytical representation of the silhouette edges that overlap the area light source, as seen from the point to shade. The algorithm uses a hemicube, which stores the penumbra wedges for all the silhouette edges. During rendering time, for each point P, the wedges that may contain P are recuperated from the hemicube, and their corresponding silhouette edges are projected on the surface of the light source, in order to reconstruct the *visibility function*. The light source is sampled and the visibility function indicates, for each sample, if the sight-line from P is blocked or not. The algorithm takes into account all the silhouette edges and handles overlapping cases correctly. However, the use of a hemicube exposes the method to several drawbacks. The algorithm is overly sensitive to the size of the scene and the orientation of the light source.

Lehtinen *et al.* [LLA06] provide a complete analysis of the previously described method [LAA+05], and propose an improved algorithm. Their main contribution is replacing the hemicube with an axis-aligned 3D BSP tree which covers the entire scene. The data structure is no longer constructed as a pre-process step, but built lazily at run time, as directed by the queries. The method calculates the exact visibility between each point to shade and all the samples chosen on the area light source. The algorithm manages to improve the solution of Laine *et al.* [LAA+05] both in terms of efficiency and memory consumption.

Forest *et al.* [FBP08] build on the works of [AAM03] and [LLA06]. For an area light source, the visible points located in the penumbra region are identified using penumbra wedges, similarly to [AMA02, AAM03]. A new evaluation of the visibility function is proposed, based on a dedicated implementation on graphics hardware. For each light source, the algorithm identifies the points located in the penumbra region, and uses a set of light samples to evaluate the visibility of the source, as seen from the point. Next, the direct illumination is calculated by solving the direct illumination component of the rendering equation [Kaj86] for a set of uniformly distributed samples. The algorithm achieves a rendering performance of several frames per second on moderate scenes and using an average of 16 light samples per area light source. However, as the authors state it, the method suffers from precision errors and robustness issues, which result in various artifacts. Also,

since the lights are sampled, the results are also subject to sub-sampling artifacts.

2.1.4 Other Methods

Laine and Aila [LA05] change the pre-processing order of ray tracing. Instead of finding a triangle hit by a shadow ray, they search all the shadow rays intersecting a given triangle. Two hierarchies are maintained simultaneously, one for the points to shade and one for the light samples. The algorithm starts by determining the visible surfaces from the viewpoint of the camera, and constructing the receiver points and the light samples hierarchies. Next, all the blocker triangles are processed. Each one of these triangles is associated with penumbra volume, which is used to reject the receiver points which are not affected by the current blocker. Thus, only the points affected by the current triangle are shaded with the appropriate value. No acceleration structure is built for the scene's geometry. Thus, the authors state that the method is particularly convenient for very large scenes and dynamic environments, where maintaining a coherent acceleration structure between frames represents an important cost. The algorithm is dependent on the output resolution and the number of light samples. However, increased area light sources has a negative impact on the performance, and the memory complexity consumption is more important than in the case of a ray tracing with a standard acceleration structure. Thus, the authors note that one algorithm may perform better than the other, according to the scene used.

Eisemann *et al.* [ED07] propose an algorithm which samples the visibility between two surfaces. The entire method is implemented on the GPU, and the triangles are treated in a similar way as [LA05]. An application to soft shadows is provided. Let S be a source and R a receiver. The algorithm traverses all the triangles and all the samples of R which are potentially affected by the current triangle, in order to find the samples of S which are hidden by the current triangle, as seen from the samples of R. In order to optimize this step, the viewpoint of the rendering can be chosen to coincide with the receiver. In order to compute the samples which are potentially affected by a triangle (called the triangles's *influence region*), penumbra wedges are used. The resulted set is conservative, and the implementation can be further optimized if an axis-aligned bounding quadrilateral is used instead of a generic receiver. The triangle is back-projected onto the source and the blocked source samples are calculated by a fragment shader. The algorithm can handle correctly very large sources that are either rectangular or simply planar polygons. However, in the latter case, a bounding rectangle approximation is used. The method approximates both visibility and direct illumination computations and takes full advantage of the GPU implementation, in order to achieves interactive frame rates, while preserving visually pleasing soft shadows. As an example, a single model composed of $4K$ polygons is rendered in less than 0.03 seconds at a 512×512 resolution. Since all triangles are treated independently, the overall performance of the algorithm suffers in the case of a highly tessellated model.

A similar GPU method is presented by Sintorn *et al.* [SEA08], which can handle arbitrary shaped area or volumetric light sources. Instead of using a back-projection, blocked source samples are calculated using a frustum defined by the triangle's edges and the source sample. The visibility calculations

are decoupled from the shading operations, which are implemented using a Phong illumination model and bent vertex normals with respect to adjacent faces. The method is significantly faster when compared to the shadow volume presented by Laine *et al.* [LAA+05] and to the beam tracer developed by Overbeck *et al.* [ORM07]. However, as the authors note, their solution is sampled, instead of the exact visibility calculations provided by Overbeck *et al.*.

2.1.5 Conclusion

Generating soft shadows is a complex process, which consists of two main operations: calculating the visibility of the light source as seen from the point to shade, and computing the direct illumination value for the point in question. In practice, these two steps are often approximated. An accurate and physically correct algorithm would have to detect the exact fragments of the light source and then calculate the shading using an analytic formulation. It is important to note that a closed form solution to the direct illumination integral only exists for uniformly emitting light sources.

In the previous sections we have reviewed a series of methods which attempt to generate high quality soft shadows. We have chosen to divide them into analytic and sampling-based algorithms. The latter ones suffer from noise because the visibility is sampled, and require an important number of shadow rays in order to attenuate this problem. However, they remain an industry standard for off-line rendering, mainly for their robustness and simplicity. On the other hand, the analytic methods are noise free, but prone to numerical instabilities and robustness issues, due to the complex intersections which need to be calculated. We have also detailed silhouette based soft shadows, which can be seen as a crossing point between the analytic and sampling methods. These methods use analytic descriptions of the shadow and penumbra volumes, but sample the light source. Thus, sampling problems remain an issue, and visual artifacts may also arise.

Initially, beam tracing methods were created to take advantage of the spatial coherence which exists between the rays sharing a common origin. However, dealing with volumetric structures instead of simple rays has proven to be a very complex task. The idea was later adapted to single rays and led to the development of packet tracing, a technique which groups rays into packets and amortizes the computational cost over the entire packet. However, none of these methods exploit the visual coherence which exists between the visibility queries originating from the points belonging to the same surface. This is mainly because computing from-polygon visibility is by nature a 4-dimensional problem, and thus much more complex than from-point visibility. An attempt in this direction is made by Mora [Mor06], which designed an algorithm capable to encode the exact occlusion information between two surfaces as a pre-process step. This information is then used during run time to accelerate the independent queries for the points to shade. However, his approach suffers from numerical instabilities and robustness issues, due to the complex 5D CSG operations involved. These operations have a prohibitive cost and limit the application of the algorithm to scenes of small to moderate size and complexity.

Therefore, we can conclude that the current implementations are still far from an analytic solution

which could successfully exploit the visual coherence between the points to shade, and which would yield accurate and high-quality soft shadows.

2.2 Algorithm Design

In this section we present our from-polygon occlusion algorithm from a theoretical point of view.

The applicative context of our algorithm is the generation of exact and analytic soft shadows. In this context, we consider a planar light source, S, and an arbitrary polygon T, which contains the points to shade. We note $O(S,T)$ all the potential occluders of S and T.

The aim of our new method is to calculate the occlusion information related to S, as seen from T through the geometry in $O(S,T)$, and use it to speed up from-point occlusion queries. This will allow the algorithm to take advantage of the visual coherence which exists between the occlusion queries.

The starting point of our new method is the theoretical framework described by Pellegrini. First of all, we explain how we adapt the *equivalence* classes to our context (Section 2.2.1). Then, we present the design of a data structure capable to encode the occlusion information (Section 2.2.2). We then analyze how the occlusion of S can be extracted for each point on T (Section 2.2.3). Since in this section we are only providing a theoretical analysis, we do not explain how our algorithm builds the data structure. The practical details of our implementation are presented in Section 2.3.

2.2.1 Equivalence Classes of Oriented Lines

Our work relies on the concept of *equivalence* classes of lines, as described by Pellegrini, and detailed in Chapter 1, Section 1.2.2. *Equivalence* classes are continuous sets of lines which hit or miss the same subset of triangles. Therefore, in this context, they represent coherent paths through the scene, independently of any viewpoint. As a consequence, two lines belonging to the same equivalence class are spatially coherent.

This definition can be restricted to the arrangement of lines induced by S and all the geometry in $O(S,T)$. More exactly, since we want to describe the occlusion of S, through the geometry in $O(S,T)$, we only consider the oriented lines which stab S.

In Pellegrini's case, an occluder can be represented by all the oriented lines which intersect it. In our case, an occluder is represented by all the oriented lines which intersect S and then intersect the occluder. Figure 2.7 provides an illustration.

At this point we make the following assumption: For S and any occluder O, we suppose that neither polygon has an intersection with the support plane of the other polygon. This assumption is necessary in order to respect the hypothesis of Theorem 2, presented in Section 1.4.14, and which will be later used by our algorithm. This supposition is without loss of generality, since the degenerate cases can be reduced to a valid configuration, as explained in Section 1.4.14. Also, Section 2.3 provides further details on how our algorithm deals with these cases.

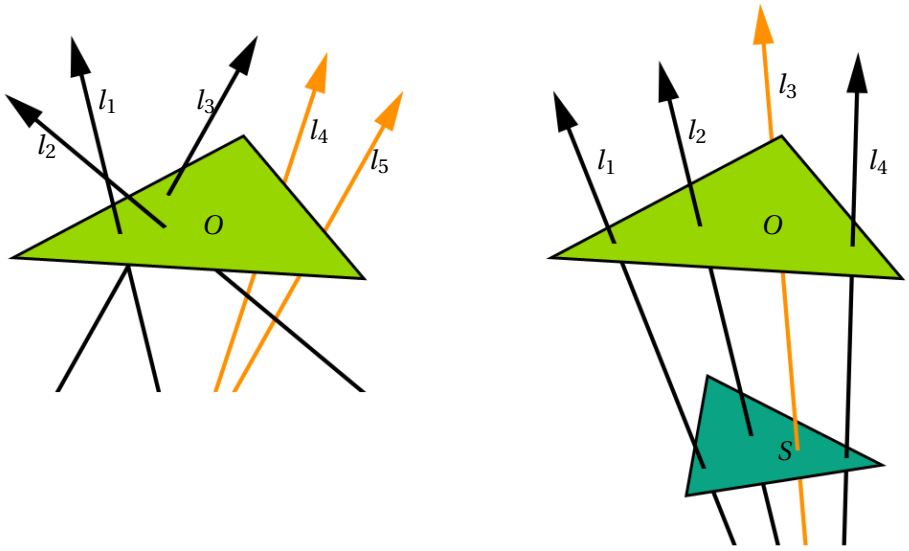


Figure 2.7: The *equivalence classes* of lines (Pellegrini) can be restrained to the arrangement of lines induced by S and its occluders. This illustration considers the case of a single occluder O. **Left**: In Pellegrini's case, we can represent O by all the oriented line stabbing it. The arrangement induced by O contains a cell corresponding to the lines stabbing O, (l_1, l_2, l_3), and the cells corresponding to the lines missing O, (l_4, l_5). **Right**: In our case, we represent O by all the oriented lines which stab S and then stab O. The arrangement induced by S and O only considers the line stabbing S, and contains a cell corresponding to the lines stabbing S and O ,(l_1, l_2, l_4), and the cells corresponding to the lines stabbing S and missing O ,(l_3).

In order to describe the occlusion from S, we focus on the coherent sets of lines which intersect S and either miss all the occluders in $O(S,T)$, or hit at least one occluder in $O(S,T)$. Thus, we want to calculate the arrangement of lines where each cell corresponds to either one of these two cases:

- A coherent set of lines stabbing S and missing all the occluders in $O(S,T)$. The Plücker points in this cell correspond to *unoccluded* lines.
- A coherent set of lines stabbing S and at least one of the occluders in $O(S,T)$. The Plücker points in this cell correspond to *occluded* lines.

Figure 2.8 gives a 2-dimensional example of a random scene and the lines which are considered in Pellegrini's case and in ours.

In our context, we only distinguish between *occluded* and *unoccluded* lines. Moreover, since we only want to know which are the fragments of S visible from T, we do not care which occluders block the same path between T and S. Thus, we are actually calculating a simplification of the equivalence classes described by Pellegrini. In order to avoid confusion, we use the term *occlusion* classes to denote the cells of our arrangement of lines. Thus, our methods considers two types of *occlusion* classes: *occluded* and *unoccluded*, according to the cells previously described.

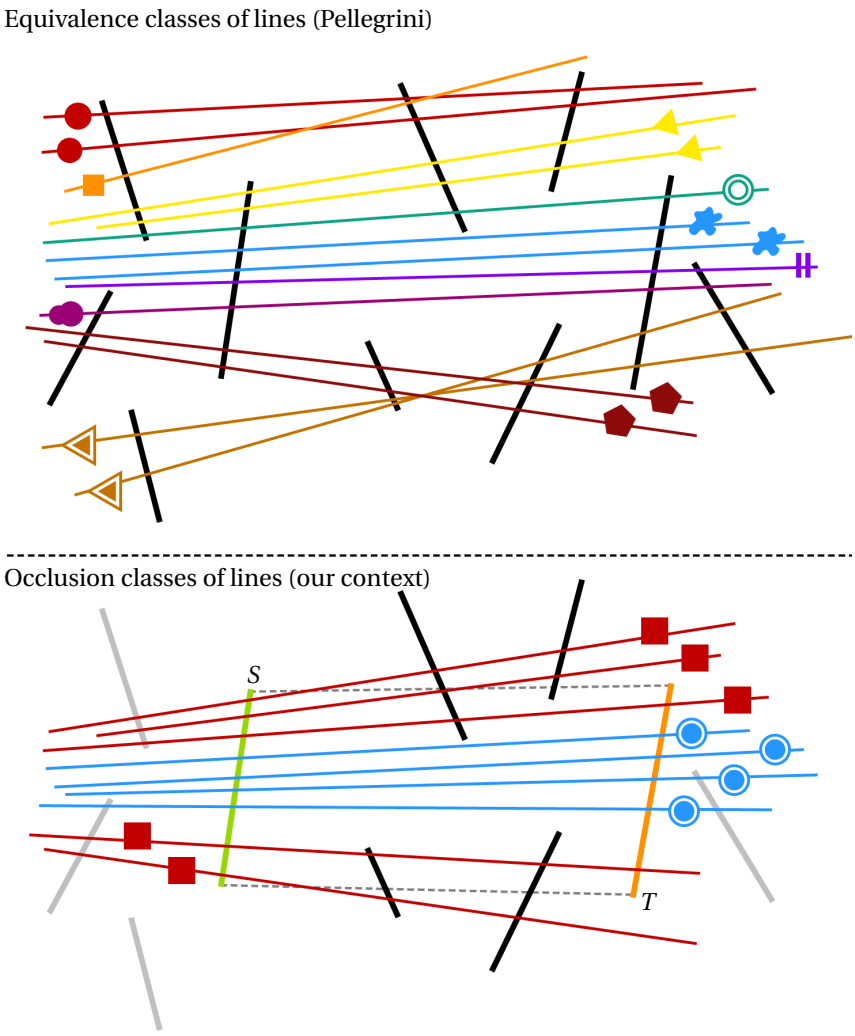


Figure 2.8: Comparison between the arrangement of lines from Pellegrini's theoretical framework (up) and the line partition considered in our case (down). The arrangement from the theoretical framework concerns all the lines in Plücker space, while we are limiting our computations to the arrangement induced by S and all the geometry in $O(S,T)$. Moreover, in the first case, the lines are distinguished according to the triangles they intersect, while in our case, we only consider two types of lines: occluded (marked with a rectangle) and unoccluded (marked with a double circle). Note that for the clarity of this figure, we only represent some classes of lines, and not all the possible configurations.

2.2.2 Encoding the Occlusion Information

From an algorithmic point of view, the occlusion classes can be stored using a binary space partitioning (BSP) tree, since they are defined by an arrangement of hyperplanes. The inner nodes contain the hyperplanes corresponding to the lines spanning the edges of the considered geometry. Each leaf represents a cell of the arrangement, and thus an *occlusion* class. This can either correspond to a coherent set of rays missing all the occluders (*unoccluded* class), or a coherent set of lines intersecting at least one occluder (*occluded* class). Figure 2.9 illustrates the case of the arrangement induced by a single occluder O.

Up to this point, we have defined an analytical representation of the occlusion from a surface S as seen through a set of occluders, as well as the structure suitable to encode it. The next section focuses on how our line partition can be used to extract the exact visible fragments of S, as seen from any point xyz on T.

2.2.3 Extracting the Occlusion Information

Given a point xyz on T, we want to find the visible parts of S from xyz. This involves all the lines originating from xyz and intersecting S. Thus, we have to find the coherent subsets of lines which are not blocked by any of the geometry in $O(S,T)$.

We assume that the data structure representing the arrangement induced by S and $O(S,T)$ and described in the previous section is constructed. This BSP tree already contains all the occlusion information of S, as seen from T. Thus, the information relative to xyz does not need to be calculated, but extracted. This is equivalent to finding the *occlusion* classes corresponding to the coherent set of lines which contain xyz.

Considering a line passing through xyz and stabbing S, we want to determine its occlusion class. This is can be achieved by locating the line's corresponding Plücker point into the data structure. Starting with the root node, the position of the point is tested against the hyperplanes contained in the inner nodes, until a leaf is reached. If the class associated with the leaf is an *occluded* class, then the line is blocked. Otherwise, we have determined an unoccluded line between S and xyz. Figure 2.10 illustrates this process using the example of data structure corresponding to a single occluder O, and which was previously detailed in Figure 2.9.

In order to compute the complete occlusion of S, as seen from xyz, the above operation is generalized to the infinite set of lines containing xyz and stabbing S. This can be seen as a view beam, having xyz as apex and S as base. As previously explained, we want to find all the coherent subsets of these lines, which are not blocked by any geometry. First of all, we explain how these sets can be represented using convex fragments of S. Then, we illustrate how the sets can be tested against the hyperplanes contained in the inner nodes.

Let us consider a line, l and its corresponding hyperplane π_l in Plücker space. We observe that xyz and l form a plane, $plane(xyz,l)$, which sets apart the lines intersecting xyz and S into two sets: The first one with a positive orientation with respect to l, the second one with a negative orientation. Notice that the orientation of the plane is coherent with the orientation of the line. Figure 2.11 gives an illustration. Testing the orientation of the set of lines stabbing S with respect to the hyperplane π_l is equivalent to testing the position of S with respect to the $plane(xyz,l)$. If S lies in the positive (or negative) half-space induced by the plane, the set of lines has a positive (or negative, respectively) orientation with respect to the hyperplane π_l. Otherwise, S can be subdivided by the plane into two convex fragments, leading to two coherent sets of lines, one with a positive orientation, the other with a negative orientation.

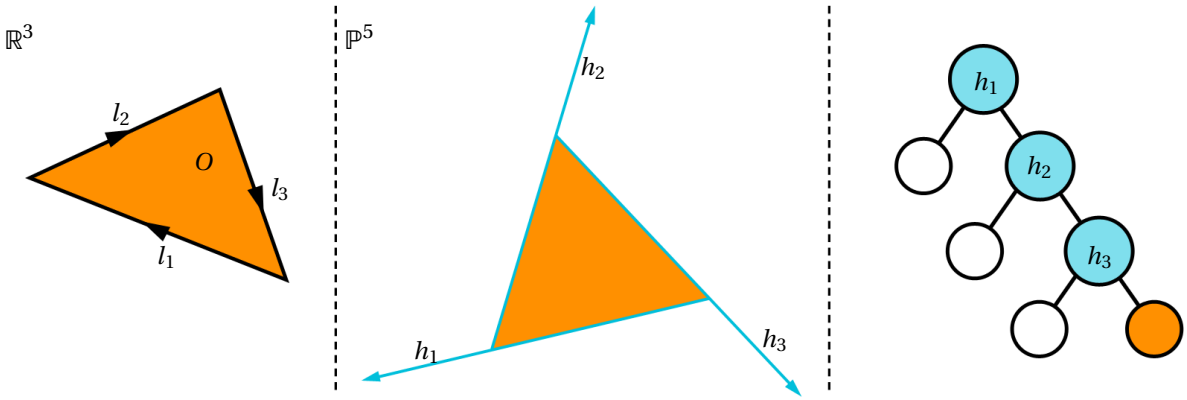


Figure 2.9: The BSP representation of an occluder O. **Left**: Let l_0, l_1 and l_2 be the oriented lines spanning the edges of O. **Center**: We note h_0, h_1 and h_2 their dual hyperplanes in Plücker space. **Right**: We can build a BSP tree whose leaves are the four *occlusion* classes generated by the triangle: Three *unoccluded* classes (left leaves) and one *occluded* class (right leaf). The inner nodes contain the three hyperplanes, h_0, h_1 and h_2. We note this representation $BSP(O)$.

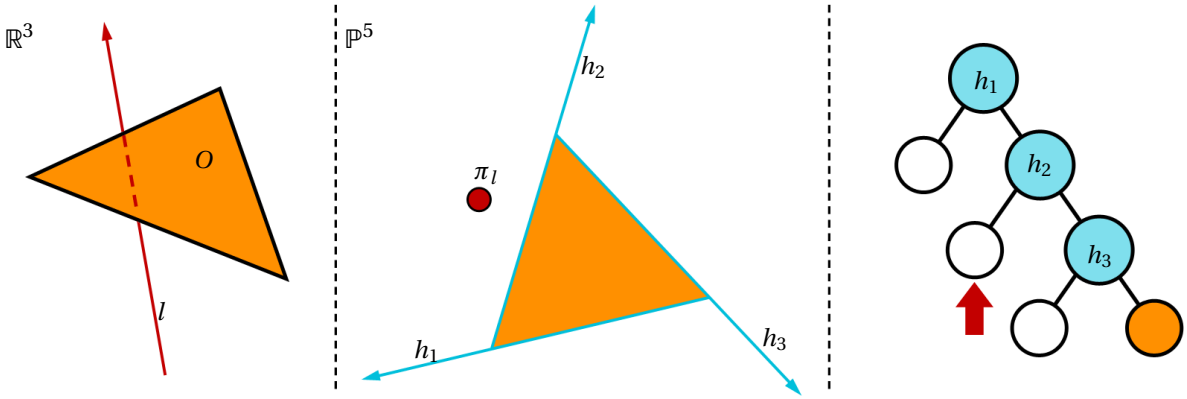


Figure 2.10: Locating a line into the $BSP(O)$. **Left**: The line l which stabs O. **Center**: Let h_0, h_1 and h_2 be the dual Plücker hyperplanes corresponding to the lines defined by O's edges. π_l is the Plücker point corresponding to the line l. **Right**: By testing the orientation of π_l with respect to the hyperplanes in the inner nodes, we can locate the line into the *occlusion* class it belongs to.

This is actually a particular case of Theorem 2, presented in Chapter 1, Section 1.4.14, when the first polygon is reduced to a single point.

The above procedure is repeated in each inner node reached by a subset of lines corresponding to a fragment of S. Therefore, all the initial lines (containing xyz and intersecting S) are located into the leaves of the BSP tree. If a fragment corresponding to a coherent set of lines reaches a leaf representing an *occluded* class, then the lines are blocked by some geometry in $O(S,T)$. Thus the fragment can be discarded. Otherwise, we have determined a coherent set of unoccluded lines and a corresponding visible fragment of S.

The result is a list of convex polygons (fragments of the initial S polygon), which correspond to

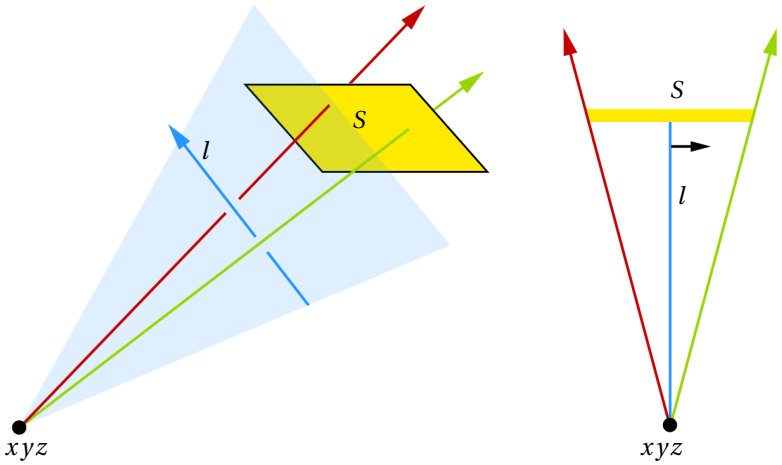


Figure 2.11: The point xyz and the line l define a plane which sets apart the lines stabbing S and having a positive or negative orientation with respect to l.

coherent sets of unoccluded lines. These are oriented lines which contain xyz and stab only S, and none of the occluders in $O(S,T)$. Although the fragments do not alter that data structure, they are used to represent the sets of stabbing lines, and to identify their *occlusion* classes.

Once all the visible fragments of S are known, the exact illumination of point xyz can be calculated.

The next section details the practical aspects of our method.

2.3 Implementation

The previous section has detailed from a theoretical point of view a data structure representing an arrangement of lines induced by a light source and its occluders, and how we can extract from this structure the occlusion information for a point. However, in practice, the two steps are not independent. The data structure is actually built on-demand depending on when and where the occlusion information is needed. This lazy construction is directed by the occlusion queries so that only the required classes are computed.

As indicated by the theoretical analysis provided by Pellegrini, the upper bound complexity of an arrangement of lines in Plücker space is $O(n^4 \log n)$. This represents an important limitation, which can drastically restrict the application of an algorithm. Therefore, special care must be given to the construction of the data structure, in order to keep the complexity low. Since we are interested in the soft shadows for a scene as seen from a given viewpoint, we only apply our calculations to the points which are visible from the viewpoint. The occlusion information is calculated only where and when it is needed, in order keep a low complexity of the data structure and gain in performance.

2.3.1 Overview

The oriented lines stabbing S are partitioned into coherent sets according to the *occlusion* classes they belong to. In Section 2.2.1 we have defined two types of *occlusion* classes: *occluded* and *unoccluded*, representing the lines which are blocked by the geometry in $O(S,T)$ and those which are not, respectively. However, since our implementation is lazy, the occlusion classes are developed as directed by the algorithm queries. Therefore, during execution, some leaves may not yet represent an *occluded* / *unoccluded* class. In order to handle this case correctly, we define a third type of *occlusion* class, the *undefined class*.

Our implementation works with the following classes of oriented lines:

- An *unoccluded* class: Any *occlusion* class representing a set of lines that do not intersect any occluder.
- An *occluded* class: Any *occlusion* class representing a set of lines that intersect at least one occluder.
- An *undefined* class: An *occlusion* class which has not yet been found as either *unoccluded* or *occluded*.

The algorithm builds a BSP tree in Plücker space, providing a hierarchical representation of the *occlusion* classes generated by the occluders. Each leaf represents one of these classes.

We first summarize the main operations performed by the method, and then provide a complete description of each step, together with a pseudo-code and an illustrative example.

- At first, all the potential occluders for the light S and the triangle T are selected.

- For an occluder O, the lines stabbing both S and O are contained in the 5-dimensional minimal polytope, $poly(S \rightarrow O)$. A partial representation of this polytope is then calculated for each occluder.
- A root node of a BSP tree in Plücker space is created and associated with all the occluders.
- The tree is grown by inserting the occluders. This comes down to replacing a *undefined* class with the sub-tree representing the *occluded*/*unoccluded* classes generated by an occluder.
- An *undefined* leaf is associated with one or several occluders. Therefore, after it has been replaced, its occluders are located into the newly added subtree. For an occluder O, this is achieved by successively testing the position of $poly(S \rightarrow O)$ against the hyperplanes contained in the inner nodes. In order to avoid performing 5D CSG operations, this process is conservative.

Selecting occluders

Occluders are defined as the geometry intersecting the convex hull of a triangle and a light source. In practice, the selection of the occluders relies on a shaft culling approach as described in [Hai00]. This involves the bounding box of a visible triangle and the bounding box of an area light source. The shaft is defined as the convex hull of the two bounding boxes plus the triangle's and the light's support planes. Any triangle intersecting the shaft is considered as a possible occluder. It is important to underline that our algorithm does not perform any intersection calculations between the geometry and the shaft. While this definition can lead to a conservative occluder set, it can be computed efficiently. Figure 2.12 provides an illustration of the selection process, including the various cases which can occur.

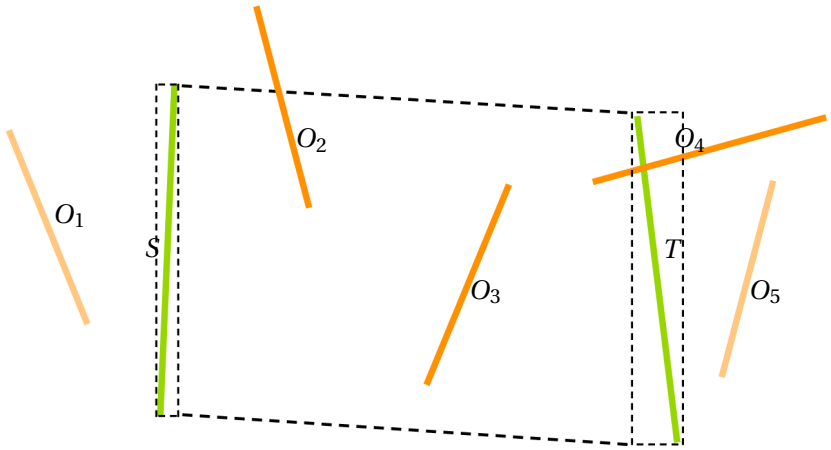


Figure 2.12: The occluders of S and T are defined as the geometry intersecting the convex hull of the two polygons. O_1 and O_5 are rejected because they are located outside of the convex hull. O_2, O_3 and O_4 intersect the convex hull, thus they are considered as potential occluders. Note that we are not performing any intersection calculations for the geometry which intersects S and/or T or the edges of the convex hull. This leads to a conservative set of occluders. Although it is best to have as few occluders as possible, calculating the exact blocking geometry would be too expensive than dealing with a slightly conservative set.

If the definition of the scene allows it, a back-face culling selection can also be applied to the occluders.

Creating a 5-dimensional minimal polytope for each occluder

Let O be an occluder from $O(S,T)$. As before, we assume that neither O, nor S, has an intersection with the support plane of the other polygon.

As previously explained in Chapter 1, Section 1.4.14, all the lines stabbing S and O are contained in a minimal polytope defined in Plücker space, noted $poly(S \rightarrow O)$. Algorithm 1 details the elements of this polytope (lines 1 – 5), and the procedure used to create it (lines 7 – 41). In our case, the V-Representation is obtained by considering all vertex-to-vertex combinations, using one vertex of S and one vertex of O (lines 12 – 18). The H-Representation would normally include the H-Representation of the light source, as well as that of the occluder. However, our algorithm only deals with the lines stabbing S, and all the queries are formulated with respect to the light source. Therefore, we can omit its H-Representation, and only consider the H-Representation corresponding to O (lines 20 – 24). This corresponds to the Plücker hyperplanes generated by the support lines of the occluder' edges.

In order to ensure an uniform orientation for the entire data structure, each polytope must be verified and oriented respecting the same convention. Without loss of generality, we assume that the interior of a polytope has a positive sign, while its exterior has a negative side (lines 26 – 40).

BSP representation of an occluder

Figure 2.9 has illustrated the BSP representation of an occluder. As previously explained in Section 2.2.1, the BSP representation of an occluder O, noted $BSP(O)$, is built using the Plücker hyperplanes corresponding to the support lines of O's edges, placed in the inner nodes. These are the hyperplanes considered in the partial H-Representation of the polytope. Also, the right leaf corresponds to the coherent set of lines stabbing O (*occluded* class, corresponding to the occluder's interior), while the left leaves correspond to coherent sets of lines missing O (*unoccluded* classes, corresponding to the occluder's exterior).

Creating the root node

The root node corresponds to the oriented lines stabbing S. These lines will be partitioned according to whether or not they are blocked by the occluders in $O(S,T)$.

For each one of these occluders, a Plücker polytope is created, using the procedure described in Algorithm 1. The structure of a node is presented in Algorithm 2, and the initialization of the root node is summarized in Algorithm 3. The main idea is to build one polytope for each occluder O in $O(S,T)$ and the light S. All the occluders are then associated with the root node.

Algorithm 1 Creation of the 5-dimensional minimal polytope corresponding to the lines stabbing two convex polygons, P_S and P_T, $poly(P_S \rightarrow P_T)$

```
1:  Structure Polytope
2:  {
3:      VRep : list of Plücker points
4:      HRep : list of Plücker hyperplanes
5:  }
6:  ____________________________________________________
7:  Function create5DOccluder (Polygon P_S, Polygon P_T)
8:  return Polytope
9:
10: Polytope O
11:
12: //Compute the V-Representation
13: for each vertex v_S of P_S do
14:     for each vertex v_T of P_T do
15:         Plücker v ← createPlücker (v_S, v_T)
16:         O.VRep.add (v)
17:     end for
18: end for
19:
20: //Compute the partial H-Representation
21: for each two consecutive vertices v_T^1 and v_T^2 of P_T do
22:     Plücker h ← createPlücker (v_T^1, v_T^2)
23:     O.HRep.add (h)
24: end for
25:
26: //Check the orientation
27: Point BP_S ← barrycenter of P_S
28: Point BP_T ← barrycenter of P_T
29: Plücker l ← createPlücker(BP_S, BP_T)
30: Int check ← 0
31: for each hyperplane h of O.HRep do
32:     if side(h, l) < 0 then
33:         check ← 1
34:     end if
35: end for
36: if check = 1 then
37:     for each hyperplane h of O.HRep do
38:         reverseOrientation (h)
39:     end for
40: end if
41: return O
```

As previously described, degenerate cases may appear if S is intersected by the support plane of the occluder. As explained in Chapter 1, Section 1.4.14, in this case it is not possible to group together the lines stabbing both polygons in a single polytope. Thus, S needs to be split with respect to the support plane of the occluder, and two polytopes are created. This is illustrated in Figure 2.13 and detailed in Algorithm 3, lines 14 – 19.

Another configuration that needs to be avoided occurs when an occluder O intersects T. In

this case, the fragment of O located below T, with respect to S, would also be counted as an occluding geometry, thus resulting in an over occlusion of T. In this case, O can and must be clipped with respect to the support plane of T. This is illustrated in Figure 2.14 and further explained by Algorithm 3, lines 8 – 9.

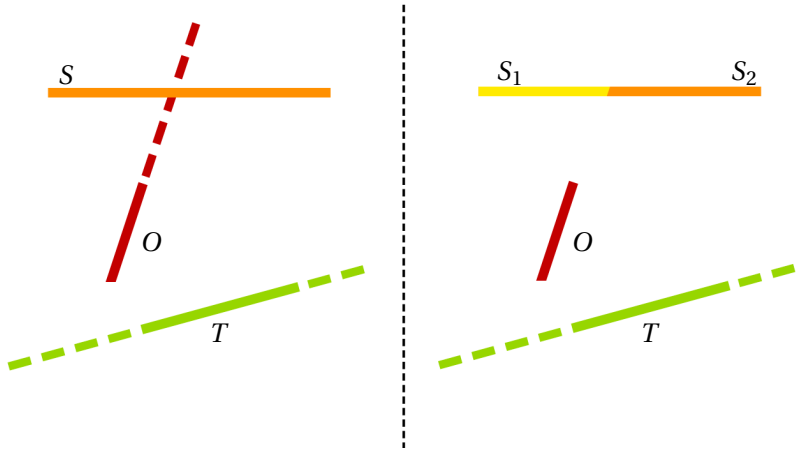


Figure 2.13: Dealing with the degenerate case when an occluder intersects the light source. In this case, the lines stabbing both O and S cannot have coherent orientations. Thus, the source needs to be split with respect to the support plane of the occluder, and two polytopes are created: $poly(S_1 \rightarrow O)$ and $poly(S_2 \rightarrow O)$. This operation is also described in Section 1.4.14 and Figure 1.18.

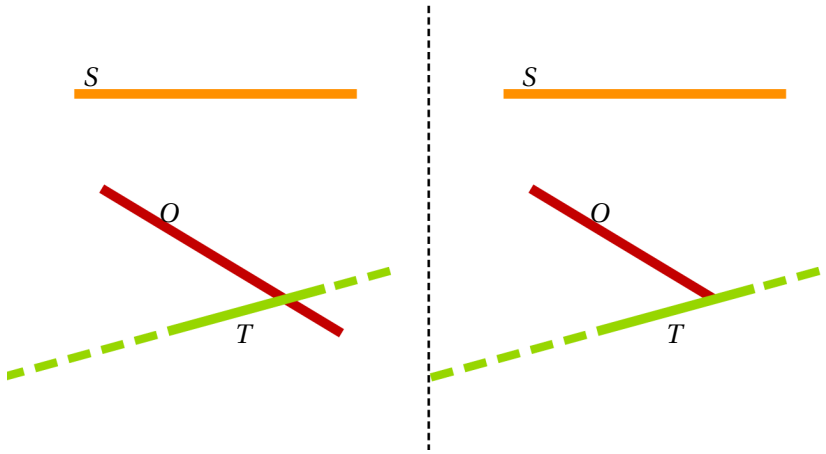


Figure 2.14: Dealing with over occlusion cases, when an occluder intersects the polygon T. In this case, the occluder is clipped with respect to the support plane of T. This is possible because we are only interested in the visibility from the points located on T.

We could also mention here the case when an occluder intersects the support plane of S. However, this is a theoretical case only. In practice, the light is usually located above the scene and does not intersect the geometry to shade. Although we are not dealing with this case in our current implementation, it can be handled easily, by clipping the occluder with respect to the support plane of S. This is similar to the process described for T.

Algorithm 2 The structure of a node

```
1: Structure Node
2: {
3:     hyperplane : Plücker hyperplane
4:     class : {occluded, unoccluded, undefined}
5:     occluders : vector of Polytope
6:     leftChild : Node
7:     rightChild : Node
8: }
```

Algorithm 3 Creation of the root node, with creation of the initial list of occluders

```
1: Function createRoot (Polygon S, Polygon T, list of Polygon O_list)
2: return Node
3:
4: Node n
5:
6: for each Polygon O of O_list do
7:
8:     if O ∩ supportPlane(T) ≠ ∅ then
9:         O ← clip(O, supportPlane(T))
10:    end if
11:
12:    // Test if S intersects the support plane of the occluder
13:
14:    if S ∩ supportPlane(O) ≠ ∅ then
15:        Split S into S_1, S_2 with respect to supportPlane(O)
16:        Polytope VO_1 ← create5DOccluder (S_1, 0)
17:        Polytope VO_2 ← create5DOccluder (S_2, 0)
18:        n.occluders.add(VO_1)
19:        n.occluders.add(VO_2)
20:    else
21:        Polytope VO ← create5DOccluder (S, O)
22:        n.occluders.add(VO)
23:    end if
24: end for
25:
26: n.class ← undefined
27:
28: return n
```

Growing the data structure: Inserting an occluder

The tree grows each time an occluder is inserted: The *occluded*/*unoccluded* classes generated by the occluder are added to the tree. Inserting an occluder O into the tree consists in replacing a leaf by the root of $BSP(O)$. A leaf corresponding to an *undefined* class represents a set of lines which may be partially or totally blocked by the occluders associated with the leaf. By replacing this leaf with the sub-tree corresponding to one of its occluders we are further subdividing the set of lines with respect to this blocking geometry.

Locating an occluder

Locating an occluder into a subtree relies on the Theorem 2, presented in Chapter 1, Section 1.4.14. The procedure is illustrated by Figure 2.15 and detailed in Algorithm 4.

Algorithm 4 Location of an occluder

```
1:  Procedure locateOccluder (Node n, Polytope O)
2:
3:  if n is a leaf then
4:      if n.class ≠ occluded then
5:          n.occluders ← n.occluders ∪ O
6:      end if
7:  else
8:      pos ← orientation(O.VRep, n.hyperplane)
9:      if pos > 0 then
10:         locateOccluder(n.rightChild, O)
11:     else if pos < 0 then
12:         locateOccluder(n.leftChild, O)
13:     else
14:         locateOccluder(n.rightChild, O)
15:         locateOccluder(n.leftChild, O)
16:     end if
17: end if
```

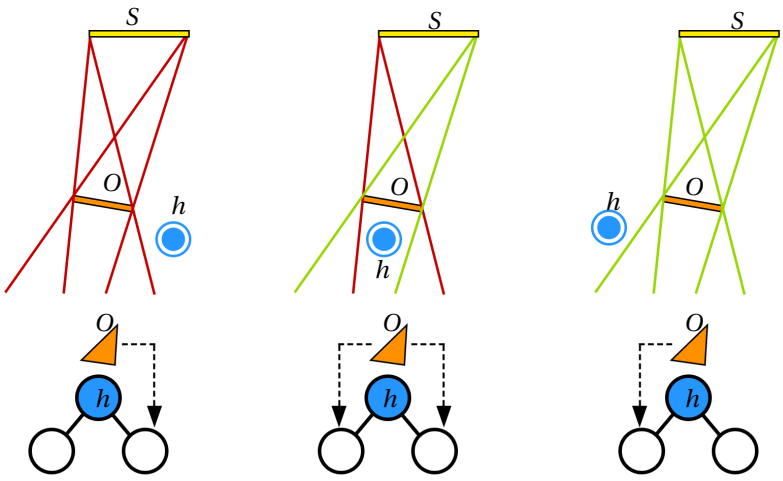


Figure 2.15: 2D illustration of how an occluder is located with respect to an inner node using Theorem 2. An occluder O intersects a subset of all the lines originating from S. This subset may have a negative (left example), a positive (right example) or a mixed (center example) orientation with respect to any hyperplane h from an inner node. Theorem 2 allows to define this orientation using the V-Representation lines, *i.e.* the lines defined by one vertex of O and one vertex of S. As a consequence the occluder O is located into the right or left or both subtrees of the node.

Occluders are inserted into the tree and located into the leaves (and thus the classes) they may affect. Inserting an occluder O comes down to testing the relative orientation of a hyperplane from an inner node and the lines occluded by O. The occlusion created by an occluder O is the set of lines

intersecting both S and O, thus the lines contained in the minimal polytope $poly(S \rightarrow O)$. Therefore, we use Theorem 2 to determine the orientation of the occluded lines with respect to the hyperplane. If the current node n is a leaf which is not an *invisible* class, it is affected by the occluder O and thus it is stored in the leaf (lines 3 – 7). Otherwise, if n is an inner node, we have to test the relative orientation of the lines blocked by O with respect to $n.hyperplane$, the hyperplane contained in the node n. This is done by applying Theorem 2 using the V-Representation of $poly(S \rightarrow O)$ (line 8, lines 9 – 17).

The three cases which may occur are detailed below and illustrated by Figure 2.15:

- If the orientation is positive, O is inserted in the right child of n (line 10).
- If the orientation is negative, O is inserted in the left child of n (line 12).
- Otherwise, O is inserted in both children (lines 13 – 17).

2.3.2 Core Algorithm

The core of our algorithm finds the *occlusion* classes related to the visible fragments of S, as seen from a point xyz, through a set of occluders $O(S, T)$. Algorithm 5 details such a query and Figures 2.16, 2.17 and 2.18 apply it to a given configuration.

Algorithm 5 starts with all the occluders associated to a single *undefined* leaf (the root node). For each inner node, we compute the plane defined by xyz and the line stored in the node (line 9), as illustrated by Figure 2.11 and explained in Section 2.2.3. Then, S is tested against this plane to determine the orientation of the lines stabbing S and xyz (line 10). If S lies in the positive (resp. negative) half-space of the plane, then all the lines have a positive (resp. negative) orientation, and the algorithm continues in the left (resp. right) subtree (line 11 or 13). Otherwise, S is split against the plane and the algorithm continues recursively in both subtrees with the relevant parts of S (lines 15 – 16).

When a fragment reaches a leaf, two alternatives can occur:

- The leaf has no occluders, therefore it is either an *unoccluded* or an *occluded* class. In the first case, the fragment is a convex part of S which is visible from xyz, thus it is returned (line 24). Otherwise, the fragment is invisible from xyz and it is discarded.
- The leaf has some occluders, thus the class is *undefined* and we cannot answer the query without further developing the tree.

In the latter case, the algorithm chooses a random occluder (line 26) RO among the occluders associated with the current leaf. This occluder is used to grow the tree by replacing the leaf by $BSP(RO)$, the BSP representation of the four *occlusion* classes generated by RO (line 27). Next, the remaining occluders are located into $BSP(RO)$ which is now part of the tree. This is achieved by the procedure at line 29 and described in Figure 2.15. When all the occluders have been located in

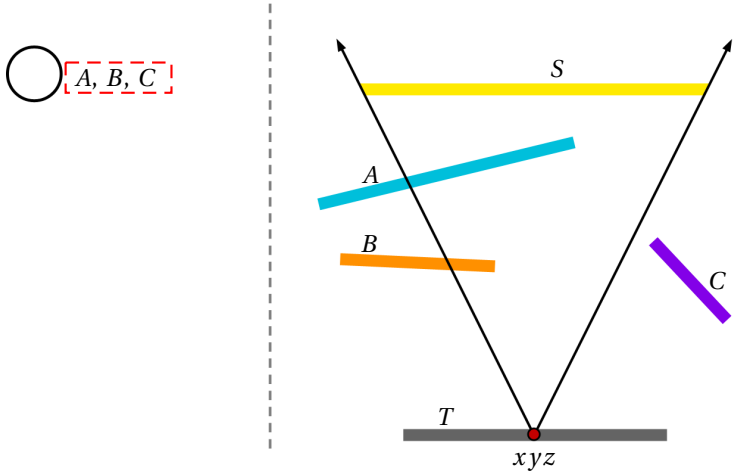


Figure 2.16: Illustration of Algorithm 5 for developing the BSP tree for the polygon S and the set of occluders $\{A,B,C\}$. This is achieved by calculating the visible fragments of S as seen from the point xyz on T. **Left:** The algorithm starts with a root node, associated with the entire set of occluders. This represents an *undefined* leaf. **Right:** The initial set of lines is defined by xyz and S.

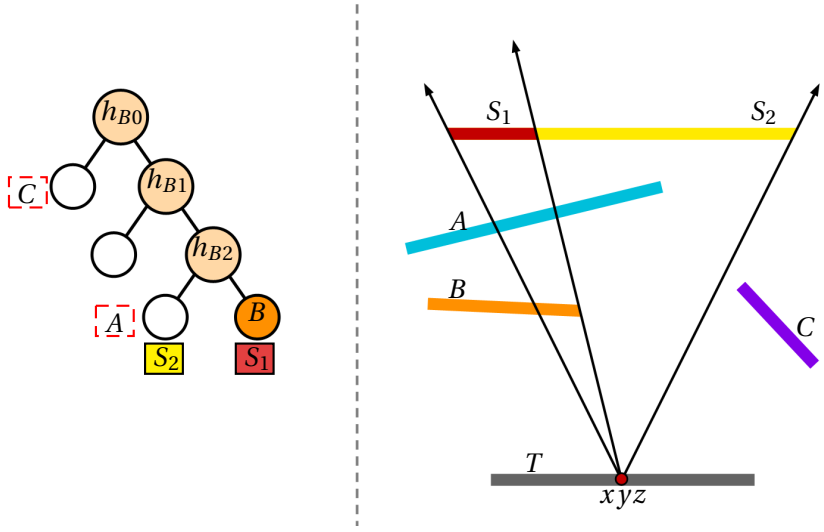


Figure 2.17: **Left**: One occluder is chosen randomly (B in this case), and the root is replaced by $BSP(B)$. The inner nodes contain the hyperplanes corresponding to B's edges (h_{B0}, h_{B1}, h_{B2}). The left leaves represent the lines missing B, while the right leaf represents the lines blocked by B. The remaining geometry is located into the tree, using the technique described by the procedure *locateOccluder*. Since $poly(S \rightarrow A)$ is found to intersect π_{B2}, the occluder is sent to both left and right leaves. However, since the right node represents a set of blocked lines, the occluder is discarded. **Right**: Next, the conntinous set of lines stabbing S are located into the tree. An intersection occurs in the last inner node ($S \rightarrow S_1, S_2$), identifying thus the lines blocked by B (defined by S_1) and the lines missing B (defined by S_2). The fragment S_1 will be discarded, while S_2 lands in the last left leaf. This leaf has an associated occluder (A), and thus represents an *undefined* class. This means that the lines missing B may stab A. In order to answer the query, the tree must be further developed.

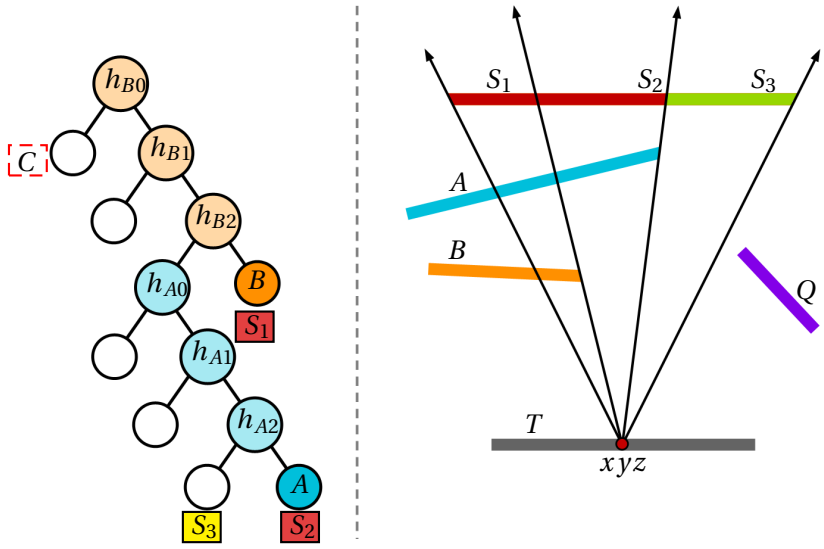


Figure 2.18: **Left:** $BSP(A)$ replaces the leaf. The newly added left leaves correspond to lines missing both A and B, while the right leaf represents a set of lines blocked by A. **Right:** P_2 is analytically split into P_2 and P_3, which are located into the tree with respect to the geometry they intersect.

$BSP(RO)$, the algorithm continues from the root of $BSP(RO)$ until it finds the *occlusion* class for the current light fragment.

Algorithm 5 Occlusion query (core of the algorithm): Given a point xyz on T, the algorithm answers the query: "which parts of S are visible from xyz ?" S is used to drive the BSP tree construction and may be split into several fragments representing coherent sets of lines from xyz. The fragments reaching *unoccluded* classes correspond to visible parts of S from xyz."

```
1:  Function mainQuery (Node n, Polygon S, Point xyz)
2:  return Polygons
3:
4:  loop
5:     ____________________________________________
6:     // Querying the data structure
7:
8:     while n is not a leaf do
9:        Plane p ← makePlane(xyz, n.hyperplane)
10:       if position(p, S) > 0 then
11:          n ← n.rightChild
12:       else if position(p, S) < 0 then
13:          n ← n.leftChild
14:       else
15:          // Split the light (S) and work recursively
16:          return mainQuery(n.rightChild, S ∩ p+, xyz) ∪ mainQuery(n.leftChild, S ∩ p−, xyz)
17:       end if
18:    end while
19:
20:    ____________________________________________
21:    // Building the data structure, if required to answer a query
22:
23:    if n.occluders is empty then
24:       return (n.class is unoccluded) ? S : ∅
25:    else
26:       RO ← random occluder from n.occluders
27:       n ← root of BSP(RO)
28:       for each O in n.occluders, O ≠ RO do
29:          locateOccluder (n, O)
30:       end for
31:    end if
32: end loop
```

2.3.3 Key Points

This section underlines significant points of the algorithm.

Random selection of occluders:

The efficiency of the algorithm is related to the balance of the tree. To develop the data structure, the algorithm chooses an occluder randomly. Obviously, some choices may lead to a more balanced tree than others. However this is not predictable. In fact, we have tested different heuristics, trying to make a "good choice". First of all, we have developed a selection procedure which chose the largest occluders first. Secondly, we considered the beam formed by xyz and a fragment of S and selected an

occluder amongst those which either intersect this beam, or are located within a certain minimum distance from it. Unfortunately, all of these heuristics achieved poor improvements compared to the extra computational cost. Moreover, their behavior can be very different according to the nature of the scene. We have reached the conclusion that a random choice gives better results, and more important, it has a consistent behavior independently of the rendered scene. This is similar to the choice of the pivot in the well known quicksort algorithm: Although a random pivot is not the optimal choice, it leads to the best performance in practice.

Conservative insertion:

If all the lines stabbing an occluder O do not have the same orientation with respect to a hyperplane, O is located in both subtrees of the relevant node. As a consequence this process is conservative. This step could be computed exactly as in [Bit02, NBG02, HMN05, MA05]. But, as explained in Chapter 1, this is expensive and prone to numerical errors. Our algorithm avoids these problems to remain simple and robust.

Tree growth:

It is important to note that not all the queries will develop the data structure. Some queries develop new *occlusion* classes, at least at the beginning since the BSP tree is empty. And more important, the majority of the queries are expected to take advantage of the previous computations thanks to the visibility coherence. This is a key point of the algorithm's efficiency.

2.4 Soft Shadows Framework

To illustrate the efficiency and the reliability of our exact occlusion algorithm, we plug it into a ray-tracing rendering software for computing high quality soft shadows. Algorithm 6 describes the process.

Algorithm 6 The following pseudocode illustrates how our occlusion algorithm is plugged in a ray tracer software to analytically compute soft shadows

```
1:  build visible_triangle, the triangle list visible from the camera
2:  for each light S in the scene do
3:     triangle_list ← visible_triangle
4:     // the following loop parallelization is straightforward
5:     while triangle_list is not empty do
6:        remove a triangle T from the triangle_list
7:        select the occluders O(S,T) of S and T using shaft culling
8:        initialize a BSP tree root node n associated with O(S,T)
9:        for each image point xyz on T do
10:          visible_parts ← mainQuery( n, S, xyz)
11:          compute the illumination in xyz using visible_parts
12:       end for
13:    end while
14: end for
```

- Using the primary rays, all the image points are grouped together with respect to the triangle they belong to. This builds a list of visible triangles (line 1).
- Multiple lights are handled successively and independently (line 2)
- For each visible triangle, an empty data structure is created and associated with its set of occluders (line 8).
- Next, for each image point, Algorithm 5 is used to compute the visible parts of the light (line 10). In our framework, we consider area light sources with a uniform emission, therefore we compute direct illumination analytically (line 11) by integrating over the visible parts of the light [NN85].

This framework is designed to allow an efficient implementation. The loop order is chosen to build one data structure per light and per visible triangle. Since visible triangles are successively handled, BSP trees are developed successively and independently. This improves the memory coherence and avoids switching many times between the data structures. This also limits the memory consumption since each BSP tree is deleted as soon as all its related image points are shaded. Moreover, the implementation can be easily multi-threaded: A thread gets a visible triangle from the list, shades its image points and starts over until the list becomes empty. In this case, the triangle list access (line 6) has to be protected.

It is important to note that our method is designed as a black box, which can be integrated

with a rendering application. The input data consists of the points to be shaded, grouped by their source surfaces, as well as the geometry of the scene. The algorithm returns, for each point, its final shading value.

Implementation details:

At last, compared to Algorithm 5, a stack is managed to avoid recursive system calls. Except for this detail, the implementation is straightforward, it uses single floating point precision and does not use SIMD instructions.

The main operations performed by the algorithm are the position test which allow locating an occluder into the leaves it may affect, and the tests which detect coherent sets of lines and direct then to the corresponding class. Both these operations are in fact point-hyperplane and point-plane tests, respectively, from which we only retain the sign of the result. This enforces the robustness of the algorithm. The most complex operation is the clipping of a polygon with respect to a plane, which usually does not raise numerical problems.

2.5 Results

All tests were run on a 2.67 GHz Intel Core i7 920 processor with 6GB of memory. For comparison purpose, all pictures were rendered at 1280×720 pixels with one primary ray per pixel. Four sets of results are presented, three sets testing the global performance of our soft shadow framework, and a last one giving an insight on the behavior of Algorithm 5.

2.5.1 Comparisons on time and quality

The first set of results compares our method to ray-traced soft shadows both at comparable time and comparable quality. The ray tracer implementation is similar to [WBWS01]: It relies on an optimized SAH kd-tree, uses SIMD instructions to trace four rays simultaneously, and supports multi-threading to render several parts of the picture in parallel. Ray-traced soft shadows are computed using groups of 4 shadow rays, and an uncorrelated stratified sampling of the area light sources. Since both our method and the ray-tracer support multi-threading, all computations are run using 4 threads. This type of implementation is particularly efficient when applied to coherent sets of rays, such as primary or shadow rays.

We use four scenes to test our method in different configurations. Despite its moderate geometrical complexity (26 673 triangles), the *T-Rex* scene is challenging for our approach because it presents difficult and complex shadows due to a long rectangular light source. This means that the light source visibility is complex and this is precisely what is computed by our algorithm. The modified *Sponza Atrium* with the statue of *Neptune* (115 737 triangles) and *Conference* (282 873 triangles) are significant and detailed models with different kinds of shadow complexities. At last, *Soda Hall* (2 147K triangles) is used to test the scalability of our approach on a massive model with heterogeneous geometry. The ambient light was intentionally increased in order to render visible the entire geometry of the scene.

Figure 2.19 details these results, and presents the time spent computing the soft shadows for each method. At comparable time, ray traced soft shadows are always noisy. At comparable quality (*i.e.* the noise is not noticeable anymore in the stochastic shadows), our method is always faster. In addition, this is very noticeable on a complex case such as the *T-Rex* scene, because the stochastic approach requires a very high number of shadow rays to remove almost all the visible noise. In contrast, the exact occlusion algorithm produces high quality results in a few seconds.

Table 2.1 presents the memory and time consumption for our algorithm. Since we compute one BSP tree in Plücker space for each visible triangle from the camera, the memory footprint varies during the process according to the building of the data structure. As a consequence, our results report the maximum memory load reached by our soft shadow framework. In any case, the memory footprint is low. The lazy evaluation is driven by the occlusion queries. Thus, the algorithm focuses on the classes related to the shadows, and avoids the computation of useless data. In addition, our framework is designed to avoid building too many BSP trees simultaneously (at most one per thread)

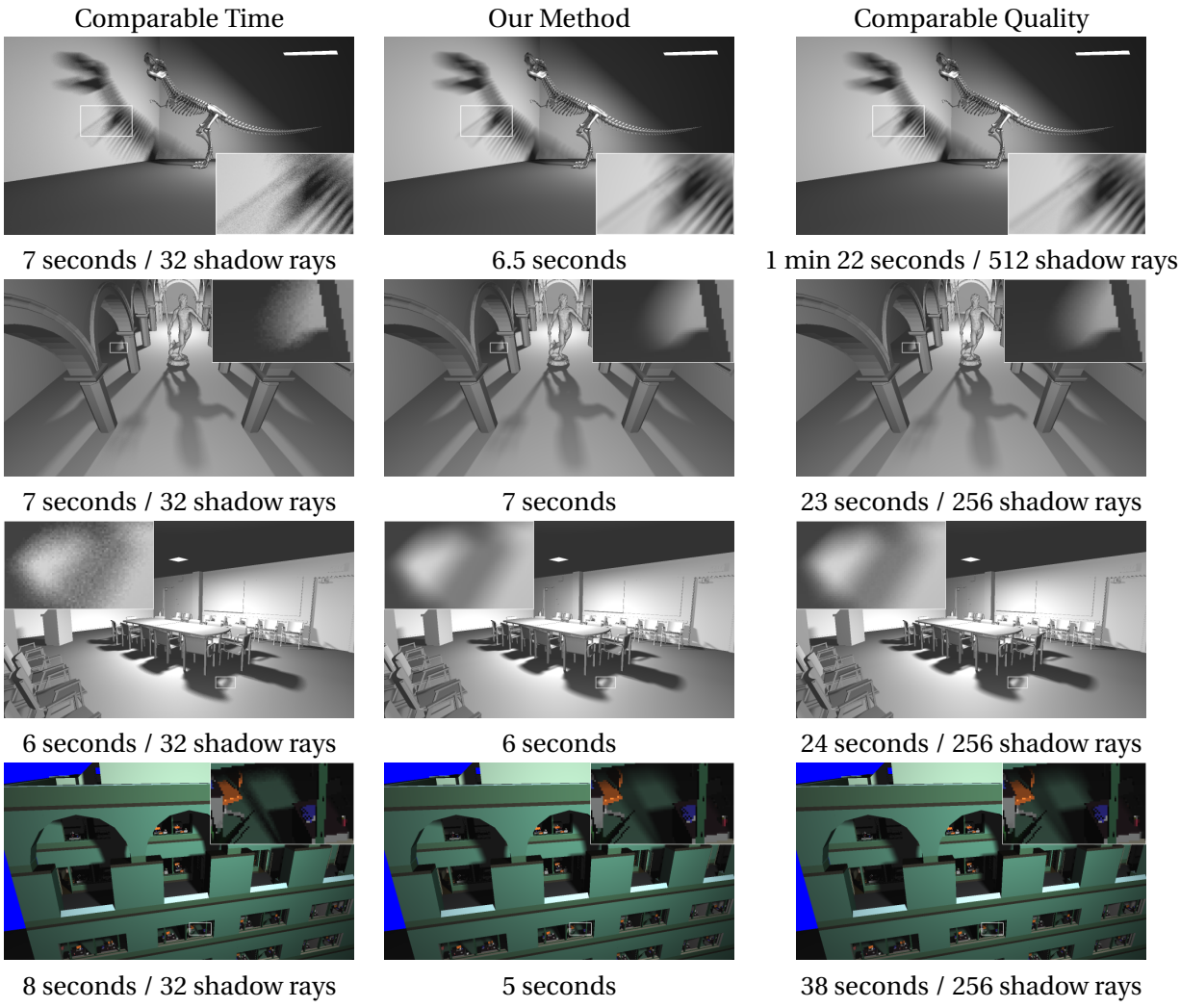


Figure 2.19: From top to bottom: T-Rex, Sponza with Neptune, Conference and Soda Hall. The middle column shows the results computed with our algorithm. The left column presents the same pictures computed at comparable time and the right one at comparable quality, both using shadow rays. As an indication of the performance of our comparison method, we have rendered the same pictures under the same circumstances (same computer, 4 threads), using Mental Ray© and obtained the following results: Trex (512 samples - 4"43) Sponza-Neptune (256 samples - 3"52), Conference (256 samples - 2"20), Soda (256 samples - 3"04). These timings concern only the shadow rays.

thus keeping the memory consumption low.

The total time can be subdivided in three steps: The occluder selection, the BSP tree initialization (including the computation of the H-Representation and V-Representation for each polytope), and the occlusion queries used for shading the image points. We can notice that the computation time is clearly dominated by the occlusion queries (*i.e.* calls to the *mainQuery* function (Algorithm 5)) which is the core of our method.

Aside the comparison with the stochastic approach, these results show that we can make the most of from-polygon occlusion coherence to design an efficient and robust algorithm, in contrast to previous works on this topic. As an example, we were unable to process the scenes presented in

	Memory	**Time**				**Modified Version**	
Scene	Max Size (KB)	Shaft Culling (%)	Init (%) (%)	Queries (%)	Total (s)	MV Time (s)	Acceleration Factor
T-Rex	19 916	4.8	1.9	93.3	6.5	314	×48
Sponza	16 246	4.0	2.0	94.0	7	191	×27
Conference	20 758	4.0	4.5	91.5	6	146	×24
Soda Hall	20 991	1.6	3.5	94.9	5	83	×16

Table 2.1: The time and memory consumption using our algorithm. The *Max Size* column is the maximum memory load reached during the process. The *Shaft Culling* column gives the time percentage spent to select the occluders. The *Init* column gives the time percentage spent to initialize each BSP tree. The *Queries* column gives the time percentage spent to query the BSP trees for shading each image point. The *Total* column gives the time in seconds for the whole process. *MV Time* column presents the result obtained using a Modified Version of our framework, where all queries are prevented from taking advantage of the others. The last column gives the acceleration factor between our framework (*Time*) and its modified version (*MV Time*).

this work with a method such as the one described in [MAM05]. The scenes are too complex for such an approach, which relies on 5D CSG operations. This becomes numerically unstable and leads to degenerate results.

As expected, the sensitivity of our algorithm to the visual complexity of the light source is confirmed. For instance, the *T-Rex* scene required roughly the same amount of time as the *Sponza* and *Neptune* scene or the *Conference* scene.

About the visual coherence:
A crucial property of our method is its ability to take advantage of the visual coherence between image points. To test this ability, the same pictures were also rendered using a modified version of our algorithm: Between each query (*i.e.* between each *mainQuery* call), the related BSP tree is reset to its root node associated with its former occluder set (this additional operation is excluded from the timings). In such a case, each query is "the first one" and we prevent all queries from taking advantage of the previous ones. The last two columns of Table 2.1 present the computation times achieved by our modified version, as well as a comparison between the two algorithms. The loss of efficiency is considerable. This demonstrates the capacity of the algorithm to benefit from the visual coherence, which is a key point to its efficiency.

2.5.2 Increasing the Area of the Light

The complexity of the soft shadows also depends on the size of the area light source. This is intrinsic to the soft shadows problem and it will inevitably affect our algorithm since it relies on the visibility coherence of the light source. If its area increases, the visibility coherence may decrease and lead to a loss of efficiency. As a consequence, the second part of our tests investigates the behavior of our framework when the area light source is increased. These tests were run on the *Conference* model. Our choice was motivated by the fact that this model has a significant number of triangles, as well as the wide range of shadows. The tests start using a small area light source whose size is progressively increased until it becomes 100 times larger. For each light size, the time and maximum memory

used by our method are measured. Figure 2.20 sums up the results. As expected, it shows a loss of efficiency, both in time and memory, as the light source size grows (left and right graph of Figure 2.20). By extent, there is inevitably a critical light size where the time and the memory consumption would become a problem, in particular the memory since it is a limited resource. However, our tests show that even with the largest area light source we are far from such a point. In addition, further comparisons using our ray-tracer show that our approach remains fast. At comparable quality, 1024 samples are required for the largest light and the computation takes 91 seconds against 39 seconds using our algorithm (2.33 times slower). Independently of the increase in the noise, the break-even occurs for a light source 184 times larger than the smallest one (92 versus 93 seconds using 1024 samples).

Figure 2.20: Increasing the light size. **Left**: The time consumption. **Right**: The maximum memory load reached during the computations. **Center**: A picture from our tests with the largest area light source (100 times larger than the smallest one. This corresponds to a square whose side is exactly the width of the conference table).

Figure 2.21: Increasing the number of lights. **Left**: The time consumption. **Right**: The maximum memory load reached during the computations. **Center**: A picture from our tests with 36 area light sources.

2.5.3 Increasing the Number of Lights

Generally, a scene has several light sources. As detailed by Algorithm 6, our implementation supports multiple lights which are handled successively. Thus, it is interesting to test the behavior of our framework in such a case. The *Conference* model is used again and rendered with 2 to 36 area light sources. All the lights have the same size and cast roughly the same "amount" of shadow. Figure 2.21 presents the results. The left graph shows that the time consumption is linear with respect to the number of area light sources. This is the expected behavior since our framework evaluates the contribution of each light source one after another. In addition, this makes the memory load independent from the number of lights, as shown on the right graph. However, we can notice a significant growth of the memory load when the number of lights is more than 12. Indeed, the 12 first

lights are above the conference table while the 24 other lights are mainly above the chairs, casting more complex shadows. The memory growth is independent from the number of lights in the scene, but it is coherent with the increase in the visibility complexity.

By increasing the size of the light source we have shown that an over-sized area light source can become a limitation for our method, mainly because of the memory consumption. By increasing the number of lights, we have found a possible solution to this problem. Any huge light source, even with a critical size, could always be treated as an union of several smaller lights. This does not affect the quality of the results, which remain noise free. This is especially interesting, since a stochastic method would require a high number of shadow rays for handling a huge light source.

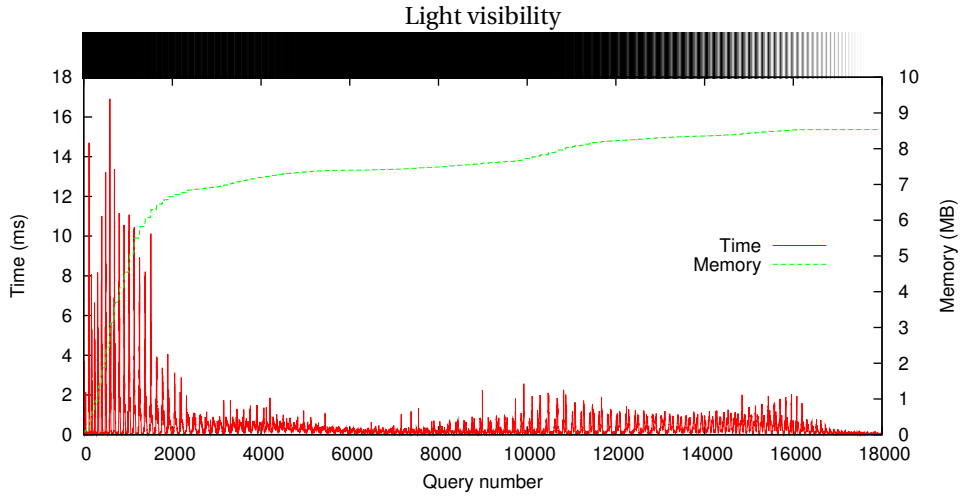


Figure 2.22: Lazy construction of a representative BSP tree. The abscissa corresponds to the number of queries. The left ordinate is the time in milliseconds and is related to the continuous curve, while the right one is the memory consumption in MB and is related to the dashed curve. Above the graphic, a half-tone illustration of the light visibility for each query. Black means invisible while white is fully visible.

2.5.4 Focus on the $mainQuery$ Algorithm's Behavior

The previous results demonstrated the global efficiency and robustness of our method, but they do not highlight the behavior of the algorithm's core (Algorithm 5), *i.e.* how the occlusion data and the computational cost evolve with respect to the occlusion queries. Figure 2.22 provides such an insight. It focuses on the construction of a single BSP tree for the *Conference* model. This tree was selected because it is representative of the *mainQuery* behavior. It has 2 896 occluders and a significant number of queries (17 878) are performed. In particular, most of the image points are located in the umbra or penumbra, which represent the most complex cases for any algorithm.

At first, the tree grows quickly because there is no occlusion data and the algorithm has to develop it in order to answer the occlusion queries. The timings show the extra computational cost required for this construction. As a second step, the tree growth slows down drastically because the previously computed occlusion classes can be re-used and only need to be completed from time to time. As a consequence, the computational cost falls down. This is the global behavior of the algorithm. In addition, the image points are shaded in the scan-line order. This allows consecutively

handling neighbor points which are likely to share the same occlusion data. This is noticeable locally: An "expensive" query is always followed by "cheaper" queries, taking advantage of the previous computational effort.

2.6 Discussion and Future Work

In this section we discuss our algorithm and point out some limitations.

The lazy construction of the occlusion data is an important feature of our algorithm. The construction is driven by the occlusion queries, which allows the algorithm to fit the image resolution and to focus the computational effort where and when it is needed. On the other hand, such an approach cannot solve all types of visibility problems. For example, it is not suitable to prove that the light source is invisible from a polygon or to compute exact Potentially Visible Sets. Indeed, it is not possible to query all the points on a polygon or on the PVS boundaries. However, using an adaptive sampling of the surfaces, our approach could still be used to query the visibility from each sample and produce an accurate solution to these problems.

In the current implementation, a BSP tree is built for each triangle with image points to shade and makes the most of the visibility coherence between those points. This is an easy solution to group image points. However, if the geometric resolution is very high with respect to the image resolution, it can lead to very few image points per triangle and thus, a loss of efficiency, because the occlusion data may be dropped before being re-used. To overcome this problem we plane to develop an approach independent from the geometry. We are thinking about mapping image points into a Bounding Volume Hierarchy built to balance the number of image points per leaf. Next, the light source visibility could be computed per leaf, using its bounding box faces to apply our visibility algorithm. This would probably require to solve the self occlusions that may occur inside a bounding box. As a future work this is our main idea since it would solve any problems related to the image versus object resolution. For example, it could even handled micro-polygons.

***

This chapter has presented our from-polygon occlusion algorithm, as well as it application in the context of soft shadow generation. The next step is to build upon this work and make the step towards a from-polygon visibility representation.

Chapter 3 :

From-Polygon Visibility
Application to Ambient Occlusion

IN the first chapter of this thesis, we described a theoretical framework based on the Plücker parametrization, which can be used to solve from-polygon visibility problems. At that point, we have outlined the void which exists between the theoretical tools and the existing implementations. In this context, our objective was to provide a robust visibility algorithm which can be successfully used in a given applicative context.

The first step towards this objective is our from-polygon occlusion algorithm and its application to the generation of soft shadows. In the previous chapter, we have explained how we can simplify Pellegrini's theoretical framework, and use it to determine the coherent sets of lines which describe the occlusion of a light source, as seen through a group of occluders. Although such an occlusion data is sufficient for some applications, other require more complex information. For example, in order to calculate the ambient occlusion for a point, we need to determine which polygons are directly visible from the point. This would require an extra depth information, which is not present neither in Pellegrini's theoretical framework, nor in our from-polygon occlusion algorithm.

Pellegrini's theoretical framework allows to analytically group lines together, according to the triangles they intersect. As explained in Chapter 1, Section 1.2.2, this corresponds to an information of occlusion. The *equivalence* classes, as defined by the theoretical framework, provide an indication on which geometry is stabbed, but not the order in which these polygons are intersected by the lines belonging to the *equivalence* class. Moreover, since we are dealing with random lines in space, the notion of first intersection cannot be properly defined.

Our simplification, as presented in the previous chapter, only distinguishes between *occluded* and *unoccluded* lines. These are the lines stabbing an area light source and which are partitioned according to a set of occluders. The next step is to design an algorithm which encodes the exact visibility information from a source polygon. More exactly, given a polygon and a set of occluders, we want to know which of these polygons, or fragments of these polygons, are directly visible from the source polygon. Therefore, we consider the set of rays originating from the source polygon. These rays need to be analytically grouped according to the first geometry they intersect. Note that in this context, contrary to the theoretical framework, the notion of first intersection can be defined with respect to the source polygon.

Our new algorithm retains all the advantages of the method described in the previous chapter: it avoids all complex 5D CSG operations, thus being very accurate and robust. Also, since we are only interested in the ambient occlusion for the points visible from the camera, our algorithm executes the visibility queries lazily at run time, when and where the information is required.

The first part of this chapter details the ambient occlusion theory (Section 3.1), and more exactly the *obscurances* and *ambient occlusion* illumination models. Next, we review the main techniques used to calculate ambient occlusion (Section 3.2). We equally present an analytic solution to the ambient occlusion integral, which will be useful in the context of this work (Section 3.1.4). We

then provide a description of our method from a theoretical point of view (Section 3.3), followed by the practical aspects of our implementation (Section 3.4). We also detail how we can enhance our ambient occlusion algorithm to take into account a *falloff* function (Section 3.5). After describing our ambient occlusion framework (Section 3.6), we present the results we obtained (Section 3.7), followed by a discussion and some perspectives (Section 3.8).

The work presented in this chapter has been published as

Analytic ambient occlusion using exact from-polygon visibility
O. Apostu, F. Mora, D. Ghazanfarpour, L. Aveneau
in
Computers & Graphics,
Volume 36, Issue 6, pages 727–739, October 2012

3.1 Ambient Occlusion Theory

Ambient occlusion is an empirical illumination model used to simulate global illumination effects, at a less expensive cost. It was first introduced by Zhukov *et al.* [ZIK98], under the name of *obscurances*, and later adapted to the movie industry [Lan02, Bre02]. Roughly speaking, obscurances and ambient occlusion represent a geometrical property of a point which approximates the amount of ambient light blocked by the directly visible geometry close to the point. Moreover, the model does not take into account the color of the objects, or the properties of their materials. The following subsection briefly presents the obscurances illumination model, followed by a more complex analysis of its simplification, ambient occlusion.

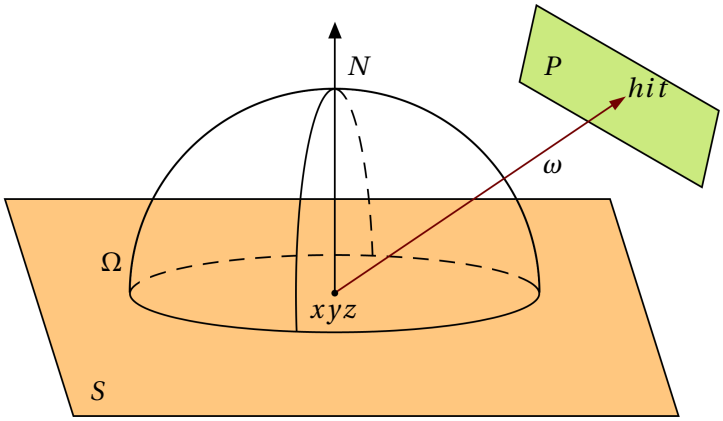


Figure 3.1: Geometry for obscurances / ambient occlusion for point xyz on surface S. The point hit on polygon P is visible from xyz in direction ω. Thus, $d(xyz,\omega) = d(xyz,hit)$.

3.1.1 The Obscurances Illumination Model

Let S be a surface with normal N, and xyz a point on S. The obscurances of xyz is defined by the following integral:

$$Obs(xyz) = \frac{1}{\pi}\int_{\omega\in\Omega} \rho(d(xyz,\omega))(N\cdot\omega)d\omega \tag{3.1}$$

Where

- Ω denotes the upper hemisphere with respect to N,
- $d(xyz,\omega)$ is the distance from xyz to the first intersection in direction ω,
- $\rho(d(xyz,\omega))$ is an empirical attenuation function (*falloff function*), dependent on the distance to the intersected surface in direction ω,
- $\frac{1}{\pi}$ is a normalization factor, such as if $\rho(d(xyz,\omega)) = 1, \forall\omega \in \Omega$, then $Obs(xyz) = 1$.

Figure 3.1 provides an illustration.

The ρ function. The obscurances model makes the assumption that the farther a point is obscured in a given direction, the more light is coming from that direction. This attenuation with respect to the distance is translated by the ρ function. Therefore, ρ needs to be a monotone increasing function of the distance d.

Moreover, obscurances is a local property. The visible geometry that blocks the incoming light is only considered in a given environment around the point. This is based on the same assumption that objects which are too far away will no longer be able to influence the amount of light reaching the point. Thus, ρ also needs to be up bounded. Let δ be the maximum distance around xyz for which we are considering occlusions. The function ρ is defined on the following interval

$$\rho : [0, \delta] \rightarrow [0, 1] \tag{3.2}$$

and has a shape approximated by the plot in Figure 3.2. Since no formal definition of ρ is given by the original article [ZIK98], various formulas have been used. In [MSN03], Méndez et al. suggested two possible functions (Equations 3.3 and 3.4) and presented their results.

$$\rho(d) = 1 - e^{\frac{d}{\delta}} \tag{3.3}$$

$$\rho(d) = \sqrt{\frac{d}{\delta}} \tag{3.4}$$

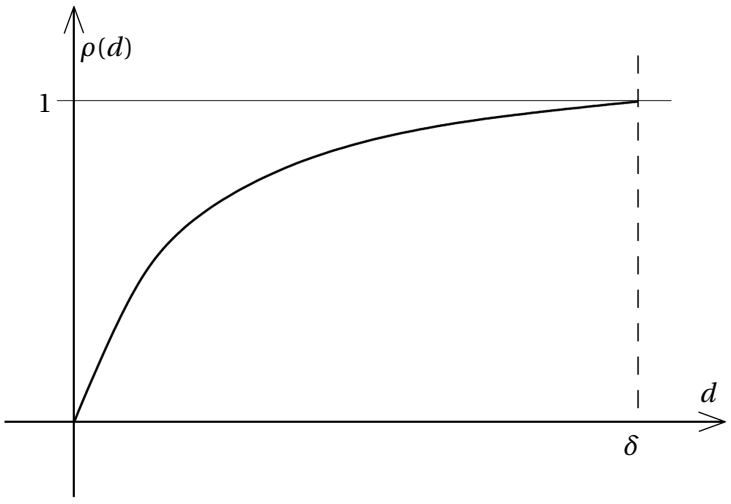


Figure 3.2: Graph of function ρ.

The δ parameter. All geometry located within a δ radius from the point is taken into account when calculating the obscurances for the point. This parameter is usually chosen by the user and it determines the amount of shadow contained in the final result. Although it has to be in concordance

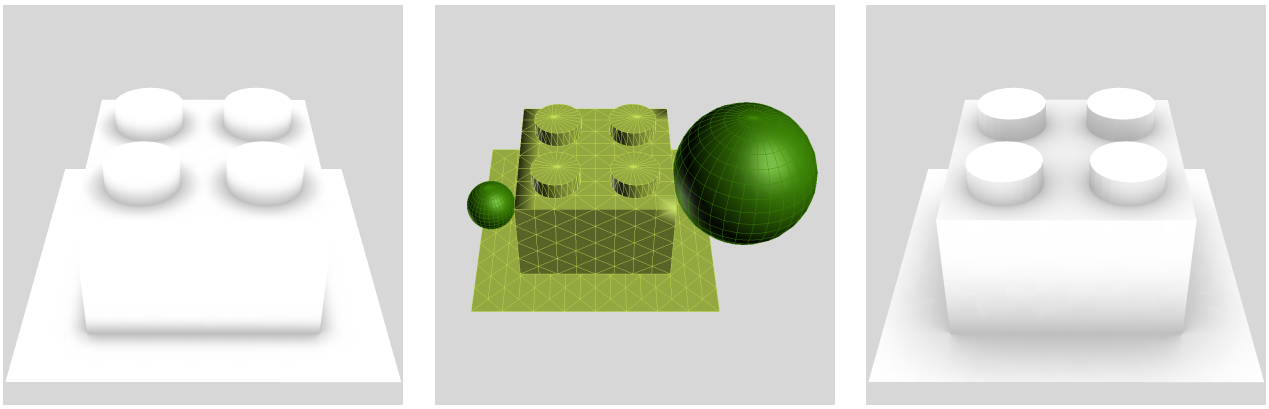


Figure 3.3: Impact of the δ parameter. The two bricks were rendered using two different δ values (illustrated by the two spheres in the central image). From a practical point of view, both values are correct since they produce shadowing effects and allow a good comprehension of the scene. However, the question of which image is better / more pleasing remains a subjective matter and greatly dependent on the context in which the result will be integrated.

with the size of the different elements in the scene, its value is rather artistic. Figure 3.3 gives an illustration by showing the same scene rendered using two different δ values.

3.1.2 The Ambient Occlusion Illumination Model

Applying an attenuation function for each intersection with the surrounding environment can be an expensive task. This can be challenging especially for games, where the rendering speed is crucial. Also, in the movie industry, the main concern is to obtain a visually pleasant effect. The exactness of the result is less important. Usually, the ambient light is attenuated in the final composition phase, according to the influence of the visible surfaces. Thus, a simplification of the obscurances model, known as ambient occlusion, was introduced.

The new illumination model replaces the ρ function with a *visibility function*:

$$AO(xyz) = \frac{1}{\pi}\int_{\omega\in\Omega} vis(xyz,\omega)(N\cdot\omega)d\omega \tag{3.5}$$

In this context, we retain the following definition of the visibility function:

$$vis(xyz,\omega) = \begin{cases} 1 & \text{, if an object is visible in direction } \omega \\ 0 & \text{, otherwise} \end{cases} \tag{3.6}$$

Note that some methods use the complementary definition ($vis(xyz,\omega) = 0$, if an object is visible in direction ω, and 1 otherwise), which is directly deductible from the $\rho(d(xyz,\omega))$ function.

3.1.3 Differences and Similarities

Both ambient occlusion and obscurances represent a purely geometric property of a point. Their effect can be seen as an interpretation of the point's position with respect to its surrounding environment. Therefore, their computation is independent of the light sources or the illumination model used for rendering. Moreover, ambient occlusion and obscurances allow the visualization of scene without any light source, since they translate the geometric position of objects.

The main difference between obscurances and ambient occlusion is the attenuation function considered by the first method. According to [MFS09], the obscurances technique respects the geometry of the scene by taking into account the distance between the points and the surfaces which occlude them. On the other hand, ambient occlusion considers all geometry within a given radius to be equally distant. However, in today's research papers, the difference between the two methods is fading more and more. Recent research on ambient occlusion include various types of attenuation functions or various approximations of the original formulas in order to achieve different effects or to take into account certain types of scenes.

In this study we consider the definition of ambient occlusion as given by Equation 3.5. However, our aim is to design an algorithm which can either obtain basic ambient occlusion effects, or use the base definition together with an attenuation function.

We present an analytic method based on applying a closed form solution of the ambient occlusion integral in the context of exact visibility, in order to achieve noise free and high quality results. More exactly, we use from-polygon visibility to exploit the visual coherence which exists between the points belonging to the same surface. Such neighbor points share similar views of their surrounding environment, and thus have close ambient occlusion values.

It is important to note that for the ambient occlusion integral, a closed form solution exists. More exactly, this evaluation is frequently used in illumination theory to compute the irradiance function, or the form factors corresponding to the radiative transfer between a point and a polygon. All these solutions are equivalent to the base ambient occlusion integral 3.5. In the next subsection, we explain how the closed form of this integral can be obtained, and indicate some related bibliographic resources.

3.1.4 An Analytic Solution to the Ambient Occlusion Integral

S, xyz and N retain their previous definitions. Let P be a convex polygon which is entirely visible from xyz. The goal is to produce a closed form evaluation of the ambient occlusion due to P. In fact, such an evaluation already exists in illumination theory and it is used in the computation of irradiance. In physically-based rendering, direct lighting is expressed using the surface irradiance function. Irradiance is a physical quantity which measures the radiant energy incident on a surface [CWH93]:

$$I(xyz) = \int_{\omega \in \Omega} L(\omega)(N \cdot \omega) d\omega \tag{3.7}$$

Where $L(\omega)$ corresponds to the radiance arriving from direction ω and Ω denotes the upper hemisphere with respect to the local surface normal.

Two assumptions are made in the context of this study: All polygons are ideal diffuse surfaces and each polygon P emits constant radiance L_P uniformly in all directions across its surface. Therefore, the radiance term in Equation 3.7 becomes a function of position only and can be moved outside the integral [SP94]:

$$I(xyz) = \int_{\omega \in \Omega} L(\omega)(N \cdot \omega) d\omega = L(xyz) \int_{\omega \in \Omega} vis(xyz, \omega)(N \cdot \omega) d\omega \tag{3.8}$$

For a completely visible polygon P, the irradiance integral (Equation 3.7) can be rewritten as a surface integral over P.

$$I(xyz, P) = L_P \int_P (N \cdot \omega) d\omega \tag{3.9}$$

This surface integral can be reduced to a line integral over the boundary of P (contour integral), which can be evaluated in closed form using Lambert's formula [Lam60]:

$$I(xyz, P) = \frac{L_P}{2} \sum_{i=1}^{n} \beta_i \cos \gamma_i \tag{3.10}$$

Where n is the number of vertices of P, β_i is the angle (in radians) subtended by edge i of P, and γ_i is the angle between N and the outward normal of the plane defined by xyz and the edge i. Figure 3.4 provides an illustration.

This analytic solution can be applied only for planar convex polygons and under the assumption that the polygon is fully visible from the point. The detailed, yet far from trivial, demonstrations to Equations 3.9 and 3.10 can be found in [Sta02].

On the other hand, the occlusion by P of ambient light can be represented by the radiant energy emitted by P and incident at xyz. Thus, the ambient occlusion equation can be deducted from irradiance:

$$\begin{aligned} I(xyz, P) &= L_P \int_E (N \cdot \omega) d\omega \\ &= \pi L_P (\frac{1}{\pi} \int_E (N \cdot \omega) d\omega) = \pi L_P AO(xyz, E) \end{aligned} \tag{3.11}$$

By replacing the left term with the result from Equation 3.10, we obtain a closed form expression for the ambient occlusion due to an entirely visible polygon P:

$$\begin{aligned} AO(xyz,P) &= \frac{1}{2\pi}\sum_{i=1}^{n}\beta_i\cos\gamma_i \\ &= \frac{1}{2\pi}\sum_{i=0}^{n}(\cos^{-1}\frac{v_i\cdot v_{i+1}}{\|v_i\|\cdot\|v_{i+1}\|})(\frac{v_i\times v_{i+1}}{\|v_i\times v_{i+1}\|}\cdot N) \end{aligned} \tag{3.12}$$

Where n, β_i and γ_i retain their previous definitions and $v_i = p_i - xyz$ (the vectors defined by the point xyz and each vertex of P). If there is more than one blocking polygon, the total ambient occlusion is the sum of the integrals over each surface. As previously underlined, this is valid under the assumption that each blocking polygon is fully visible from xyz.

Equation 3.12 can also be deducted using form factor theory. According to the initial definition [ZIK98], the ambient occlusion due to a polygon P is equivalent to the form factor between xyz and P, since form factors represent the fraction of radiant flux transfer between P and xyz. In the simplified case of a point and a directly visible polygon, the form factor can be evaluated in closed form using Lambert's formula [Lam60], which yields the same definition as Equation 3.12.

An important property which can be illustrated using Equation 3.12 is that projectively equivalent polygons produce the same irradiance, and thus ambient occlusion. The angles β_i represent the lengths of P's edges when projected onto the unit sphere centered at xyz, and the angles γ_i indicate their position.

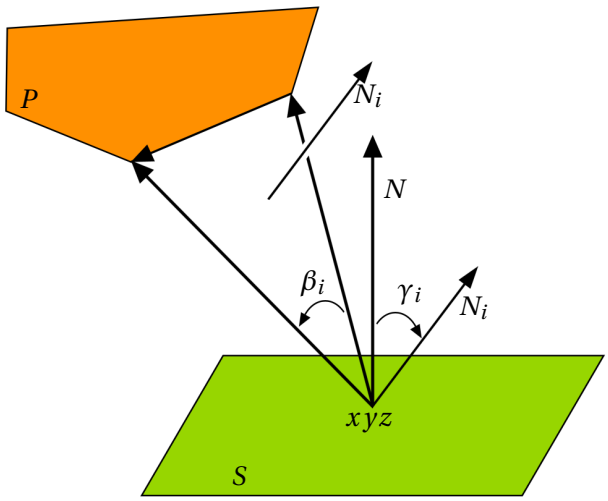


Figure 3.4: Geometry for Lambert's formula.

The next section focuses on the different ambient occlusion computation techniques. We briefly review the major directions and provide a more detailed analysis of the methods directly related to this study. A more comprehensive survey of the different obscurances and ambient occlusion techniques has been provided by Méndez and Sbert [MFS09].

3.2 Ambient Occlusion Computation

According to [ZIK98], the ambient occlusion for a fully visible polygon resembles the form factor corresponding to the diffuse radiative transfer between the polygon and the considered point. This involves a visibility calculation in order to determine the exact visible geometry from a point, and an evaluation of the integral in Equation 3.5. Usually, approximations are made when performing both these steps. These approximations can lead to visual artifacts or noise in the final images. In this survey we focus on how the current methods deal with these two problems: the visibility of the environment and the evaluation of the ambient occlusion integral.

We start with a short description of the methods which introduced the notions of *obscurances* and *ambient occlusion* (Section 3.2.1). Next, we present the ray traced ambient occlusion, which is considered to be the reference computation method (Section 3.2.2). As explained in the previous chapter, the ray tracing solutions are based on sampling the visibility of a polygon (soft shadows) or of an entire environment (ambient occlusion). This solves both the visibility and the evaluation of the integral, and can achieve quality results if the number of samples is sufficiently high. We then discuss the techniques which either propose analytic solutions or achieve results matching those produced by a ray tracer (Section 3.2.3). In this context, the analytic methods are those techniques which are based on a closed form evaluation of the ambient occlusion integral. Their advantage is that the results are no longer subject to noise, as in the case of the ray tracing method. However, the approximated visibility can lead to visual artifacts. Next, we briefly review various ambient occlusion methods, which are mostly adapted to specific contexts (Section 3.2.4). Finally, we summarize the advantages and drawbacks of all the techniques presented in this chapter, and conclude with a few indications on our own algorithm (Section 3.2.5).

3.2.1 Calculating Ambient Occlusion : The First Milestones

Direct illumination models simulate indirect lighting using a constant *ambient term*. This represents a crude approximation which ignores the geometry of the scene and the different occlusions between the objects. Therefore, it is computationally inexpensive. However, such a simplification usually results in a lack of realism and richness in the final images. Thus, various research have constructed on the classic concept of ambient term in order to provide better alternatives, without paying the computational price of global illumination methods.

Before Ambient Occlusion. The first animated short movie, "The Adventures of André and Wally B." (1985), used a particle system shading model in order to approximate the indirect illumination effects on trees and grass. Their model used attenuation distances, and a random probabilistic component which decided if a vegetation particle needs to be shaded or not [RB85]. Almost ten years later, Miller [Mil94] defines the *accessibility* of a surface and proposes two algorithms that shade the more hidden parts of the objects in different ways than the rest of the scene. In [CNS00], Castro *et al.* replace the classic constant ambient term with a set of terms which take into account the orientation of the polygons in the scene. In order to remain computationally inexpensive, the proposed method

does not calculate the occlusions between the various objects.

An Ambient Light Illumination Model. The *Obscurances* illumination model is formally defined in [ZIK98]. Zhukov *et al.* describe "an empirical ambient light illumination model", which simulates the darkening of the more hidden areas as a result of the lack of secondary light ray reflections. The model is defined using the integral in Equation 3.1, and a physical justification is attempted. Although the stated conclusion is that the physical foundation is empiric, the model remains interesting because it successfully simulates indirect illumination without being expensive. A computation method is also suggested, based on the resemblance between a patch's obscurances and form factors.

A New Star is Born. At SIGGRAPH 2002, Hayden Landis [Lan02] from *Industrial Light + Magic* and Rob Bredow [Bre02] from *Sony Pictures Imageworks* each spoke about *ambient occlusion*, a technique that had been used in the movies *Pearl Harbor* and *Stuart Little 2*, respectively. In the first course, ambient occlusion for a point was calculated by casting rays in a hemisphere around the surface normal. This was used to shadow the surfaces that were less exposed, and also to calculate a *bent normal*, a direction obtained by averaging the unoccluded directions. This allowed obtaining the proper lighting for the unoccluded geometry. In the second course, ambient occlusion was simulated using two large area lights, representing the sky and the ground, respectively. The method had been implemented in the widely known *RenderMan* software.

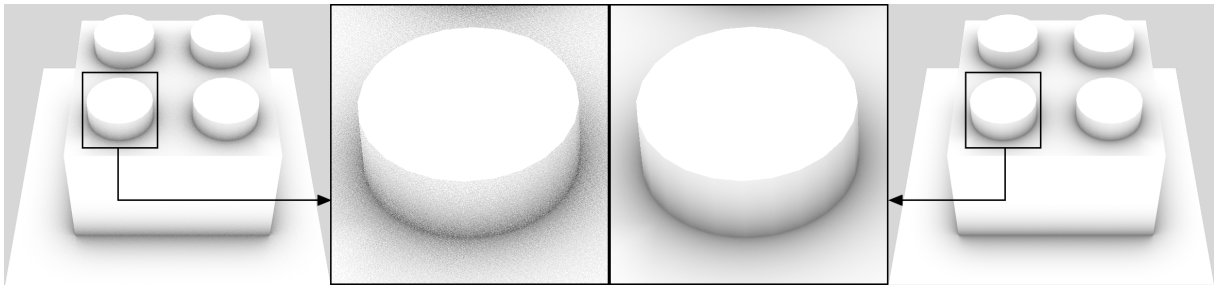
3.2.2 Ray Tracing

One way to test the visibility from a point xyz is to cast rays from the point into the surrounding environment and test for occlusions. The integral in Equation 3.5 (or Equation 3.1 for obscurances) can be evaluated using Monte Carlo integration [Nie92]. The point's upper hemisphere is sampled in order to obtain the directions for the rays which will calculate the visibility from xyz. Suppose ns is the number of samples and $\{\vec{R}_1,...,\vec{R}_{ns}\}$ denote the rays corresponding to the chosen point distribution. Equation 3.5 can be estimated using the following sum:

$$AO(xyz) = \frac{1}{ns}\sum_{i=1}^{ns} vis(xyz,\vec{R}_i)(N\cdot\vec{R}_i) \tag{3.13}$$

Note that for the obscurances models, the visibility function is replaced with the ρ function.

The ray traced ambient occlusion has the same disadvantages as any other sampling based method. The quality of the final images is dependent on the number of rays traced and the results usually contain an important amount of noise. Moreover, this problem is more accentuated when the δ parameter is increased, since the angular dispersion of the rays is more important. An illustration is given in Figure 3.5. The sampling technique also has an impact, as shown in [MS04]. A more comprehensive analysis of ray tracing optimizations and sampling distribution improvements has been provided in Chapter 2, Section 2.1.1.



a) 8 rays / pixel *b) 8 rays / pixel (zoom)* *c) 256 rays / pixel (zoom)* *d) 256 rays / pixel*

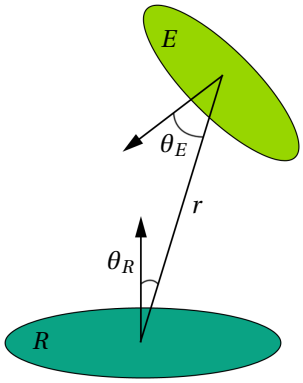
Figure 3.5: The impact of the number of samples. The images obtained using ray traced ambient occlusion are sensitive to noise according to the number of considered samples. Two examples are given here, using 8 rays for each pixel (a), b)) and 256 rays (c), d)). The noise is clearly visible in the left images, and bearly noticeable in the right ones.

3.2.3 Analytic and High Quality Solutions

Dynamic Ambient Occlusion and Indirect Lighting. Bunnell [Bun05] represents the scene by a hierarchical structure of disks. A disk is created for every vertex of every polygon. Each disk has a position and a normal, derived from the original polygonal information, and an area, which is calculated based on the areas of the polygons sharing the vertex. Thus, the disks act as an approximation of the scene's surfaces, in order to facilitate illumination and shadowing computation. This data is stored into a texture map, so it can be easily accessed and updated using a GPU fragment program. The occlusion between these disks is analytically approximated using a solid angle formula, as illustrated in Figure 3.6. The actual visibility between two surface elements is never calculated. The occlusion for one disk is simply a sum of shadow contributions from neighbor disks. Approximating visibility using occlusion, along with the considered disk representation, have a negative impact on the resulted images. The occlusion is over-estimated and thus, some areas appear darker than expected and an important amount of detail is lost. Several passes are required in order to attenuate the excessive shadowing, and their number needs to be carefully chosen since there is the risk of surfaces becoming too light.

High-Quality Ambient Occlusion. Hoberock and Jia [HJ07] construct on Bunnell's [Bun05] approach to improve its robustness and the quality of the results. In order to avoid boundary artifacts, a transition zone is defined where occlusion is smoothly interpolated, according to the position and the hierarchy of the disks. Instead of placing disks at each vertex of the scene's geometry, each polygon is approximated by a disk located at its barycenter. Moreover, the disks at the lowest level of the hierarchy are replaced with the actual polygons. This allows two optimizations: First of all, the polygons can be clipped against the support plane of the point to be shaded. This gives a better approximation of visibility. Second, the occlusion due to the obtained fragment is calculated using an analytic expression derived from form factors. Figure 3.7 illustrates the two operations. Note that for disks located in the higher levels of the hierarchy, the occlusion is still approximated as a sum of shadow contributions from its neighborhood. Similarly to the initial method [Bun05], the

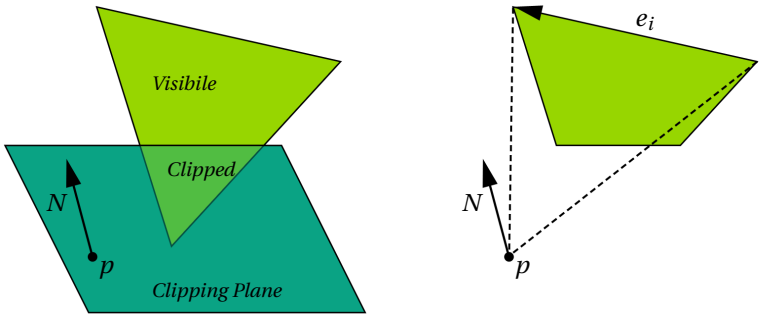
algorithm needs several passes to converge. Since the visibility is approximated, some artifacts are still noticeable and the obtained images do not always match a ray traced result, as the authors mention. Moreover, over-occlusion remains an issue.



$$Shadow(E \rightarrow R) =$$

$$1 - \frac{r \cos\theta_E max(1, 4\cos\theta_R)}{\sqrt{\frac{area(E)}{\pi} + r^2}}$$

Figure 3.6: Shadow Approximation [Bun05]. **Left**: The geometric relationship beteen an emitter and a receiver. E is the disk casting the shadow (emitter) and R is the disk which is shadowed (receiver). r is the distance beteewn the centers of the two disks. **Right**: An approximation of the disk-to-disk occlusion.



$$F(A \rightarrow p) =$$

$$= \frac{1}{2\pi} \sum_i N \cdot \Gamma_i$$

$$where$$

$$\Gamma_i = normal(plane(p, e_i))$$

Figure 3.7: Ambient Occlusion calculation in [HJ07]. **Left**: Visibility calculations al the lowest levels of the disk hierarchy. If a triangle is partially visible from point p, it will be clipped against the plane contain p and having the normal N. **Center and right**: The occlusion due to the fragment is calculated using an analytic formula derived from form factors.

Ambient Occlusion Volumes. McGuire [McG10] proposes an analytical method which yields noise-free results. He builds an ambient occlusion volume for each polygon and locates the visible points in the volumes using rasterization. Then for each visible point, its ambient occlusion value is calculated using Lambert's formula [Lam60], attenuated by a falloff function. The size of the occlusion volumes is dependent on the polygon itself and the maximum occlusion distance considered for the scene. An illustration is given in Figure 3.8. The use of an analytic solution to the ambient occlusion integral provides noise-less quality results. However, the visibility from the point to shade is approximated, which results in visual artifacts. More exactly, the visible points are simply located in the bounding volumes of the polygons which affect them. Thus, if one bounding volume

is entirely contained into another, over-occlusion occurs. This results in over-darkened areas and a loss of details. The same artifact also arises in some configurations when bounding volumes overlap. A solution based on an empiric compensation map is proposed. Although it attenuates the obscured parts, none of the missing details can be recovered.

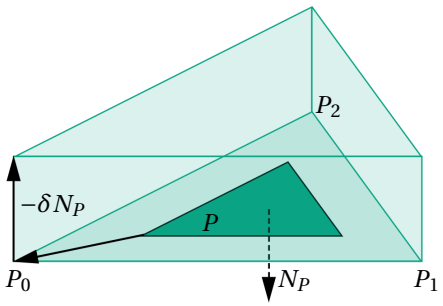


Figure 3.8: Ambient Occlusion Volumes [McG10]. **Left**: The occlusion volume for polygon P. δ represents the maximum occlusion distance for the scene. **Right**: The occlusion of P over a point x contained in P's occlusion volume is calculated using Lambert's formula.

Two Methods for Fast Ray-Cast Ambient Occlusion. Laine and Karras [LK10] propose a GPU method, which computes high quality ambient occlusion and which is targeted for rasterization-based engines. The bounding volumes are replaced with regions of influence, which are more compact. Also, the triangles which are smaller than the the occlusion radius are handled separately, in order to avoid the over-occlusion artifacts resulting from McGuire's [McG10] algorithm. Moreover, the authors propose a technique to avoid over counting occlusions. However, they replace the analytical evaluation of the ambient occlusion integral with an improved stochastic sampling. A second ray tracing method using BVH traversal is presented. An important feature of the solution proposed by Laine and Karras is the decomposition of the computations according to the near-field and far-field separation [AFO05]. The authors explain that the distant geometry has a very low contribution to the ambient occlusion and can be calculated using simplified geometry, without introducing artifacts or perturb the quality of the result. The algorithm thus constructs various approximations of the initial geometry, depending on the distance from the receiver point. The results remain noise-free, although they are no longer exact. Moreover, artifacts may appear if the geometry representation is too coarse.

Point-Based Approximate Color Bleeding. Christensen [Chr08] describes in a Pixar Technical Memo a new point-based method for computing diffuse global illumination. The technique is designed to achieve color bleeding effects and can be easily applied to obtain various effects such as ambient occlusion, soft shadows, glossy reflections and many other. The method works in two phases.

First of all, a pre-computation phase creates a point cloud representation of the directly illuminated geometry in the scene. For each surface element (surfel), its direct illumination value is calculated,

by taking into account the various light sources, and the characteristics added by the surface shaders. This data is saved in the point cloud, which is organized using an octree. For each node, the algorithm calculates a spherical harmonic representation of the power emitted from the surfels in the node. Secondly, during the rendering phase, the global illumination value for each surface point is calculated by traversing the octree and adding the contributions from the relevant surfels. In order to determine the surfels which are to be considered for each point, a low-resolution image of the scene as seen from the point is calculated using rasterization.

The method achieves noise-free high quality images, while being faster than a ray traced solution. Since this technique is dedicated solution for the movie production industry, the main criteria is visually acceptable and consistent results. Accuracy is not an issue, and the author underlines several times that the method is an approximation and that its results are not as precise as ray tracing solutions. Moreover, a series of errors may occur. These are mainly caused by the spherical harmonics approximation, the rasterization phase, and the small or hidden objects which may give incorrect contributions.

3.2.4 Other Methods

Obscurances and Beyond. Méndez *et al.* propose a series of articles which build on the initial definition of obscurances in order to add various features, by taking into account the color of the objects, and some properties of their materials. In [MSN03] they compute view dependent obscurances which handle non diffuse environments using ray tracing. The integral in Equation 3.1 is modified in order to take into account the reflectivity of a point, as seen from the point for which obscurances are calculated. This allows adding a *color bleeding* effect, a well known radiosity effect which colors surfaces using the reflection of colored light from neighbor objects [MSC03]. Moreover, obscurances are also discussed in the context of particular surfaces [MS06], such as specular, transparent or translucent. In all this cases, obscurances for a point is either calculated using the ray tracing technique described in Section 3.2.2, or by *depth peeling* [MFSC+06], a GPU technique that extracts visibility layers from the scene. In the latter case, the obscurances values are approximated using an average of the calculations done for each pair of layers.

The Quest for Speed. In the movie industry, ambient occlusion methods were used in off-line rendering to add realism to the final images. The obscurances illumination model was used in games [IKSZ03], but as a pre-process step only. Real time update was possible only for a small number of polygons. The development of graphic hardware and the game industry motivated a series of research that proposed various methods of computing ambient occlusion in the context of real time applications.

Sattler *et al.* [SSZK04] propose a method which runs without pre-calculations and handles inter object occlusions and deformable or animated meshes. The novelty of their computation is that visibility is not calculated in the classical way, from the points they wish to shade, but from a set of directional light sources distributed over the hemisphere around the point (*outside-in* approach).

The geometry, as seen from each light source, is rendered into the depth buffer where visibility is calculated and stored into a visibility matrix. Ambient occlusion is then approximated using this data. Similar to the ray traced method, this approach is dependent on the number of "samples", in this case the light sources. A small distribution results in numerous artifact. In order to reach real time performance, the visibility is sampled only for the vertices of objects. As shown in [KLA04], the methods using this type of calculations suffer from artifacts, due to under-sampling.

Kontkanen and Laine [KL05] propose an object based method which calculates inter-object ambient occlusion. For each object, its occlusion is approximated using a spherical cap, and the information is stored into cube maps, in order to be quickly accessed and used at run-time. Equation 3.5 is adapted to contain the visibility for a spherical cap. The authors state that an analytic evaluation of this equation had been tested, but then abandoned since it was too expensive to apply using their GPU implementation. In order to avoid under-sampling artifacts, a Gaussian reconstruction filter is used.

Screen Space Ambient Occlusion. The main target of the screen space ambient occlusion (SSAO) methods [Mit07] is the gaming industry. Therefore, SSAO algorithms are primarily oriented towards speed and performance, to the detriment of visual quality. Roughly speaking, the ambient occlusion between nearby geometry is crudely approximated in screen space. More exactly, a pixel shader analyzes the scene depth buffer, and for every pixel on the screen it samples and compares the depth values of its neighbors, in order to see if the pixel is occluded or not. This approximation is the source of many limitations: all the occlusion information due to fragments outside the view frustum is lost, far-away occluders are not taken into account and the final images look either noisy or blurry, depending on the sampling filter. Various attempts to address some of these limitations have been made. Shanmugam and Arikan [SA07] propose a multi-pass approach which handles close and distant occlusion separately. Bavoli and Sainz [BS09] improve quality by using enlarged depth images and an enlarged field of view. Ritschel *et al.* [RGS09] propose a method which retains the directional information of the incoming light and adds some color bleeding effects. Although this solutions bring some improvements to the visual quality of the results, SSAO methods remain crude approximations of ambient occlusion.

Specific Environments. There are other studies which target specific applications, such as molecular visualization [TCM06], tree [GSSK05, HPAD06] or character [KA06, KA07] rendering. Tarini *et al.* [TCM06] propose an ambient occlusion model which enhances the real time visualization of molecular environments. The ambient occlusion integral is solved by sampling and the models are composed of only two basic primitives: spheres and cylinders. Garcia *et al.* [GSSK05] use obscurances to simulate indirect illumination of trees. They represent trees using quadrilaterals, which simplifies the rendering process. Visibility is extracted using a depth peeling approach, similar to [MSC03]. In the same context of tree rendering, Hegeman *et al.* [HPAD06] provide a simple approximation to ambient occlusion, which gives visually pleasant results in real time. The trees are represented using a bounding volume (sphere or ellipsoid), which are filled with blocking elements, according

to a chosen distribution. Visibility is then approximated in terms of average probability that a point inside the bounding volume is visible from the outside. Kontkanen *et al.* [KA06] apply ambient occlusion in the context of character animation. They use a set of precomputed reference poses and approximate ambient occlusion as a linear combination of the ambient occlusion values calculated for these poses. Thus, the quality of the results is dependent on the number of reference poses. Kirk *et al.* [KA07] improve this initial approach, by redefining the pose function and introducing an efficient compression method which improves rendering time. The method is based on pre-processed ambient occlusion values, calculated using ray tracing.

All of the above methods provide approximated ambient occlusion models which are based on assumptions valid only in their particular context. Therefore, they are not directly related to this work.

3.2.5 Conclusion

Calculating the ambient occlusion for a point can be divided into two main operations. First of all, the visibility from the point over its surrounding environment needs to be calculated. Then, this visibility information is used to compute the ambient occlusion for the point. An analytic and accurate method would have to detect the exact fragments of geometry which are directly visible from the point and calculate its ambient occlusion value using a closed form solution of the integral in Equation 3.5. However, in practice, both operations are often approximated. In this survey, we have distinguished the various computation methods into three main categories: the standard ray tracing solution, the high quality and analytic methods which are based on closed form evaluations of the ambient occlusion integral, and the rest of the techniques which are usually adapted to a particular context.

Considered as the quality standard, sampling based methods estimate the ambient occlusion equation by using the sum in Equation 3.13. Their results are subject to noise, and an important number of rays needs to be traced in order to attenuate these problems.

On the other hand, the analytic solutions usually use a closed form solution of the ambient occlusion integral, and thus achieve high quality results. However, they suffer from artifacts, due to the approximated visibility.

And finally, there are the other various computation methods presented which provide approximated solutions, often based on their context. The methods which are oriented towards speed will sacrifice accuracy in order to improve the computational cost. This is the mostly the case for the algorithms designed for the game industry. Some algorithms will make simplifications in order to handle deformable geometry or any other geometry interactions that occur during animating, while other methods are applicable in specific environments only and thus take advantage of approximations characteristic of the considered environment. And an important number of techniques answer a demand for visually acceptable results, without paying the computational price of accuracy. Such

techniques are often used in production rendering for movies and animation films.

Therefore, we can conclude that, to our knowledge, no method exists which calculates the exact visibility from a point over its surrounding environment, and uses this information together with a closed form solution to the ambient occlusion integral. The main difficulty in such an approach would be that the visibility needs to be computed for every visible point on each polygon. This can be a very expensive task, especially since the results need to be exact in order to ensure artifact free images.

In this context, our aim is to propose a method which calculates exact from-polygon visibility and uses this information to speed up the visibility calculations from the points belonging to the same surface, by taking advantage of their visual coherence. This is combined with the analytic solution to the ambient occlusion integral provided by Lambert's formula, in order to achieve high quality, noise and artifact free results.

3.3 Algorithm Design

In this section we present our from-polygon visibility algorithm from a theoretical point of view. We also detail the application of our algorithm to calculating analytic and high quality ambient occlusion.

Our approach relies on the same theoretical framework described by Pellegrini. More exactly, we build on the occlusion algorithm presented in Chapter 2, in order to add the missing depth information, and thus calculate the exact visibility from a source polygon. First of all, we explain the theoretical passage from occlusion to visibility (Section 3.3.1). Then, following the same guiding lines as in our previous chapter, we describe the design of the data structure used to encode this information (Section 3.3.2), and how we can extract the visibility for each point of the source surface (Section 3.3.3) and use it to calculate the ambient occlusion information for the point. Since our two methods share a common basis, our description focuses mainly on the evolution from our occlusion to our visibility algorithm.

3.3.1 From Occlusion to Visibility

Our aim is to describe the visibility from a surface S, over the set of all its potential occluders, noted $O(S)$.

Without loss of generality, we assume that the occluders in $O(S)$ are not intersecting each other. Otherwise, the existing intersections have to be handled as a pre-process step.

In Chapter 2 we have restricted Pellegrini's definition of *equivalence* classes to the lines induced by S and all the geometry in $O(S,T)$, where T was a polygon containing the points to shade. Moreover, we have simplified the initial framework, by only distinguishing between *occluded* and *unoccluded* lines. This distinction is no longer sufficient in our new visibility context. Since we need to calculate the complete visibility from the polygon S, we consider all the lines stabbing S and having an orientation which is coherent to the upper normal of S. These *view rays* must be grouped according to the first occluder in $O(S)$ they intersect.

By restricting Pellegrini's definition to the view rays originating from S, we obtain an arrangement where each cell corresponds to either a coherent set of rays missing all the occluders in $O(S)$, or a coherent set of rays stabbing the same subset of occluders. We suppose that this subset is associated with the cell. Since these occluders do not intersect each other, one of them is stabbed before all the other by all the rays in the class. Moreover, all the other geometry associated with the cell are behind the support plane of this occluder, with respect to S. Thus, they can be sorted in order to find the first intersected one. Figure 3.9 provides an illustration. It is important to note that the degenerate case when the support plane of the occluder intersects S is treated by splitting S, as explained in Chapter 2, Section 2.3.

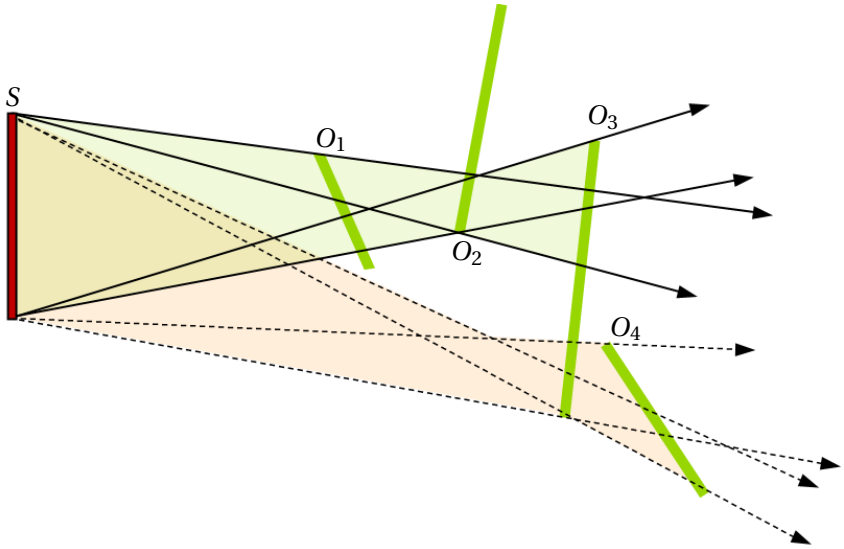


Figure 3.9: If a coherent set of rays stab the same occluders, and these occluders do not intersect each other, one of them is intersected before all the others. In this illustration, S, O_1, O_2 and O_3 are all stabbed by a set of rays, and S, O_3, O_4 are all stabbed by another set of rays. The first set of rays intersects O_1 before intersecting O_2 and O_3, since these two occluders are located behind the support plane of O_1 with respect to S. Similarly, O_3 is intersected before O_4 by the second set of rays.

In conclusion, the rays originating from S can be grouped together according to the first occluder they intersect. The result is continuous sets of rays. This corresponds to an analytical representation of the geometry directly visible from S. Therefore, we have a new *visibility* equivalence relation, with respect to a surface.

Each cell of the new arrangement induced by S and the geometry in $O(S)$ corresponds to either one of these two cases:

- A coherent set of rays originating from S and missing all the occluders in $O(S)$. The Plücker points in this cell correspond to rays belonging to a *free* class.
- A coherent set of rays originating from S and intersecting the same first occluder O before stabbing any other geometry. The Plücker points in this cell correspond to rays describing the visibility of O, as seen from S. These rays belong to a *hit* class. Note that in this case, O is associated with the cell.

Figure 3.10 provides a 2-dimensional illustration of the differences between the *equivalence* classes described by the theoretical framework, our *occlusion* classes defined in the previous chapter, and our new concept of *visibility* classes. This illustration represents a continuation of the example given by Figure 2.8 in Chapter 2, Section 2.2.1. Also, Figure 3.11 gives a 3-dimensional comparison between Pellegrini's definition and our new *visibility* classes.

It is important to note that each visibility class contains both a directional and a depth information.

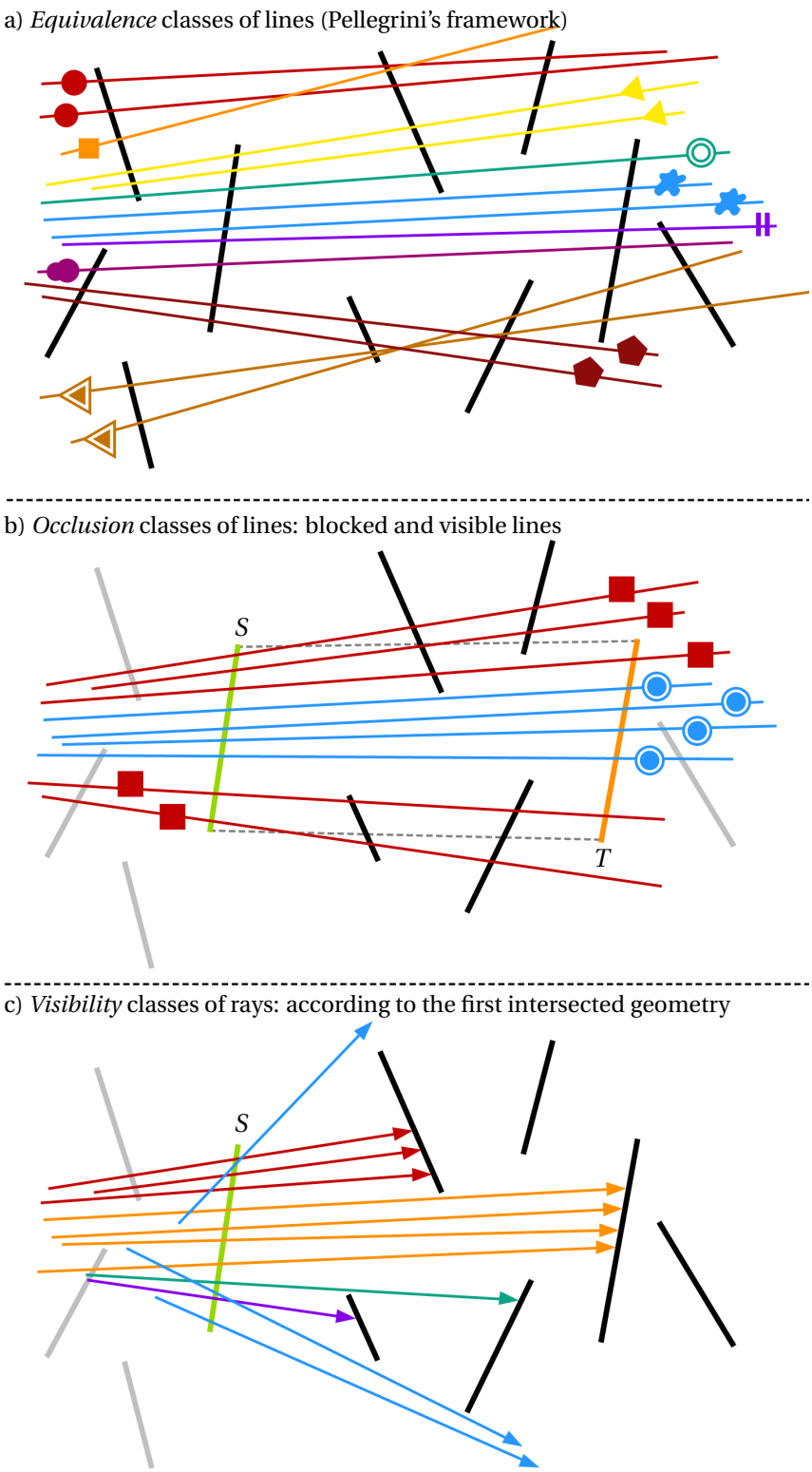


Figure 3.10: Comparison between the arrangement of lines from Pellegrini's theoretical framework (a), the arrangement considered for our from-polygon occlusion algorithm (*occlusion* classes) (b), and the one considered for our from-polygon visibility algorithm (*visibility* classes)(c). The theoretical framework concerns all the lines in Plücker space, and the occlusion algorithm limits the computations to the set of lines stabbing S and the occluders of S and T, and distinguishes between *occluded* and *unoccluded* lines. In the case of from-polygon visibility, we need to group the rays originating from S according to the first triangle they intersect. These new definition of *visibility* classes remains simpler than the theoretical framework, but more complex than the notion of *occlusion* classes.

More exactly, the rays are grouped with respect to the geometry in the scene (directional information) and according to the first triangle they intersect (depth information).

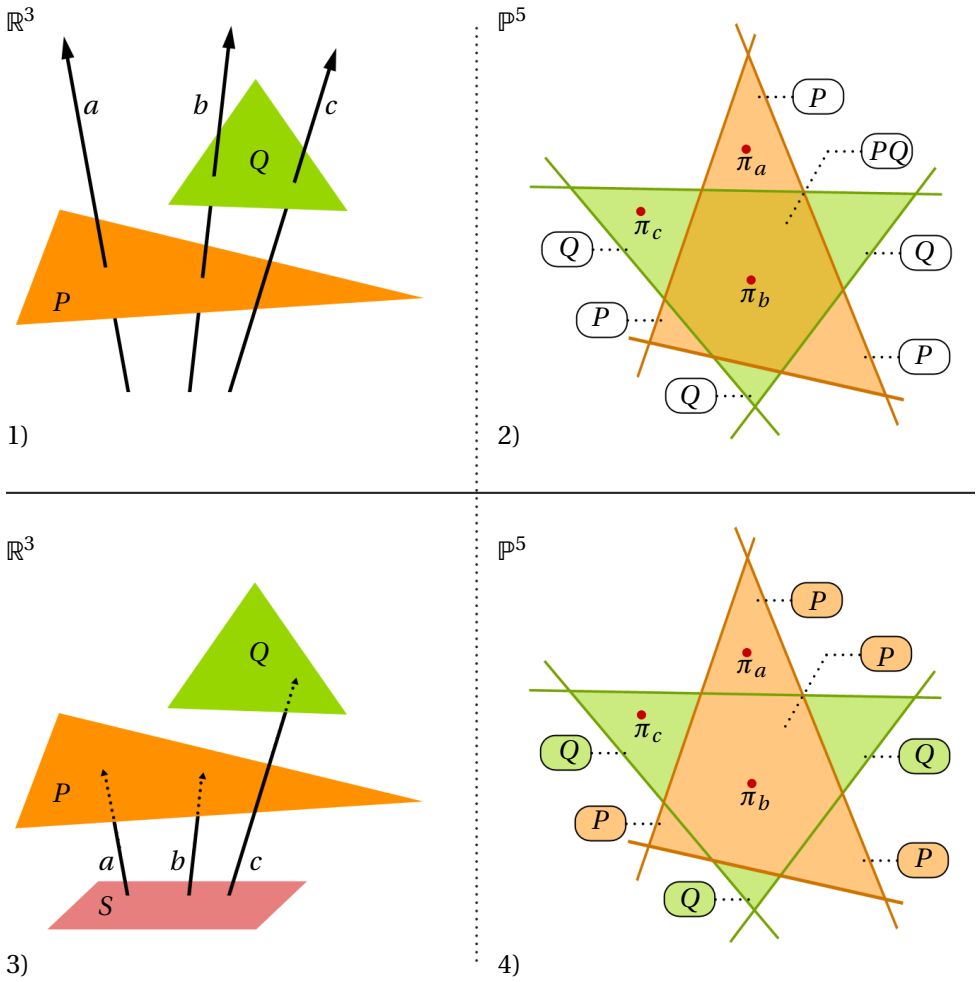


Figure 3.11: Up (1,2): *Equivalence* classes according to Pellegrini. **1)** Two triangles (P, Q) and three lines (a, b, c) in various configurations. **2)** The arrangement of hyperplanes (illustrated by 6 2D lines) mapped from the triangles edges. Filled cells are set of lines intersecting at least one triangle. The intersected triangles are associated with the corresponding cells. π_a, π_b and π_c are the Plücker points mapped from a, b and c, respectively. They are located in the cells according to the triangles they stab. For example, π_b has a consistent orientation with respect to the 6 hyperplanes, since b intersects the two triangles. However, there is no indication on which is the first intersected triangle. **Down (3,4)**: Our new *visibility* classes in the context of visibility from a surface S. **3)** The three lines (a, b, c) can be associated to visibility rays originating from S. **4)** The triangles formerly associated with the central cell have been depth sorted to find the first intersection with respect to S. Thus, only P is associated with the cell and no ambiguity remains on which is the first triangle intersected b.

3.3.2 Encoding the Visibility Information

From an algorithmic point of view, both our algorithms use a BSP tree to store the occlusion / visibility information. However, the data structure needs to be adapted to each algorithm to account for the differences between the two types of information. More exactly, the from-polygon visibility algorithm associates an unique triangle to each *visibility* class. Therefore, we distinguish one *visibility* class

from another. This is an important difference with respect to our from-polygon occlusion algorithm, where all *occlusion* classes had the same significance. Figure 3.12 provides an illustration.

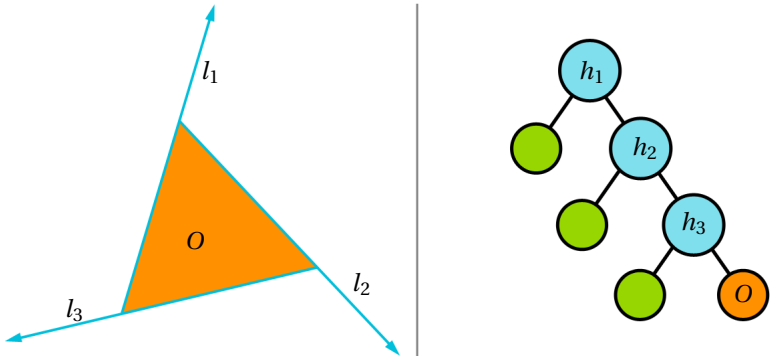


Figure 3.12: The BSP representation of an occluder O. We retain the same notations and the same structure as described for our from-polygon occlusion algorithm and illustrated in Figure 2.9. We can build a BSP tree whose leaves are the four *visibility* classes generated by O: Three *free* classes (left leaves), corresponding to sets of rays missing O, and one *hit* class (right leaf), corresponding to a set of rays intersecting O. However, in the case of from-polygon visibility, we distinguish one *visibility* class from another. Thus, O is associated with the right leaf and used to make the distinction.

Up to this point, we have provided a theoretical description of how we can analytically represent the visibility from a surface, and the design of the data structure used to encode this information. The next section focuses of how the BSP tree can be used to extract the exact visibility from any point xyz located on S.

3.3.3 Extracting the Visibility Information

Let xyz be a point on S. Similarly to the description provided in the previous chapter, the visibility information for xyz is already contained in the data structure and therefore does not need to be calculated, but extracted. The need for such an operation is explained by the fact that an occluder which is visible from S, is not necessarily visible from all the points located on S. Figure 3.13 provides an illustration.

In order to describe the complete visibility from xyz we need to focus on all the view rays originating from it. However, this is not as straightforward as in the case of from-polygon occlusion, mainly because we are no longer limited to two polygons (S and T), but to a single polygon (S) and its entire environment. In the context of from-polygon visibility we start with a single ray passing through xyz and a point above S. This example is then generalized to the view rays originating from xyz and stabbing an arbitrary polygon. Finally, we use the entire upper hemisphere centered at xyz, in order to take into account all the rays issued from xyz. These three cases are illustrated in Figure 3.14.

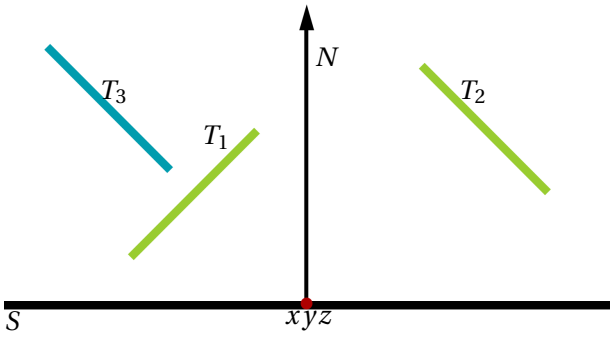


Figure 3.13: The visibility from xyz needs to be extracted from the data source which encodes the exact visibility from the surface S. The occluders T_1, T_2 and T_3 are visible from S. This information is contained in the data structure. However, T_3 is not visible from xyz, thus the need to extract only the relevant visibility information for xyz.

Let p be a point located above S. Note that p may or may not belong to any of the occluders in $O(S)$. The vector $\vec{v} = \overrightarrow{p,xyz}$ defines the direction of a view ray, having xyz as origin. The view ray is located in a leaf of the BSP tree using the same process described in Chapter 2, Section 2.2.3. If the leaf corresponds to a *free* class, than the ray misses all the occluders in $O(S)$, and p is directly visible from xyz. Otherwise, the occluder associated with the class is the first geometry intersected by the ray. In this case, p is not visible from xyz if it is located behind this occluder, with respect to S, and visible otherwise. Figure 3.14.a provides an illustration.

Next, let P be a polygon located above S. xyz and P form a view beam containing all the rays originating from xyz and stabbing P. Using the same method described in Chapter 2, Section 2.2.3, P is split into convex fragments, each representing a coherent sub-set of view rays located in the same leaf. In the context of our visibility algorithm, this translates to the fact that each fragment represents a coherent set of view rays which either intersect the same first occluder, or miss all the geometry in $O(S)$. Figure 3.14.b provides an illustration. Let P_i be such a fragment. In order to decide if P_i is visible or not from xyz, its position with respect to O_i must be determined. This process in detailed below in the context of ambient occlusion computation and illustrated in Figure 3.15.

Calculating Ambient Occlusion

The above operation can be generalized for an arbitrary number of polygons. We describe this generalization in the context of calculating the ambient occlusion for a point xyz. An illustration is provided by Figures 3.14.c and 3.14.d.

Since ambient occlusion is a local property, we need to extract and limit the visibility information to the local environment of each point xyz. This corresponds to the visible geometry located within a δ radius from the point. Thus, we consider a polygonal representation of the upper hemisphere centered at xyz ($hemisphere(xyz,\delta)$). The extraction process described for a single polygon can be applied to all the polygons which form this representation. The result is a set of convex fragments, each one representing a coherent set of view rays originating from xyz and stabbing the same first

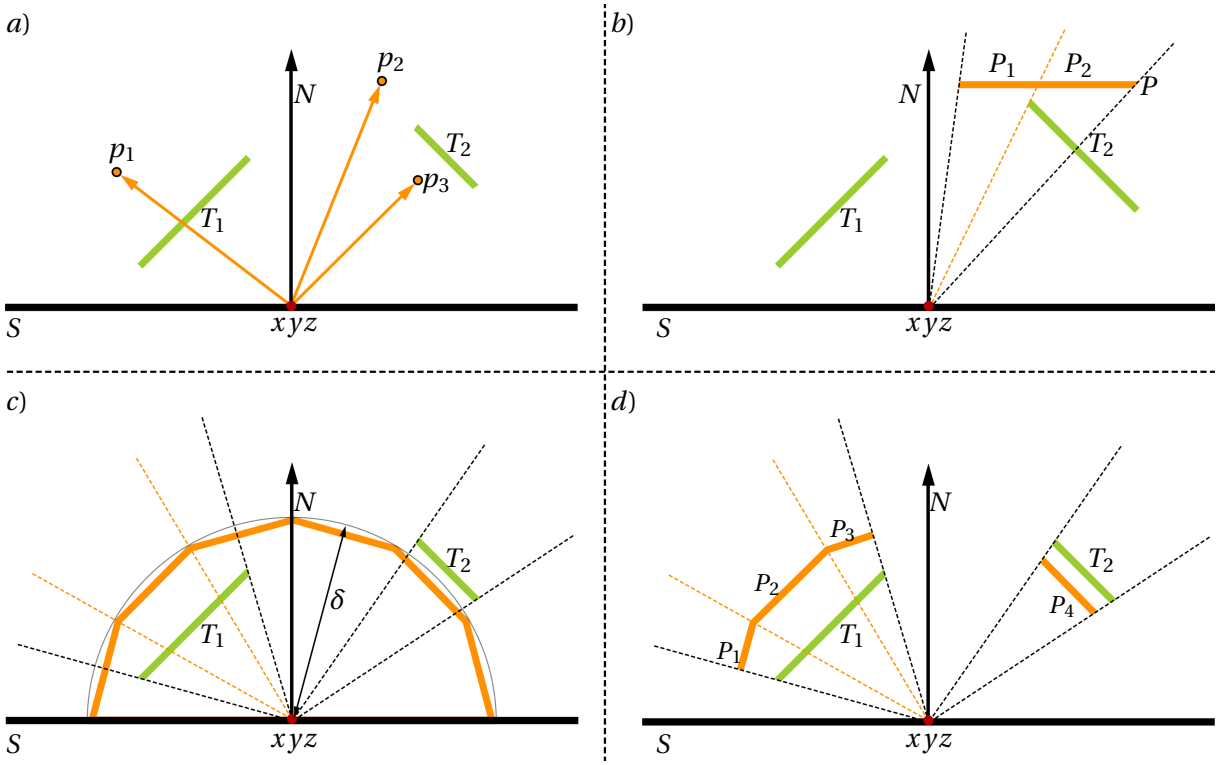


Figure 3.14: Extracting the visibility information for point xyz.
a) We consider three points, p_1, p_2 and p_3, and we want to test if they are visible from xyz. In order to do that, we locate into the data structure the Plücker points corresponding to the rays defined by xyz and p_1, p_2, p_3. This yields the *visibility* classes for each ray. The ray defined by p_1 intersects the occluder T_1. Since p_1 is located behind T_1, with respect to S, it is not visible from xyz. On the other hand, p_3 is located in front of T_2 and thus is visible from xyz. The ray defined by p_2 belongs to a *free* class, and thus p_2 is directly visible from xyz.
b) Let P be a polygon located above S. This polygon is used to guide the visibility extraction process for xyz. This is achieved by locating the set of rays contained in the view beam defined by xyz and P into the BSP tree, and splitting P if necessary. In this figure, we obtain two fragments, P_1 and P_2, the first one corresponding to a *hit* class, and the second one to a *free* class. In the first case, there is no occluder intersected by the rays, and thus P_1 is visible from xyz. In the second case, P_2 is located behind $T2$, with respect to xyz, and thus it is not visible from xyz.
c) The above operation is generalized in the context of ambient occlusion computation, to a set of polygons corresponding to a polygonal representation of the $hemisphere(xyz, \delta)$.
d) In the context of ambient occlusion, we can retain only the fragments of the hemisphere representation which correspond to *hit* classes. In order to detect which occluder influences the ambient occlusion of xyz we study the position of each fragment with respect to the occluder intersected by the set of rays represented by the fragment.

occluder, or missing all the geometry in $O(S)$. The latter case is the simplest, because it corresponds to rays belonging to a *free* class and which have no contribution to the ambient occlusion calculation.

Let P_i be a fragment which represents a coherent set of rays which intersect the same first occluder O_i. As explained in Section 3.1.4, Equation 3.12 yields the same results when applied to projectively equivalent polygons. Therefore, we can either apply it to P_i, or to the exact fragment of O_i which is visible from xyz. However, before performing this operation, the position of the occluder with respect to the fragment needs to be further tested. Three cases can occur (see Figure 3.15 for an

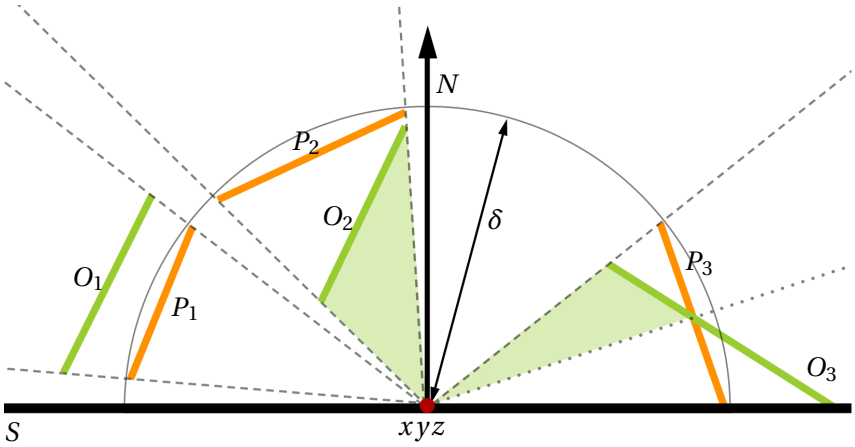


Figure 3.15: Restricting the visibility information to the upper hemisphere $hemisphere(xyz,\delta)$. The occluders O_1, O_2 and O_3 are entirely visible from xyz. P_1, P_2 and P_3 are fragments of the $hemisphere(xyz,\delta)$, which correspond to coherent sets of view rays intersecting the occluders. The three fragments subtend the same solid angles as the three occluders. Thus they produce the same ambient occlusion. δ is the maximum occlusion distance, beyond which the visible geometry no longer influences the ambient occlusion computation. This extra condition allows to completely reject O_1 and to determine the part of O_3 which can be taken into account.

illustration):

- O_i is in front of P_i, with respect to xyz. Thus, the visible fragment of O_i lies completely within the $hemisphere(xyz,\delta)$ and contributes to the ambient occlusion for xyz. Equation 3.12 can be applied to P_i.
- O_i is behind P_i, with respect to xyz. Thus, the visible fragment of O_i lies completely outside the $hemisphere(xyz,\delta)$ and has no contribution to the ambient occlusion for xyz.
- O_i intersects P_i. Thus, the visible fragment of O_i intersects the $hemisphere(xyz,\delta)$ and it partially contributes to the ambient occlusion for xyz. In this case, P_i is clipped against O_i and Equation 3.12 is applied to the resulting fragment.

The above discussion was made under the assumption that locating O_i with respect to $hemisphere(xyz,\delta)$ is equivalent to calculating the position of O_i with respect to P. This point is further discussed in Section 3.4.3.

The next section focuses on the practical aspects of our method.

3.4 Implementation

The from-polygon visibility algorithm represents an evolution of our from-polygon occlusion method. Our new algorithm calculates and represents a more complex information than the previous occlusion version. The main guidelines and some of the operations performed by our two methods are similar. However, since we compute a different information, in a different context, the algorithm needs to evolve. In this section, we build upon the from-polygon occlusion implementation to describe our new visibility algorithm. We highlight the main differences and point to various fragments of the previous chapter for the operations which are analogous.

3.4.1 Overview

Our new algorithm retains the lazy implementation of the from-polygon occlusion calculation: the data structure is built on-demand at run time, as directed by the visibility queries. Thus, we need the notion of temporarily *undefined* classes, in order to designate those leaves which, at some point during execution, may not yet represent a *free* / *hit* class.

We work with these types of classes:

- A *hit* class: Any *visibility* class representing a set of rays stabbing the same first occluder.
- A *free* class: Any *visibility* class representing a set of rays which miss all the occluders.
- An *undefined* class: Any *visibility* class that has not yet been needed by a visibility query. Thus, it is neither marked as *free* nor as *hit*.

Following the same approach as in the description of our occlusion algorithm, we first summarize the main operations performed by the method, and then provide further details.

- At first, all the occluders for the surface S are selected. Although the selection step serves the same purposes as for the soft shadows algorithm, the selection process is different.
- For an occluder O, the lines stabbing both S and O are contained in the 5-dimensional minimal polytope $poly(S \rightarrow O)$. This polytope is then calculated for each occluder. This procedure is identical to the one described in Chapter 2, Section 2.3.1 and illustrated by the pseudo-code in Algorithm 1.
- A root node is created and associated with all the occluders. The node structure is different for the visibility algorithm, since we need to take into account a depth information.
- The tree is grown by inserting the occluders, which is equivalent to adding the visibility classes generated by an occluder to the tree. Each newly added subtree replaces a leaf representing a previously undefined class. In the context of our new algorithm, the insertion of the subtree must take into account the visibility information contained in the replaced leaf.

- An *undefined* leaf has one or several occluders associated with it. Before replacing it, its associated occluders need to be depth culled. This represents a new step with respect to our previous implementation.
- After replacing an *undefined* leaf, its occluders are located into the newly added subtree. This is achieved by a conservative insertion process, as described for the soft shadows algorithm, in Chapter 2, Section 2.3.1 and illustrated by Figure 2.15. However, contrary to this previous implementation, the occluders can be located in all the leaves, including the right ones.

Selecting occluders

When calculating the ambient occlusion for a point xyz on S, we need to take into account the geometry located above S, with respect to its upper normal, and within a δ radius. More exactly, we are interested in the polygons which are either intersect or are contained in the upper hemisphere centered at xyz and having a δ radius, $hemisphere(xyz, \delta)$.

Since our algorithm calculates the visibility from the entire surface S, we need to consider all the potential occluders for all the points of S. Therefore in order to select all the necessary geometry for each point on S, we compute the bounding sphere of S and increase its radius by δ. We then select the occluders using a hierarchical sphere culling process over the scene[Hub93]. Any geometry intersecting the sphere is considered as potentially visible. At this point, a second test eliminates all the geometry located below S, with respect to its upper normal.

The result is an overly conservative set, which is further refined by another test, which eliminates the geometry located above a plane parallel to S and situated at a δ distance. Figure 3.16 provides an illustration of the these selection processes.

If the definition of the scene allows it, a back-face culling selection is also applied to the occluders, in order to avoid dealing with useless geometry.

BSP representation of an occluder

The BSP representation of an occluder O, noted $BSP(O)$, has been described in Section 3.3.2 and illustrated by Figure 3.12. The only difference is that the right leaf corresponds to a *hit* class, and that O is associated with this leaf. The left leaves correspond to coherent sets of lines missing the occluder, and thus to *free* classes.

Creating the root node

The root node corresponds to all the view rays originating from S. These rays will be partitioned according to the first occluder in $O(S)$ they intersect.

For each one of these occluders, a Plücker polytope is created, using the procedure described in Algorithm 1. The structure of a node, presented in Algorithm 7, is slightly different from the

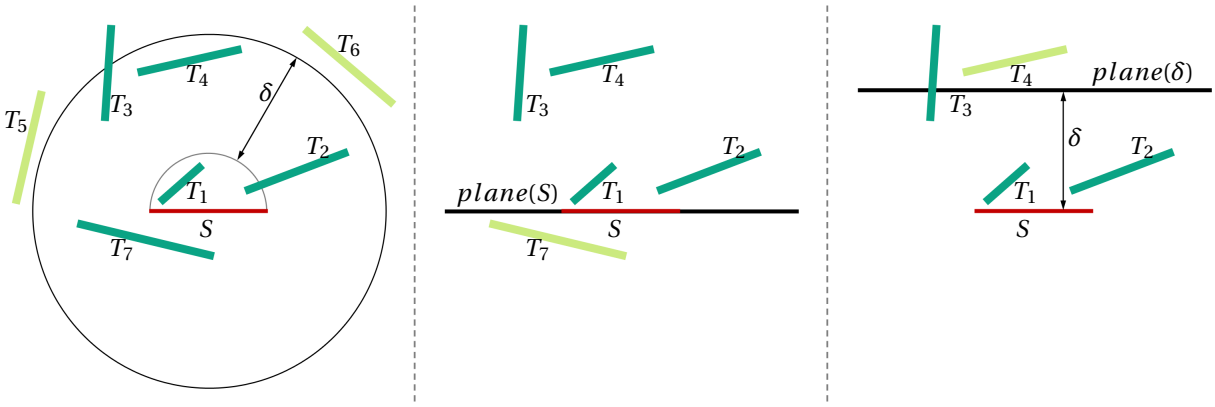


Figure 3.16: The selection of the potential occluders for the polygon S is divided into three steps. **Left**: First of all, we compute the bounding sphere of S and increase its radius by δ. We then select all the geometry which intersects or it is contained into this second sphere. Thus, T_1, T_2, T_3, T_4 and T_7 are selected while T_5 and T_6 are discarded. **Center**: Then, we discard all the occluders located below S, with respect to its upper normal. This eliminates T_7. **Right**: The first step yields an overly conservative set, which can be further simplified by eliminating the occluders located completely above a plane parallel to S and situated at a δ distance ($plane(\delta)$). This third test discards T_4. Note that T_3 is kept, because it intersects the $plane(\delta)$, and thus is partially located within a δ radius from S.

one presented in the previous chapter (see Algorithm 2), in order to handle the additional depth information. More exactly, each inner node contains one hyperplane, whereas each leaf may contain two types of information:

- The occluder intersected by all the rays located in the leaf, stored in $node.hit$ (Algorithm 7, line 4).
- A list of potential occluders, stored in $node.occluders$ (Algorithm 7, line 5).

The initialization of the root node is similar to the one described for the from-polygon occlusion algorithm (see Algorithm 3). We build one polytope for each occluder $O_i \in O(S)$ and the surface S, and we deal with the degenerate cases by splitting S, as explained in Section 2.3.1 and illustrated in Figure 2.13. We also take into account the case when an occluder O intersects S and clip it with respect to the support plane of S, in order to avoid false visibility.

Growing the data structure: Inserting an occluder

This step is similar to the one described in Chapter 2, Section 2.3.1. The tree is grown by inserting the occluders, which is equivalent to replacing a leaf with the root of the BSP representation of an occluder. In the case of the from-polygon occlusion algorithm, this insertion was not conditioned in any way. However, when calculating visibility, the depth factor must be taken into account before each insertion.

An *undefined* leaf is a leaf which has one or more potential occluders. If the leaf also has an associated hit occluder, then all the coherent set of rays located into the leaf intersect it. More exactly, this occluder is the farthest intersected geometry (maximal depth) by the rays reaching the leaf, and

Algorithm 7 The structure of a node

```
1: Structure Node
2: {
3:     hyperplane : Plücker hyperplane
4:     hit : Polygon
5:     occluders : list of Polytope
6:     leftChild : Node
7:     rightChild : Node
8: }
9:
10: /*
11: If occluders = ∅ then the class is either free or hit
12: If occluders = ∅ and hit = ∅ then the class is free
13: If occluders = ∅ and hit = O_i then the class is hit and O_i is the first intersection for the rays located in
    the leaf
14: If occluders ≠ ∅ then the class is undefined, whatever the value of hit
15: */
```

with respect to S. All the potential occluders in the leaf which are located in front of the hit occluder may be intersected before it. And all those located behind it cannot be intersected by the rays in the leaf, since they are beyond the farthest possible intersection.

Another difference with respect to our from-polygon occlusion algorithm is that each new insertion must preserve the depth information. Thus, the newly added subtree is modified to take into account this information. Figure 3.17 gives an illustration.

Depth Culling

Let O be an occluder and $BSP(O)$ its BSP representation. The right leaf of $BSP(O)$ corresponds to the view rays which intersect O. This leaf also has a list of occluders. As mentioned before, O is the farthest possible intersection for the rays located into the right leaf. Thus, we can discard all the potential occluders which are located behind the support plane of O, with respect to S. Figure 3.18 provides an illustration.

This depth culling is valid for all cases under the assumption that the support plane of O does not intersect S. Since our algorithm deals with this case properly, the depth culling can always be applied accurately.

Locating an occluder

The process of locating an occluder into a sub-tree is similar to the one described in Chapter 1, Section 2.3.1, except for one difference. In the case of from-polygon occlusion, we never locate occluders in the right leaves, because they represent *occluded* lines, so we are no longer interested if other geometry blocks the same coherent set of lines. However, in the case of from-polygon visibility, during the execution of the algorithm, right leaves represent sets of view rays which intersect one or more occluders and which need to be distinguished into sub-sets according to the first intersected

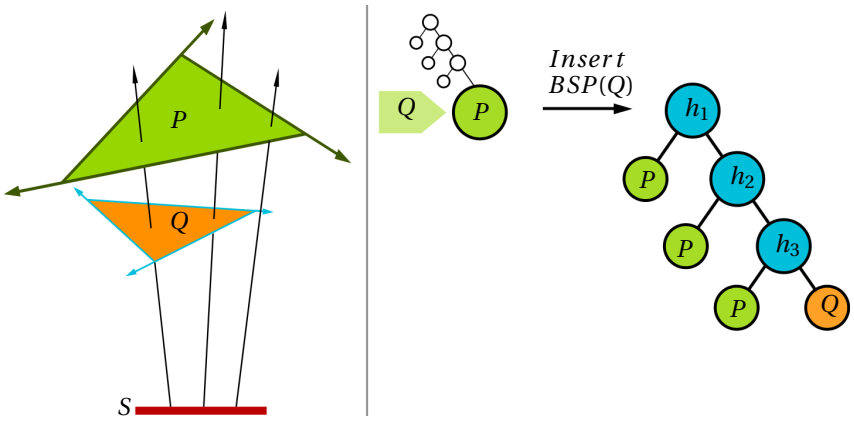


Figure 3.17: When an *undefined* leaf is replaced by the BSP tree corresponding to an occluder, the depth information needs to be inherited in the newly added structure. Let S be the source surface from which we are calculating visibility, and P and Q be two triangles in the configuration given in the left part of the figure. We suppose that the current data structure contains the BSP representation of the classes generated by P (center). Thus, the right leaf contains P (hit occluder) and has Q as potential occluder. More exactly, all the coherent set of rays located into this leaf intersect P. This leaf is replaced by $BSP(Q)$ (right). All the sets of rays which will be located into this new sub-tree intersect P. Moreover, all the sets of rays located in the left leaves of this sub-tree intersect P, and miss Q. Therefore, all the left leaves contain P as hit occluder and the right leaf reports Q as the first intersected geometry. It is important to note that this corresponds to a depth information: all the coherent sets of rays located into the right leaf intersect both Q and P, in this specific order. Thus, Q is the first geometry intersected by the rays located in the right leaf, and P is the first geometry intersected by the rays located in the left leaves.

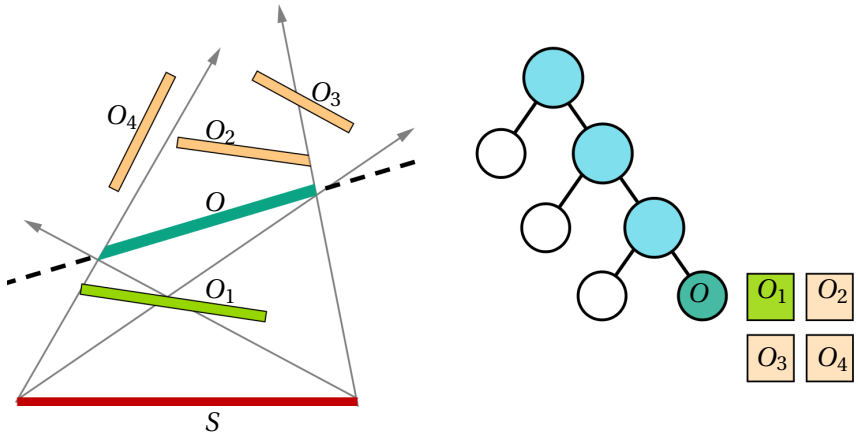


Figure 3.18: Illustration of the depth culling process for the occluders associated with a leaf. **Left**: Five polygons (O, O_1, O_2, O_3, O_4) in various configurations. **Right**: The BSP representation of the occluder O. All the view rays originating from S and which are located in the right leaf of $BSP(O)$ intersect O. The occluders O_1, O_2, O_3, O_4 are also associated with this leaf. Thus, they may or may not be intersected by the view rays reaching the leaf. However, we are only interested in those occluder which are intersected before O. Thus, we can discard the geometry located behind the support plane of O, with respect to S, which in this case is represented by O_2, O_3, O_4. It is important to note that although both O_3 and O_4 are visible from S, they are discarded. This is because we are only taking into account the set of rays located into this particular leaf (the boundaries of the set are represented in this 2-dimensional illustration by the four arrows). And these rays intersect O_3 after intersecting O, and never intersect O_4.

geometry. Therefore, the occluders are located in all the leaves of the tree, and not only into the left leaves, as it was the case of our occlusion algorithm.

The pseudo-code for this procedure is summarized in Algorithm 8.

Algorithm 8 Location of an occluder

```
1:  Procedure locateOccluder (Node n, Polytope O)
2:
3:  if n is a leaf then
4:      n.occluders ← n.occluders ∪ O
5:  else
6:      pos ← orientation(O.VRep, n.hyperplane)
7:      if pos > 0 then
8:          locateOccluder(n.rightChild, O)
9:      else if pos < 0 then
10:         locateOccluder(n.leftChild, O)
11:     else
12:         locateOccluder(n.rightChild, O)
13:         locateOccluder(n.leftChild, O)
14:     end if
15: end if
```

3.4.2 Core Algorithm

The visibility algorithm finds the *visibility* classes describing the geometry which is directly visible from a point xyz on S, through a set of occluders $O(S)$. It starts with a list of polygons representing the upper hemisphere centered at xyz and having a δ radius, $hemisphere(xyz,\delta)$, and subdivides them into several convex fragments. Each fragment represents a coherent set of rays belonging to a single *visibility* class. More exactly, each fragment corresponds to an analytical description of a continuous set of rays which intersect the same first triangle.

A visibility query can be summarized as follows: Given a point xyz on a surface S, and a polygon P from the polygonal representation of the $hemisphere(xyz,\delta)$, we want to find the visible triangles from xyz in direction of P. We also want the exact fragments of P corresponding to each coherent set of view directions. Algorithm 9 details such a query and Figures 3.19, 3.20 and 3.21 apply it to a given configuration.

As previously mentioned, the algorithm is lazy, meaning that it combines the visibility query (lines 6 – 28) and the growing of the data structure (lines 31 – 38) in a single step.

In the beginning, all the occluders are associated with a single *undefined* leaf, the root node. For each inner node, the algorithm tests the orientation of the rays stabbing P with respect to the hyperplane stored in the node, using the procedure described in Chapter 2, Section 2.2.3 and illustrated in Figure 2.11 (lines 9 – 14). The *makePlane* procedure calculates the plane defined by xyz and the hyperplane (line 9). If P lies in the positive (resp. negative) half-space, then all the lines

Algorithm 9 Core algorithm: Given a point xyz on a surface S and a polygon P, the visibility query finds the exact fragments of P corresponding to coherent set of rays intersecting the same first triangle. P is used to drive the data structure construction and may be split into several fragments representing coherent sets of rays from xyz. The fragments reaching visible triangle classes correspond to geometry elements which are direcly visible from S."

```
1:  Function mainQuery (Node n, Polygon P, Point xyz)
2:  Return Polygons
3:
4:  loop
5:     ____________________________________________________
6:     // Querying the data structure
7:
8:     while n is not a leaf do
9:        Plane pl ← makePlane(xyz, n.hyperplane)
10:       pos ← position(pl, P)
11:       if pos > 0 then
12:          n ← n.rightChild
13:       else if pos < 0 then
14:          n ← n.leftChild
15:       else
16:          // Split the polygon (P) and work recursively
17:          return mainQuery(n.rightChild, P ∩ pl+, xyz) ∪ mainQuery(n.leftChild, P ∩ pl−, xyz)
18:       end if
19:    end while
20:
21:    // A leaf has been reached
22:
23:    if n.occluders ≠ ∅ and n.hit ≠ ∅ then
24:       depthCulling(n.occluders, n.hit)
25:    end if
26:    if n.occluders = ∅ then
27:       return visibleFragment(n.hit, P)
28:    end if
29:
30:    ____________________________________________________
31:    // Building the data structure
32:
33:    RO ← random occluder from n.occluders
34:    n ← root of BSP(RO)
35:    for each O in n.occluders, O ≠ RO do
36:       locateOccluder(n, O)
37:    end for
38: end loop
```

stabbing it have a positive (resp. negative) orientation, and the algorithm continues in the right (resp. left) subtree (lines 12 – 14). Otherwise, P is split against the plane and the algorithm continues recursively in both subtrees with the relevant parts of P (lines 16 – 17).

When a fragment reaches a leaf, several alternatives can occur:

- The leaf corresponds to a *free* class ($n.occluders = \emptyset$ and $n.hit = \emptyset$). Thus, the fragment

represents a set of coherent rays missing all the triangles and is discarded.

- The leaf corresponds to a *hit* class ($n.occluders = \varnothing$ and $n.hit = O_i$). Thus, the fragment represents a coherent set of rays which intersect the hit occluder associated with the leaf. Before returning or discarding the fragment, its position with respect to the hit occluder needs to be determined according to the procedure explained in Section 3.3.3 (see Figure 3.15) and represented by the *visibleFragment* procedure. It is important to note that this query is answered without developing the tree and thus taking advantage of previous queries.

- The leaf corresponds to an *undefined* class ($n.occluders \neq \varnothing$). Thus, the fragment represents a set of rays which may intersect the potentially visible geometry associated with the leaf. If the leaf also contains a hit occluder (line 23), the set of rays intersect it. Therefore, the associated geometry can be culled with respect to the support plane of this occluder (line 24, *depthCulling* procedure). This eliminates all geometry located behind the hit occluder, with respect to S and independently of xyz. If some geometry remains, we cannot answer the query without further developing the tree.

The development of the data structure is similar to the one described for our from-polygon algorithm: a random occluder is chosen among the occluders of the current leaf (line 33) and its BSP representation is inserted into the tree (line 34), by replacing the *undefined* leaf. Then, the remaining occluders are located in the new sub-tree. Two differences exist with respect to the occlusion algorithm: the inheritance of the depth information and the fact that occluders can be located in both the left and the right leaves.

Note that the depth culling is necessary only in the right leaves. Applying it in the left ones also is redundant, since they contain the depth information inherited from the replaced leaf. Although Algorithm 3.21 does not take into account this optimization, our actual implementation does.

3.4.3 Key Points

Hemisphere representation. The algorithm extracts the exact visibility from a point. However, the occlusions are only considered in a limited environment. Thus, if a visible triangle intersects the upper hemisphere, only a fragment must be taken into account. This fragment is obtained by clipping the corresponding patch against the triangle. This is the only operation dependent on the chosen tessellation. Clipping a polygon against a sphere yields a non polygonal object. Therefore, an approximation needs to be done, in order to use Equation 3.12. In order to minimize its error one might choose a fine tessellation. However, in practice, various choices produce the same visual effect. The explanation comes from the fact that each hemisphere patch is subdivided several times. This becomes equivalent to having a sufficiently fine initial subdivision.

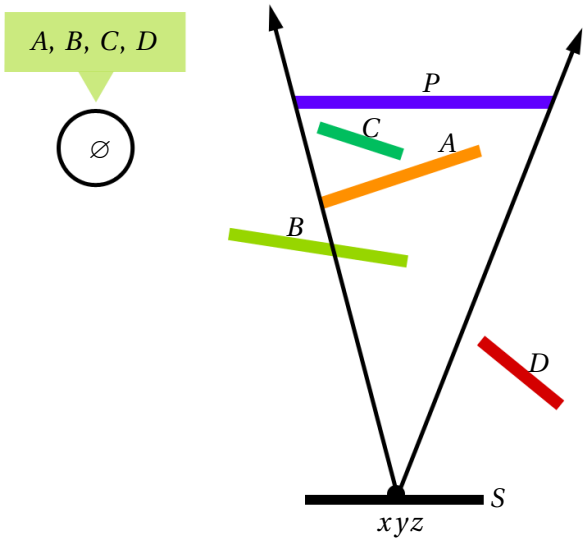


Figure 3.19: Illustration of Algorithm 9 for calculating the visibility from a point $xyz \in S$, using a fragment P from the polygonal representation of $hemisphere(xyz, \delta)$ and the set of occluders $O(S) = \{A, B, C, D\}$. **Left:** The algorithm starts with an *undefined* leaf node, associated with the entire set of occluders. **Right:** The initial view beam contains all the rays originating from xyz and stabbing P.

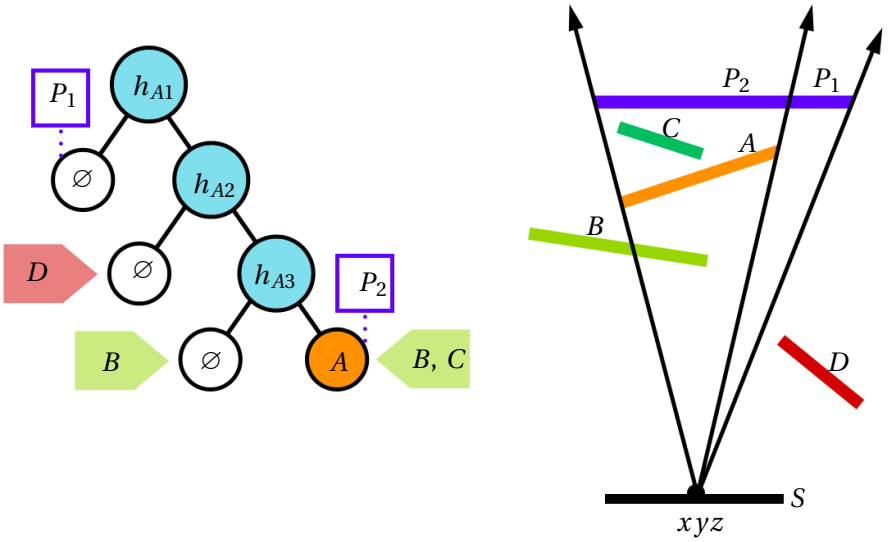


Figure 3.20: **Left**: One occluder is chosen randomly (A in this case) and the root leaf is replaced by $BSP(A)$. The inner nodes contain the hyperplanes corresponding to A's edges (h_{A1}, h_{A2}, h_{A3}). The left leaves represent sets of rays missing the triangle, while the right leaf represents all the rays stabbing A. A is considered to be the farthest possible intersection (maximal depth) for these rays. The remaining geometry (B, C and D) is then located into the tree, using the technique described by the procedure *locateTriangle* (Algorithm 9). Since $poly(S \rightarrow B)$ is found to intersect h_{A3}, the triangle is sent in both left and right leaves. **Right**: Next, the continuous set of rays stabbing P are located into the tree. An intersection occurs in the first inner node ($P \rightarrow P_1, P_2$), identifying thus the rays missing the triangle (P_1). These are rays which have an infinite maximal depth and do not intersect any geometry. Thus, the fragment P_1 can be discarded. On the other hand, P_2 is located in the right leaf. This leaf has an associated triangle (A) and some potentially visible geometry (B, C). Therefore, the rays stabbing P_2 intersect A (maximal depth) and may also intersect B and/or C. The positions of B and C are tested against the support plane of A. Since C is located behind A (with respect to S), it can be discarded. Thus, B will be used to further develop the tree.

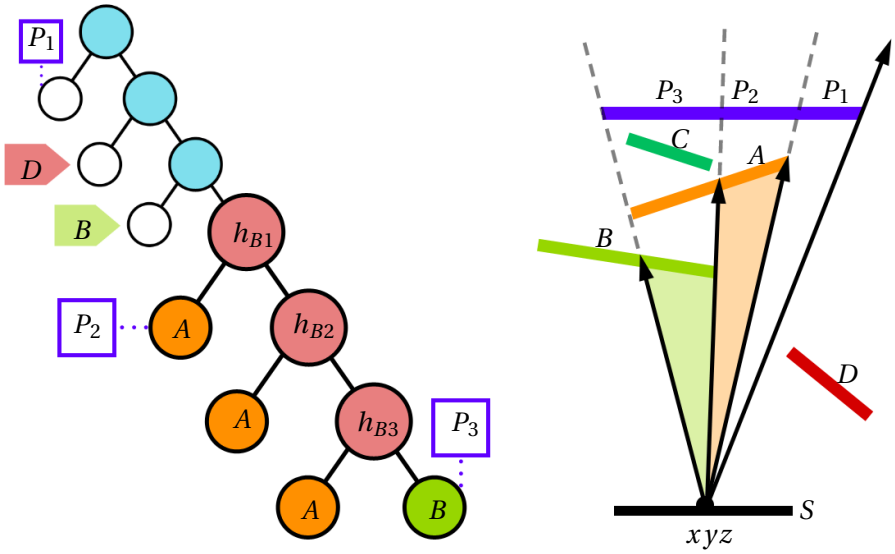


Figure 3.21: **Left:** $BSP(B)$ replaces the right leaf. Note that the intersection information is inherited in the newly added tree. The new left leaves correspond to classes of rays intersecting A (maximal depth), but missing B. The new right leaf represents the class of rays intersecting both B and A, in this particular order. **Right:** P_2 is analytically split into P_2 and P_3, which are located according to the triangles they intersect. Thus, in the end, P_1 represents a continuous set of rays which miss all the occluders, P_2 represents a continuous set of rays which intersect A (first intersection) and P_3 represents a continuous set of rays which intersect B (first intersection).

Random selection of geometry. This point is similar to the one described in Chapter 2, Section 2.3.3. Once again, the algorithm's efficiency is related to the balance of the tree, and some choices may lead to a more balanced tree than others. However this is not predictable. Moreover, in the case of our new from-polygon visibility algorithm, an optimal choice may not even exist. As noted in Section 3.3.1, each visibility class contains both a directional and a depth information. Inserting a triangle into the BSP tree is equivalent to determine its position with respect to the hyperplanes in the node, which represent the support lines of the previously inserted triangles. Moreover, the same triangles are also used to depth sort the potentially visible geometry. Choosing the best triangle for the depth sort does not ensure an optimal choice for the directional sort and vice-versa.

3.5 Falloff Function

When calculating obscurances instead of ambient occlusion, the visibility function in Equation 3.5 is replaced with a falloff function. The latter attenuates the contribution of a triangle depending on its distance with respect to the point for which the computations are being made. However, this replacement results in a new integral for which a closed form solution may not even exist. Therefore, in our implementation, we chose a falloff function which weights the ambient occlusion value for a visible polygon using an average distance between the point and the visible polygon (see Equation 3.14). This distance is calculated using the barycenter of the visible geometry (see Equation 3.15). If the area of this triangle is too important, it can be divided, in order to have a better approximation.

$$AO'(xyz,T) = F(xyz,T)\,AO(xyz,T) \tag{3.14}$$

Where

$$F(xyz,T) = \sqrt{\frac{dist(xyz, barycenter(T))}{\delta}} \tag{3.15}$$

Our tests showed that this solution provides high quality results at a negligible cost.

Attempting to find an analytic solution to the obscurances integral

We would like to mention that we studied the obscurances integral in order to find an analytic solution. We considered a fixed falloff function and started our analysis by considering the 2-dimensional case, when the visible polygon is reduced to a single line. Although we obtained a primitive, its form was too complex compared to what we need, both in terms of computational time and robustness. Thus, it was clear that the 3-dimensional case would either results in a even more complex expression, or simply not exist.

Since our result was far from the analytic form we had hoped for, we decided to use the falloff function previously described. Although we did not demonstrate that an analytic solution cannot exist, attempting to find it proved to be a very complex task. Our analysis is detailed in the Annex of the this work.

3.6 Ambient Occlusion Framework

In order to illustrate the efficiency and the reliability of our exact visibility algorithm, we plug it into the ray-tracing rendering software described in Chapter 2, Section 2.4. Algorithm 10 provides a pseudo-code of our ambient occlusion framework, and its main operations are described below.

Algorithm 10 The following pseudocode illustrates how our visibility algorithm is plugged in a ray tracer software to analytically compute ambient occlusion.

```
1: build visible_triangles, the triangle list visible from the camera
2: // the following loop parallelization is straightforward
3: while visible_triangles is not empty do
4:    remove a triangle S from the visible_triangles list
5:    select the occluders O(S) of S
6:    initialize a BSP tree root node n associated with O(S)
7:    for each image point xyz on S do
8:       build the hemisphere_polygonal_representation, list of polygons
9:       for each polygon P ∈ hemisphere_polygonal_representation do
10:         visible_fragments ← mainQuery(n, P, xyz)
11:      end for
12:      compute the ambient occlusion of xyz by applying Equation 3.12 or 3.14 on
         visible_fragments
13:   end for
14: end while
```

The main idea was to use the same framework, in order to retain the same advantages from which benefits the implementation of our from-polygon occlusion algorithm (see Chapter 2, Section 2.4).

Local complexity. For each triangle visible from the camera, we are only taking into account its local environment. The data structures are built successively and independently per triangle. This treatment ensures that the global size of the scene has little impact on the computations. The only important factor is the local complexity specific to each visible triangle. In addition, computing the visibility per triangle limits the memory consumption since each data structure is deleted as soon as all the related image points have been shaded. This is true for the from-polygon occlusion algorithm also. However, it becomes even more important in the context of the from-polygon visibility method, because we are computing a much more complex information, and thus we need to pay special attention to the memory footprint.

3.7 Results

All tests were run on the 4 cores of a 2.67 GHz Intel Core i7 920 processor with 6GB of memory. The images were rendered at 1280×960 pixels. We compared the performance of our method against the Mental Ray® rendering software (abbreviated MR through the rest of this chapter). Our choice was motivated by the fact that MR is a well known, high quality ray tracing production application. Although MR is not the fastest solution available it represents a rendering standard which is widely used. Comparing our approach directly to previous works based on Pellegrini's approach is not conceivable. They compute occlusion instead of visibility, except in [Bit02]. For all the previous methods, robustness issues restrict their application to environments of moderate size and complexity. In contrast, comparing to MR, we want to show that even if our algorithm may not be as fast as other implementations, our approach is robust, scalable and competitive with respect to a standard production solution. By doing so, we answer one of our initial goals, which was to demonstrate that a solution based on the Plücker parametrization can be as robust and competitive as a production algorithm.

Since we aim to achieve high quality ambient occlusion through analytical computations, we also include a visual comparison with the Ambient Occlusion Volumes [McG10] technique (abbreviated AOV through the rest of this chapter). To our knowledge, AOV is currently the most accomplished analytic technique. Our intention is to show that our method does not exhibit any visual artifact contrary to AOV. The AOV algorithm was executed on a GeForce GTX 285 GPU with 1GB of memory.

Four scenes were chosen to illustrate the behavior of our method. The first one, *House*, is a moderate architectural model combining large and simple areas with some detailed features. *Apples* and *StBasil* are two models with regular meshes, but completely different visual complexities. And finally, *Sibenik* is a complex model, which is often used to illustrate different ambient occlusion techniques.

Ambient occlusion is a local property, usually applied in the final composition as a complement to a direct illumination model. Thus, we chose a δ value sufficient for obtaining visually pleasant results on which all the details are visible. We also include an analysis of the impact of δ's variation, illustrated using our most complex scene (*Sibenik*).

3.7.1 *Mental Ray*® Comparison on Quality and Time

Quality. MR has been configured to produce images which are visually comparable to the results obtained using our method. We sampled the entire hemisphere, using 1024 samples. This value represents the minimum required to remove all the noise in the final images. Figure 3.22 shows the results of both methods.

Time. Since the images obtained using MR are visually comparable to the results achieved by our method, it is pertinent to compare the computation times required for both methods. The

	Our Method						Mental Ray®	
	Memory		Time				Time	Comparison
	Max Size 1th (4th) [MB]		Time / Point [ms]	Init [%]	Queries [%]	Total [s]	MR Total [s]	Acceleration Factor
House	9.02	(36)	0.0488	3.95	96.05	27	221	× 8.18
Apples	3.33	(13)	0.0586	3.54	96.46	26	325	× 12.5
StBasil	4.43	(18)	0.0606	6.78	93.22	17	145	× 8.52
Sibenik	8.87	(36)	0.0476	3.03	96.97	49	547	× 11.16

Table 3.1: Memory and time consumptions for our algorithm, as well as the time obtained using MR. The first column represents the maximum memory load reached during the process. The first value corresponds to the memory load for a single thread. The second value is the maximum obtained by summing the maximum values reported for the 4 threads during an execution. The *Time/Point* column shows the average time spent on calculating the ambient occlusion of an image point, using our data structure. The *Init* column gives the time percentage spent on initializing the trees, while the *Queries* column gives the percentage spent building and querying the trees. The *Total* column gives the time for the whole process. The *MR Total* column indicates the time achieved by MR, while the last column (*Acceleration Factor*) provides the acceleration ratio between our method and MR.

last three columns of Table 3.1 illustrate this comparison. The considered MR execution times correspond to the ambient occlusion calculations only. It can be noticed that our algorithm is faster on all models. Moreover, as shown in Table 3.3, our method remains competitive for larger δ values.

3.7.2 Time Analysis

The total time required by our method can be divided into two steps. The first one concerns the preliminary set-up for each surface: selecting geometry and calculating the necessary hyperplanes, computing a polygonal representation of the upper-hemisphere, and initializing the data structure. The second step involves the visibility queries which compute the needed ambient occlusion values while developing the tree. These values can be found in Table 3.1 (*Init* and *Queries* columns). We notice that the computation time is dominated by the visibility queries, which represent the core of our method.

Our method calculates ambient occlusion for each image point belonging to a visible triangle. Thus, in order to better understand the behavior of our method, we indicate the total time spent on each image (Table 3.1, *Total* column), as well as the average time spent on each image point for which a data structure has been queried (Table 3.1, *Time (ms) / Point* column). This column shows that although the chosen scenes have different complexities, the variation of the average time spent per image point is moderate. As previously explained, the algorithm handles the surfaces one after another. Therefore, the computations are local to each visible source triangle and limited to its close neighborhood, restricted within a δ radius. As a result, our approach escapes the global complexity of the scene.

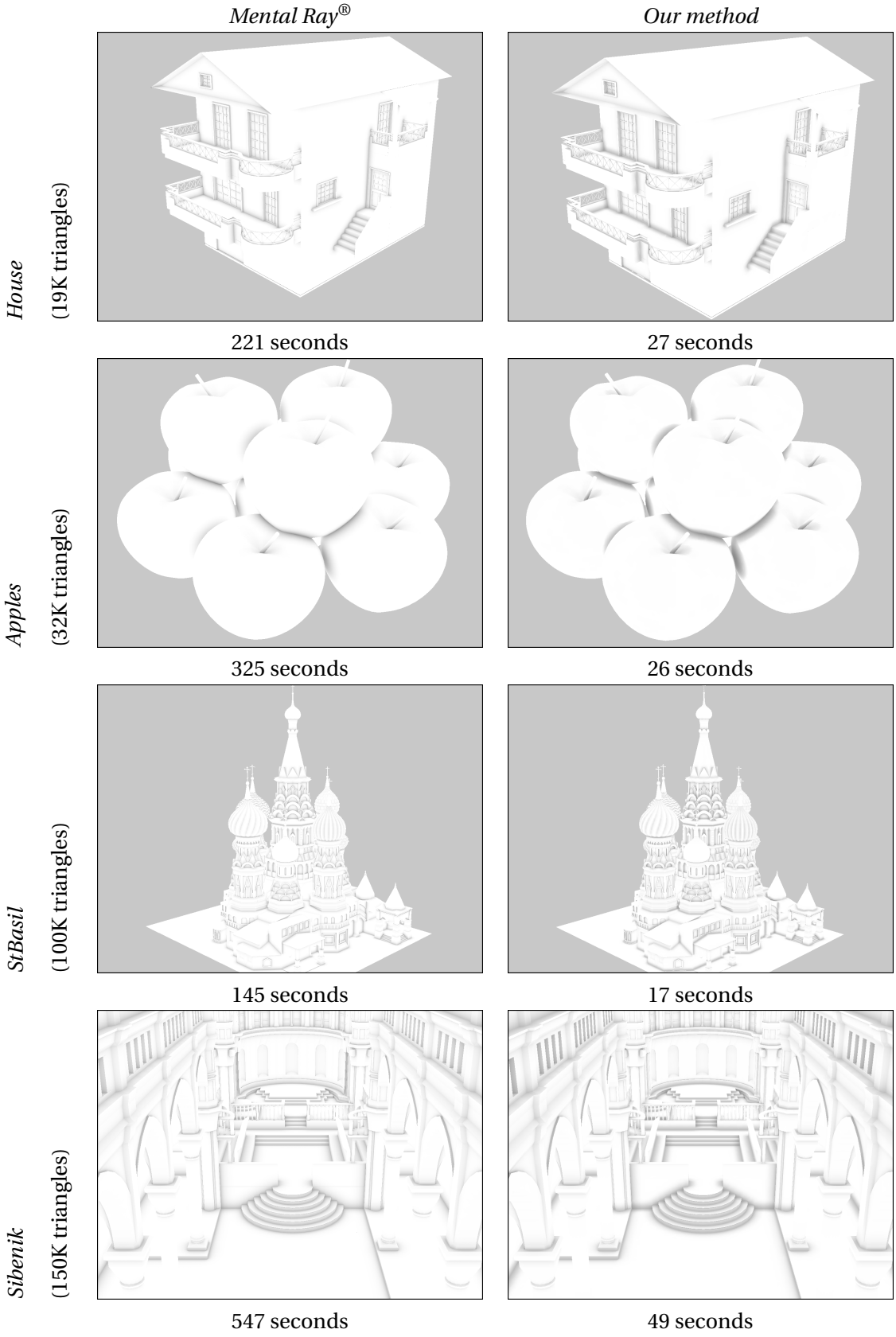


Figure 3.22: The first row presents the results obtained using MR with 1024 occlusion rays per pixel. The second row contains the images achieved using our method. At comparable quality, our method yields better rendering times than MR.

3.7.3 Memory Analysis

Since the algorithm builds one data structure in Plücker space for each visible triangle from the camera, the memory consumption varies during the rendering process. Therefore, we report the maximum memory load that was reached for each scene (see Table 3.1, column *Max Size*), for a single thread. However, since we ran our tests on 4 threads, we are also indicating the maximum memory footprint reported for four simultaneously built data structures.

Figure 3.23 illustrates the memory consumption reported for a single thread. Each vertical bar corresponds to a data structure built for a visible polygon from the camera. The differences between the data structures are due to the different geometric configurations, the number of occluders and the number of queries considered for each visible polygon. The peaks correspond to large BSP trees which encode a complex environment. Although several of these peaks are present, most of the memory footprint remains low.

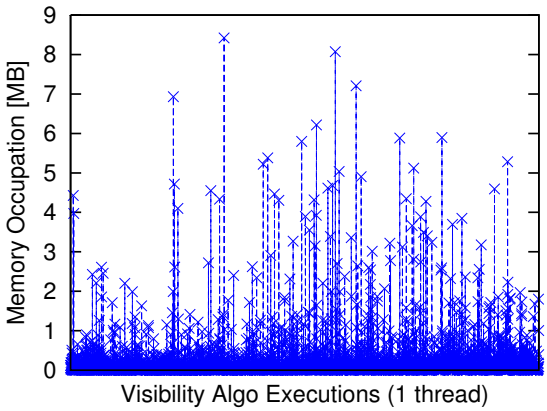


Figure 3.23: Memory consumption for our algorithm, during an execution, measured for a single thread. The scene is *Sibenik*. Each vertical bar corresponds to a data structure built for a visible polygon from the camera. The peaks represent large data structures, corresponding to polygons having a complex environment. However, the majority of the data structures have a low memory footprint.

As shown in Table 3.1 (*Max Size* column), the memory consumption is significantly different between the considered scenes. Both *House* and *Sibenik* have more important memory consumptions than *Apples* and *StBasil*. These last two models have regular meshes, while the first two are irregular. For each visible triangle, our algorithm attempts to take advantage of the visibility coherence between the points on the triangle. Thus, if a large area surface "views" an increased amount of geometry, a loss of the visibility coherence may occur. This is the case with *House* and *Sibenik*.

3.7.4 Visibility Coherence

As for our from-polygon occlusion technique, we tested the ability of our visibility algorithm to take advantage of the visual coherence between the image points. Between each visibility query, the data structure is reset to it root node, associated with the initial set of occluders. In this context, we have defined a visibility query as a call to the *mainQuery* function.

Figure 3.2 provides a comparison between the times required for both versions of the algorithm. Although a direct comparison between our occlusion and visibility algorithms is not straightforward, we can outline an important difference concerning the visual coherence. The acceleration factor is between 16 and 48 for the occlusion method, and between 4.17 and 5.15 for our visibility technique, with an exception of 9 for the *House* model. This decrease can be explained by the fact that we are building a representation of a much more complex information. Let us consider the case of a coherent set of lines belonging to an *occluded* class, in the case of out from-polygon occlusion algorithm. These lines can be blocked by one or several occluders. If we consider the same lines in the context of our visibility algorithm, they are no longer coherent and need to be further grouped according to the first occluders they intersect. Thus, they have a weaker coherence.

In order to explain the exception encountered for the *House* model, we also calculated the average number of queries executed on the data structures, for each scene. This results are presented in the last column of Table 3.2. On the *House* model, the average number of queries is at least three times higher than for all the other models. A low number of queries implies that we are building trees for a small number of points, and thus the majority of queries are used to develop the tree rather than taking advantage of the already calculated information.

	Time		**Comparison**	
	Standard Version [s]	Modified Version [s]	Acceleration Factor	AVG queries / BSP tree
House	27	243	× 9.00	157
Apples	26	134	× 5.15	56
StBasil	17	71	× 4.17	16
Sibenik	49	215	× 4.38	65

Table 3.2: Comparison between the time required for our standard from-polygon visibility algorithm (*Standard Version* column) and a modified version (*Modified Version* column) for which the BSP tree is reset to the initial root node between each query. The *Acceleration Factor* column indicates the performance drop factor between the two executions. The last column indicates the average number of queries for a BSP tree.

3.7.5 The δ Parameter

Time. The computation time according to the δ parameter is illustrated by the first graph of Figure 3.24. When δ increases, our algorithm needs to consider more potentially visible geometry. This leads to an over-cost required for computing a larger data structure. As a result, the algorithm performance decreases compared to MR. This is illustrated by the last column of Table 3.3. The last row reports the crossing point between our method and MR. The image corresponding to this crossing point is presented in Figure 3.25 (right image).

Although our method can handle larger δ values, the interest of performing exact ambient occlusion calculations is diminished when the distance becomes too important. As explained by Laine and Karras [LK10], distant geometry has a very low contribution to the ambient occlusion and can be

	Our Method				Mental Ray®	
	Memory		Time		Time	Comparison
δ	Max Occ / Surface	Max Size 1th (4th) [MB]	Time / Point [ms]	Total [s]	MR Total [s]	Acceleration Factor
2	455	2.43 (10)	0.014	13	459	× 35.31
5	777	5.71 (23)	0.037	38	531	× 13.97
9	1270	21.7 (87)	0.092	101	592	× 5.86
13	2136	36.85 (147)	0.146	169	643	× 3.80
17	2738	51.66 (206)	0.288	342	683	× 2.00
21	3246	64.42 (258)	0.438	527	727	× 1.38
25	3836	80.87 (323)	0.639	776	744	× 0.96

Table 3.3: The variation of the parameters in Table 3.1 with respect to the δ value. The *Max Occ / Surface* represents the maximum number of potentially visible polygons for a surface. All the other columns retain their previous definitions provided in Figure 3.1. Although both memory and time consumption increase, our method remains competitive for all the considered δ values.

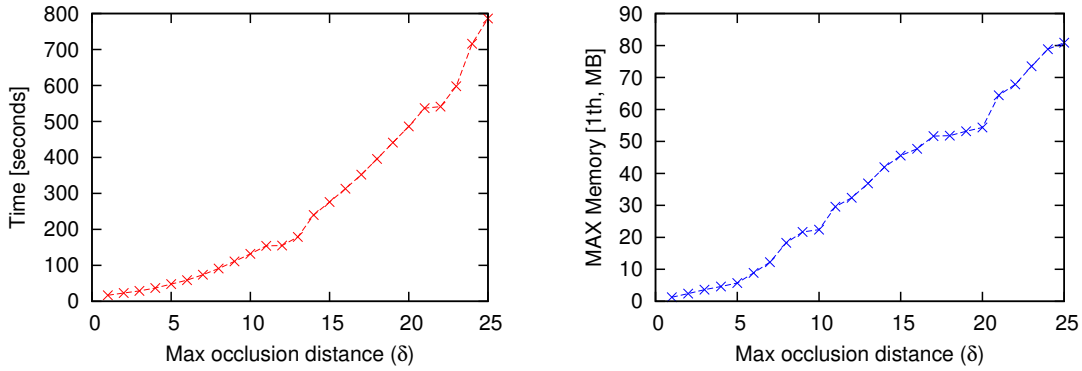


Figure 3.24: Increasing the maximum occlusion distance (δ). The two graphs illustrate the time consumption (left) and the maximum memory load (right) with respect to the variation of δ. The data corresponds to the values from Table 3.3, *Time Total* and *Max Size*. Both graphs show that our method remains practicable, despite the increase with respect to the δ radius.

calculated using simplified geometry, without introducing artifacts or perturb the quality of the result. This relies on the decomposition of illumination into *far-* and *near-field* illumination, and it has been formalized by Arikan *et al.* [AFO05].

Therefore, we believe that an exact and analytical method should not be applied on the raw data in the far-field. In Section 3.8 we analyze how the far field occlusion can be taken into account as a possible development of our algorithm.

Memory. As previously stated, an increase in δ is equivalent to an increase in the potentially visible geometry. The second graph in Figure 3.24 illustrates the memory footprint with respect to δ variation. Although the memory consumption increases with the distance, it is far from being a limitation even for the largest radius.

According to Pellegrini [Pel91, Pel04], the complexity of an arrangement of hyperplanes in Plücker

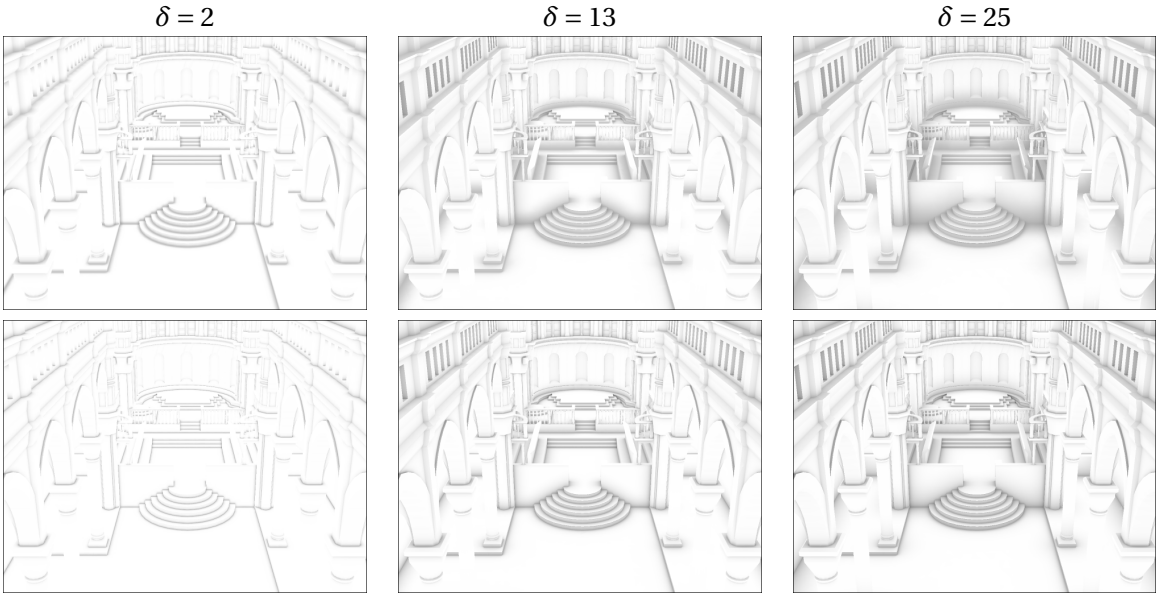


Figure 3.25: A visual comparison between three images corresponding to different δ values (2, 13, 25). The first row presents the results obtained using our base version (ambient occlusion), and the secons row presents the results obtained using our enhanced version (ambient occlusion using our falloff function).

space is $O(n^{4+\epsilon})$ in memory, where n is the number of triangles. Such a complexity may let one think that no practicable application can be build on this theoretical framework. However, our results prove the contrary. A least square fitting analysis using our experiments indicates a practical memory complexity of our application of $O(n^{1.82})$. Note that this practical complexity concerns the ambient occlusion application, where the visibility computations are restricted within a δ radius.

Local complexity of the arrangement. The practical complexity achieved by our method can be explained by the local character of the computations involved. While the arrangement from the theoretical framework concerns all the lines in Plücker space, we are limiting our computations to the arrangement of rays originating from a surface. Thus, we are only constructing a local partition of rays, with respect to the considered surface. Moreover, the lazy evaluation of the visibility avoids computing useless parts of this arrangement.

3.7.6 *Ambient Occlusion Volumes* Comparison

We have performed a quality comparison between our method and the *Ambient Occlusion Volumes* [McG10]. Although both AOV and our algorithm are analytic solutions, we need to underline the fact that its context is different from our own. The *Ambient Occlusion Volumes* algorithm targets near-field ambient occlusion and makes approximations in the visibility calculations is order to achieve interactive frame rates. In contrast, we target off-line high quality rendering, and we aim to provide a robust solution based on accurate visibility computations.

For our tests, we used the *House* model, mainly because this model is used in the original AOV paper [McG10] to highlight the artifacts of the results. It is also used for comparison purpose in other

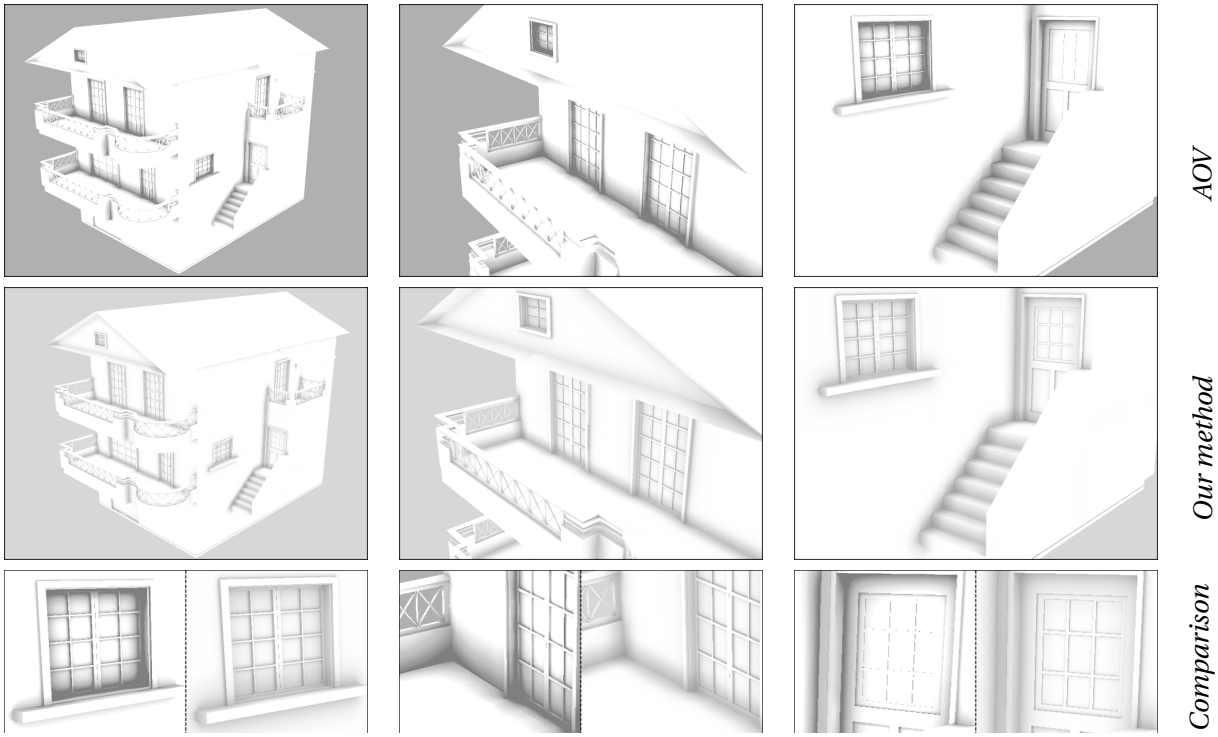


Figure 3.26: The first row presents the results obtained using the Ambient Occlusion Volumes[McG10] technique. The main artifact is over-occlusion which results in a loss of details, especially in the corners of windows or doors. The second row contains the images achieved using our method. The color variations from gray to white are smooth and all the details are present. The third row presents a comparison, which further underlines the visual imperfections of AOV and how they are avoided in our results.

papers (Laine and Karras [LK10]).

We have also tested the performance of both techniques with respect to the δ parameter, using the *Sibenik* model.

For these comparison tests we have used the AOV application available for academic use on the author's web page [AOV12]. Note that this is the same application used for the tests and the results presented in the *Ambient Occlusion Volumes* [McG10] article.

Figure 3.26 provides a visual comparison between AOV and our method. Although both methods produce noise free results, the images obtained using AOV suffer from artifacts resulting from the approximated visibility. The main problem is over-occlusion, which has a negative impact especially in the areas where details are present, such as the windows or the doors. Since our method computes the exact visible fragments of the surrounding geometry, all the details are rendered correctly.

As an indication, the main image took 221 seconds using MR, 27 seconds using our method and 0.074 seconds using AOV. As underlined before, AOV and our own algorithm target different applications and have different objectives. AOV is certainly faster than our technique. However, from a quality point of view, our method provides results that cannot be matched by AOV. But this quality

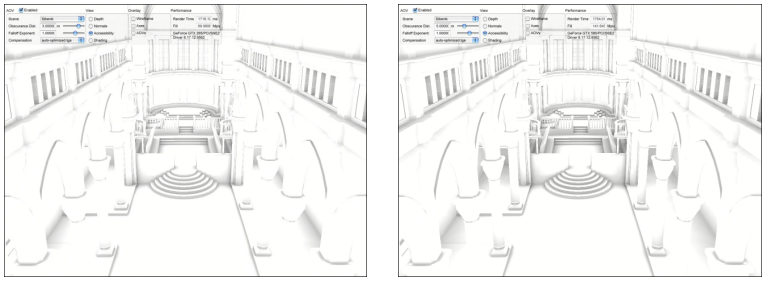
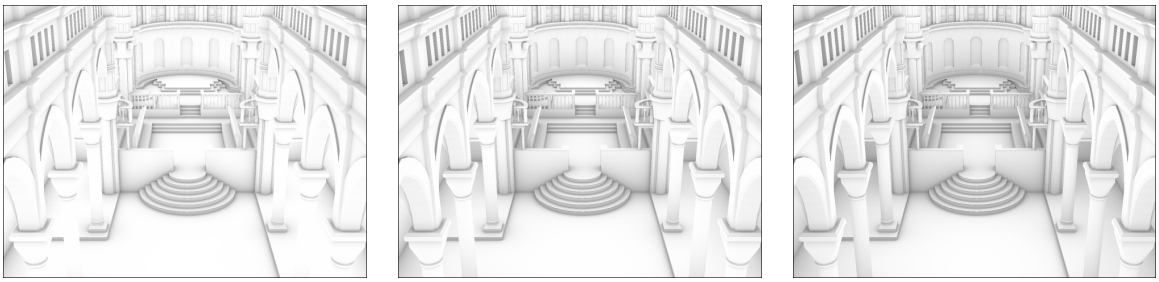


Figure 3.27: Maximum supported δ radius for the *Ambient Occlusion Volumes* (McGuire) technique, on the computer used for testing our algorithm (left image) and on a second computer equipped with a more powerful GPU, and disposing of more dedicated graphics memory (right image). Using the same unit measure as for our own tests, the δ value present in the right image corresponds to approximately $\delta = 10$.



$\delta = 25$
$max\ occluders$: 3836

$\delta = 50$
$max\ occluders$: 11772

$\delta = 70$
$max\ occluders$: 19886

Figure 3.28: Applying our algorithm for the following δ values: 25 (crossing point with Mental Ray), 50, 70.

is not free of charge. Contrary to other methods that may sacrifice quality in order to gain speed, our approach favors quality over speed.

The second series of tests were designed discover the limits of both methods, with respect to the δ parameter. For the AOV renderings, we used two test computers: a Core i7 920 processor equipped with a GeForce GTX 285 GPU with 1GB of memory (also used for the first series of tests) and a Core i7 2600 processor with a GeForce GTX 580 with 1.5GB of memory. Both computers have 6GB of RAM. The maximum δ radius supported by the AOV implementation is visually equivalent (AOV artifacts aside) to our $\delta = 10$ value. For larger values, the algorithm reaches the limits of the graphical card, and the application crashes because it runs out of memory. The results obtained are presented in Figure 3.27.

In contrast, using our method, we present results until $\delta = 25$ which corresponds to the crossing point with MR. However, in terms of robustness, our method can handle larger values. Figure 3.28 presents three images corresponding to the following δ values: 25 (the crossing point with MR), 50 and 70 (as a visual indication: a sphere having a 70 radius and which is placed in the center of Sibenik will contain all its interior, from floor to ceiling). The maximum number of occluders is 3836, 11772 and 19886, respectively. It is important to underline that the visual difference between the last two images is almost imperceptible to the naked eye.

The AOV technique is analytic and fast, but comes with artifacts and focuses on near-field occlusion. Our approach is analytic and based on an exact visibility algorithm. Moreover, it can handle more significant δ values while remaining competitive to MR. Beyond the crossing point, our method keeps the advantage of remaining noise-free, while for a ray-traced solution it may be necessary to increase the number of samples. However, when large δ values need to be considered, we do not think that the far geometry should be handled the same way as the close one. This point is further developed in Section 3.8.

3.7.7 Falloff Function

This section presents the results we obtained using the falloff function detailed in Section 3.5. Figure 3.29 shows a visual comparison between the results obtained using the base (ambient occlusion) version of our method and the enhanced one (including falloff). As indicated by the rendering times, the extra computational cost is negligible. The reason for this is that the majority of the required information is already present in the data structure, and the additional calculations consist of basic operations.

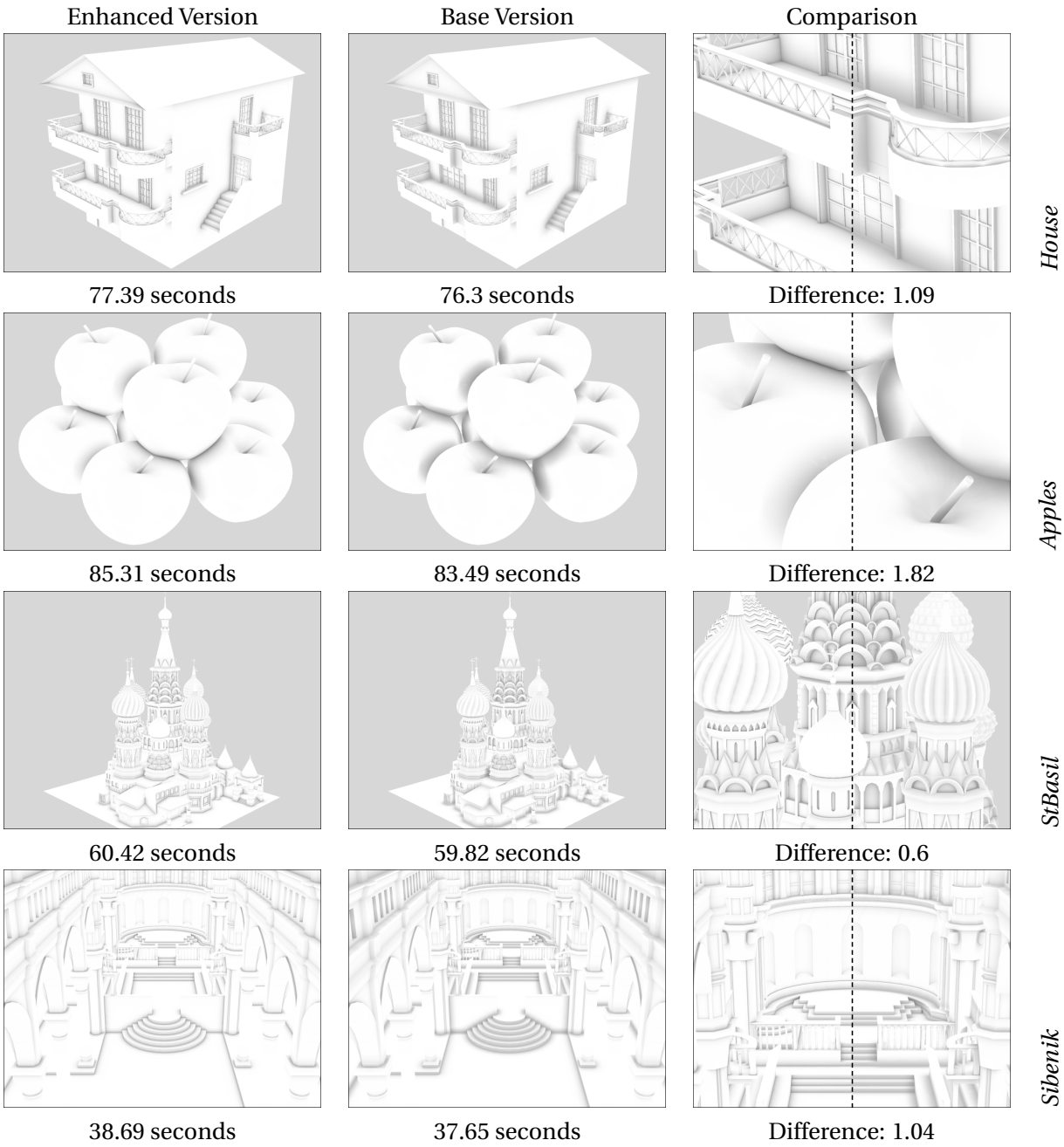


Figure 3.29: Visual comparison between the results obtained using the enhanced version (ambient occlusion with our falloff function) (first column) and the base version of our algorithm (center column). The third column offers a more detailed comparison. The rendering times, as well as the difference between the two versions, are indicated below each image. The extra computational time required to apply the falloff function represents a negligible cost with respect to the total rendering time.

3.8 Discussions and Future Work

This section outlines some of the limitations of our method, as well as some possible solutions which are to be addressed in the future.

If the considered local environment goes beyond a very important δ radius, the algorithm reports an increased memory load. More precisely, a visible polygon for which a large set of geometry needs to be analyzed will result in an expensive data structure, both in terms of memory consumption and execution time. An interesting solution would be to deal with sequential layers of geometry, instead of considering at once all the potential occluders for a surface. For a chosen δ value, we can start with the geometry contained into a δ/k radius. If some queries require further development, the geometry contained between δ/k and $2 \times \delta/k$ is added and so forth (k can be a constant or not). The advantage of such an approach is that if we decide to calculate the ambient occlusion for a radius of $\Delta > \delta$, we can reuse the previous computations.

This leads us to another possible development of our ambient occlusion algorithm. The current algorithm treats all the potentially visible geometry within a δ radius equally. No distinction is being made between the near and the far field. Therefore, it may construct a complex tree in order to add a very small occlusion effect from a far away object. Moreover, as outlined by Laine and Karras [LK10], such distant geometry may be used in a simplified form without altering the visual results. Since only the near geometry needs to be exact and thus processed accurately, we could adapt our current exact method to handle far field occlusion using an approximation of the actual geometry. Therefore, our algorithm would be building complete data structures only for close and relevant geometry, and approximate the occlusion from far away objects. Thus, we could improve the memory footprint and the time consumption without modifying the visual quality of the result.

Our tests showed that the we could further improve the ability of the algorithm to take advantage of the visual coherence between the image points. As detailed in Section 3.7.4, we often build BSP trees for a small number of points, and thus we are spending more time developing the data structures than exploiting the already calculated information. This problem can also arise in the context of our soft shadow algorithm. In the context of our ambient occlusion implementation, the simplest solution would be to increase the image resolution, and thus have more points for each visible surface from the camera. We have tested this solution and observed a slow growth in the acceleration factor between our standard algorithm and its modified version. However, this is in fact a false solution because increasing resolution only delays the problem. Also, new surfaces can become visible at higher resolutions, and raise the same issue.

A solution would be to group the surfaces which compose a partial fragment of an object, and their occluders. The simplest example is that of a lot of buildings. Each side of a block of flats is composed of several small polygons which represent the walls and the windows. Instead of computing the ambient occlusion for each one of these surfaces, we can consider a side as an independent surface and query its environment for occlusions. The auto-occlusions (the details on

the windows or on the walls) can be handled in a second step. This is somehow similar to handling the level of detail, but this is time the simplification concerns the source surfaces instead of the distant geometry.

The last issue we want to address concerns the balance between the visual coherence and the size of the source polygon. Our algorithm is based on the idea of taking advantage of the visual coherence which exists between neighbor points. If the geometric resolution is very high with respect to the image resolution, it can lead to very few image points per triangle and thus, a loss of efficiency. In this case, the data structure is built for a few points only and dropped before being re-used. This problem also concerned our soft shadows algorithm. However, in the case of from-polygon visibility, we have to deal with a second issue. If the source polygon is very large with respect to its occluders, the coherence is further diminished. Thus, in order to answer all the queries, the algorithm needs to build a large data structure, and will eventually spend more time building it than actually reusing the calculated information. Therefore, we are faced with two problems: not enough points on the source polygon to efficiently exploit the encoded information, and much more information that is efficient to encode. The interesting fact is that by trying to solve one issue, we may create another. We believe that a balance must be found between the size of the source polygon and its considered environment. Grouping the occluders according to a certain criterion may provide a potential solution. Grouping the points on the polygon and building one data structure for each group is another idea that needs to be investigated.

***

This chapter has presented our from-polygon visibility algorithm, as well as it application in the context of ambient occlusion calculation generation. With respect to our initial goals, described in the first chapter of this thesis, we have demonstrated that we can successfully encode the visibility from a surface and achieve a robust and efficient solution in a given applicative context. The next and final chapter of this thesis summarizes our contributions and provide further insight on the limitations of our two algorithms.

Conclusion

THIS chapter summarizes the context of this thesis and our contributions with respect to the initial goals. We then provide an analysis of the limitations of our from-polygon occlusion and from-polygon visibility algorithms.

In the first chapter of this work we have distinguished and presented two main types of visibility problems: from-point and from-polygon visibility. Various solutions concerning the first category were proposed over the years. In contrast, from-polygon visibility received limited attention because of its inherent complexity. Few algorithms exist and they either have incomplete implementations, or limited practical applications because of their lack of robustness and numerical issues.

As explained in the first chapter of this work, we dispose of a solid and elegant theoretical framework which can be used to address from-polygon visibility problems. The first element of this framework is the Plücker parametrization, which maps any real line to a point or a hyperplane in a 5-dimensional oriented projective space. Its main advantage is that it allows describing continuous sets of lines using convex volumes, which are more intuitive to manipulate. The second element is represented by the equivalence classes of lines, described by Pellegrini, which allow grouping oriented lines according to the geometry they intersect. This motivated a series of theoretical results which concern various problems involving oriented lines [Pel97].

In practice, however, there is a big gap between the current implementations and the theoretical framework. It is exactly this void that we aimed to fill, by proposing a robust and accurate solution, which can be successfully used in a specific applicative context. More exactly, our objective was to encode the visibility from a polygon into a data structure and use this information to optimize from-point independent queries by taking advantage of the visual coherence which exists between neighbor points. Besides accuracy and robustness, we intended to design an algorithm which does not require any pre-process operations. Moreover, its applications needed to escape the upper-bound complexities announced by Pellegrini.

Summary of Results

Our study of Pellegrini's theoretical framework and of the existing visibility algorithms, allowed us to conclude the following:

- Both the *equivalence* classes described by Pellegrini and the current from-polygon implementations based on the Plücker parametrization [MP99, NBG02, HMN05, MAM05] calculate an information of occlusion. A single exception exists [Bit02], but its practical applications are severely limited. In this context, our main concern was if we could propose a true from-polygon visibility algorithm, which is robust and can successfully handle arbitrary scene configurations.
- The lack of robustness and the numerical issues from which suffer the current from-polygon implementations are caused by the complex 5D CSG operations. Also, these operations add an

additional cost that restricts the application of these methods to pre-process steps only. Thus, one of our main objectives was to avoid all 5D CSG operations, and thus gain both in robustness and performance.

- Concerning from-point visibility, the majority of existing methods do not take advantage of the visual coherence which exists between the points belonging to the same surface. During rendering, all visibility queries are actually from-point queries, since in the end we need to calculate the value of each point visible from the camera. Thus, we wanted to allow these individual queries to take advantage of from-polygon visibility.

From-polygon Occlusion

The starting point of our works are the *equivalence* classes described by Pellegini. His definition concerns the complete arrangement of lines induced by a set of convex polygons. Each cell of this arrangement corresponds to a coherent set of lines which intersect the same polygons, in no particular order.

Our first applicative context was the generation of soft shadows. Thus, having a light source, its occluders and a group of points to shade, we needed to know which parts of the light are visible from the points to shade. The first step was to restrain the definition of *equivalence classes* to the lines stabbing the light, and consider the arrangement induced by the light's occluders. We further simplified Pellegrini's approach, by only distinguishing between *occluded* and *unoccluded* lines. Our occlusion algorithm starts with the complete set of lines stabbing the light source and divides it into coherent sets, each one corresponding to a convex fragment of the light source which is either visible of invisible as seen from the points to shade. This information is encoded in a BSP tree in Plücker space, lazily and at run time, as directed by the individual visibility queries.

Our data structure encodes a from-polygon occlusion information. However, the individual queries which help build it are from-point visibility queries with respect to the light source. By combining from-point queries with a conservative distribution of 5-dimensional occluders into the BSP tree, we manage to avoid all 5D CSG operations, thus achieving the desired robustness and accuracy.

If one or more points share the same occlusion information, only the first query develops the data structure. The other ones benefit from the already existing information, thus improving the global performance. Our tests showed that if we prevent all queries from taking advantage of the previous ones, the loss of efficiency is considerable. Thus, our occlusion algorithm exploits the visual coherence between neighbor points successfully, and optimizes the individual from-point queries using the global from-polygon occlusion information.

Our algorithm is practicable and can handle scenes of various size and configuration. Compared to an optimized ray traced implementation, our method is faster, and its results are exact and noise free.

Therefore, we have successfully proposed an algorithm which encodes the analytic and conservative from-polygon occlusion information. The individual from-point queries direct the construction of the data structure and use it to exploit the visual coherence and take advantage of previous computations. Our algorithm is robust and numerically stable and can handle scenes of different sizes and complexities. The results we achieve are exact, hight quality and noise free.

From-polygon Visibility

Our from-polygon occlusion algorithm demonstrates that we can successfully use the theoretical framework and the Plücker parametrization and provide a robust implementation in the context of a practical application. The next step was to build upon this first method and add the missing depth information, in order to calculate from-polygon visibility instead of simple occlusion.

The applicative context was the generation of ambient occlusion, which requires calculating all the directly visible fragments located withing a given radius from the points to shade. If we consider the entire polygon which contains these points, we need to compute its exact visibility over the surrounding geometry. Similarly to the from-polygon occlusion method, we restricted the computations to all the view rays originating from the polygon. However, in this new context, we needed to group these rays according to the first geometry they intersect. In order to do that we introduced a depth test in the construction of the data structure and an inheritance system allowing us to maintain a coherent visibility information at each development.

The main characteristics of the method remain similar to the from-polygon occlusion algorithm: analytic representation, avoid all 5D CSG operations thanks to a conservative representation and from-point queries, lazy construction directed by the independent queries, robustness and accuracy. Our method is competitive with respect to a production solution such as Mental Ray®, and yields high quality, noise and artifact free results. Also, it achieves better quality results that the best analytic ambient occlusion solution currently available.

A final remark needs to be done before analyzing the limitations of our two algorithms. Although our from-polygon visibility algorithm is exact, its application to ambient occlusion makes a negligible approximation. Since we needed to use only the geometry located within a given radius from the points, we must clip the visible fragments with respect to a polygonal representation of the upper hemisphere. Thus, the data structure encodes the exact from-polygon visibility, but this information is slightly approximated when the ambient occlusion calculation is performed. However, as our tests showed, this approximation is not visible in the ambient occlusion results.

Limitations and Future Work

The second and the third chapters concluded with a summary of the limitation of the soft shadow and ambient occlusion applications, respectively. In this section, we want to outline some limitation of the occlusion and visibility algorithms. Thus, we only consider the from-polygon occlusion and from-polygon visibility implementations, independent of the possible applicative contexts.

In our current implementations, the occlusion/visibility data is dropped as soon as it is not needed any more. Based on the tests we made, we believe that this data could be saved in order to be used again. Our data structure can be seen as a representation of the visibility function from a surface. For a given source polygon and its environment of occluders, the visibility queries develop and use the BSP tree until a certain point is reached. From this point, the encoded information is stable and the queries only exploit it. The first question which arises is how we can define this balance point. When does the data structure become representative for the from-polygon visibility ? The BSP tree stops growing when all the equivalence classes have been found. This data can be saved and reused in order to render the same geometric configuration at different resolutions.

This leads us to another important point. Both our algorithms use a conservative process which locates the 5-dimensional occluding volumes into the leaves they may affect. This may lead to the development of unnecessary and redundant information. An extreme example is the case when all the occluders are conservatively duplicated in both children of a node. In this case, we would be building two identical sub-trees, instead of a single one. Therefore, in addition to saving the BSP trees for further renderings, we could also simplify them to remove the redundancy created by the conservative insertion process. Smaller data structures would require less space and querying a more compact tree would speed up the rendering process.

Both these points (saving and simplifying the data structures) are limited to static environments only. However, we believe it would be interesting to study the possibility of integrating dynamic objects, and what are the exact limitations in such a context. For example, if we have a static environment which contains some moving geometry, how much of the data structure could we reuse in order to take into account the dynamic component ? Analyzing the possibility of a non-static context raises the question of render time and performance.

Since our algorithms are designed to run on several threads, they would benefit from an implementation on a parallel computing architecture such as CUDA or OpenCL, which offer a higher number of threads on the graphic hardware. However, our algorithms are based on a lazy evaluation which builds the data structures during render time. Thus, they rely on dynamic allocations of memory. Although this could be implemented using nVidia CUDA, the dynamic memory allocations from the graphical device are not always possible, or they are made from the global memory, which could lead to a loss of efficiency. Thus, in its current design, the algorithm is better suited for a CPU implementation. Which brings us back to the saving and reusing of the data structures. Indeed, it would be interesting to develop a CUDA implementation which relies on

using the data structures which were already developed. This would eliminate the need for massive dynamic allocations of memory. Moreover, if we also simplify our saved structures, we could store even more data on the graphic device, and increase the number of threads.

***

We would like to conclude by saying that we hope to have reached a milestone in from-polygon visibility. We also hope that this research represents a proof that solutions can be found in order to fill the existing void between the theoretical framework based on the Plücker parametrization and the practical implementations in from-polygon visibility.

Annex

Attempting to find an analytic solution to the obscurances integral

First of all, we choose a fixed falloff function, $\rho(d) = \sqrt{\frac{d}{d_{max}}}$. The main integral becomes:

$$\begin{aligned} Obs(M_0) &= \frac{1}{\pi}\int_{\omega\in\Omega}\rho(d(M_0,\omega))(N\cdot\omega)d\omega \\ &= \frac{1}{\pi}\int_{\omega\in\Omega}\sqrt{\frac{d(M_0,\omega)}{d_{max}}}(N\cdot\omega)d\omega \end{aligned}$$

This can be translated as a sum of integrals over the visible polygons:

$$Obs(M_0) = \frac{1}{\pi}\sum_{i=1}^{n}\int_{p\in P_i}\sqrt{\frac{d(M_0,\omega)}{d_{max}}}(N\cdot\omega)d\omega$$

Where

- n is the total number of exact fragments of polygons which are directly visible from M_0
- $\{P_1,...,P_n\}$ are these fragments

Thus, we needed to find an analytical solution to the following integral:

$$\begin{aligned} Obs(M_0,P_i) &= \frac{1}{\pi}\int_{p\in P_i}\sqrt{\frac{d(M_0,\omega)}{d_{max}}}(N\cdot\omega)d\omega \\ &= \frac{1}{\pi}\int_{p\in P_i}\sqrt{\frac{d(M_0,\omega(\theta))}{d_{max}}}\cos\theta d\theta \end{aligned}$$

where p is a point on P_i, in direction $\omega(\theta)$.

We started our analysis by considering the 2-dimensional case, when the polygon P_i is reduced to a single line segment, $[M_1M_2]$. Figure 3.30 provides an illustration.

In 2D the integral becomes:

$$\begin{aligned} Obs(M_0,[M_1M_2]) &= \frac{1}{\pi}\int_{M(\theta)\in[M_1M_2]}\sqrt{\frac{d(\theta)}{d_{max}}}\cos\theta d\theta \\ &= \frac{1}{\pi}\int_{\theta_1}^{\theta_2}\sqrt{\frac{d(\theta)}{d_{max}}}\cos\theta d\theta \\ &= \frac{1}{\pi}\int_{\theta_1}^{\theta_2}\frac{1}{\sqrt{dmax}}\sqrt{d(\theta)}\cos\theta d\theta \end{aligned}$$

Let $l(\theta)$ be the line passing through $M_0(x_0,y_0)$ and $M(\theta)$, and let $ax+by+c=0$ be the line passing through $M_1(x_1,y_1)$ and $M_2(x_2,y_2)$. These two lines intersect in an unique point: $M(\theta)$.

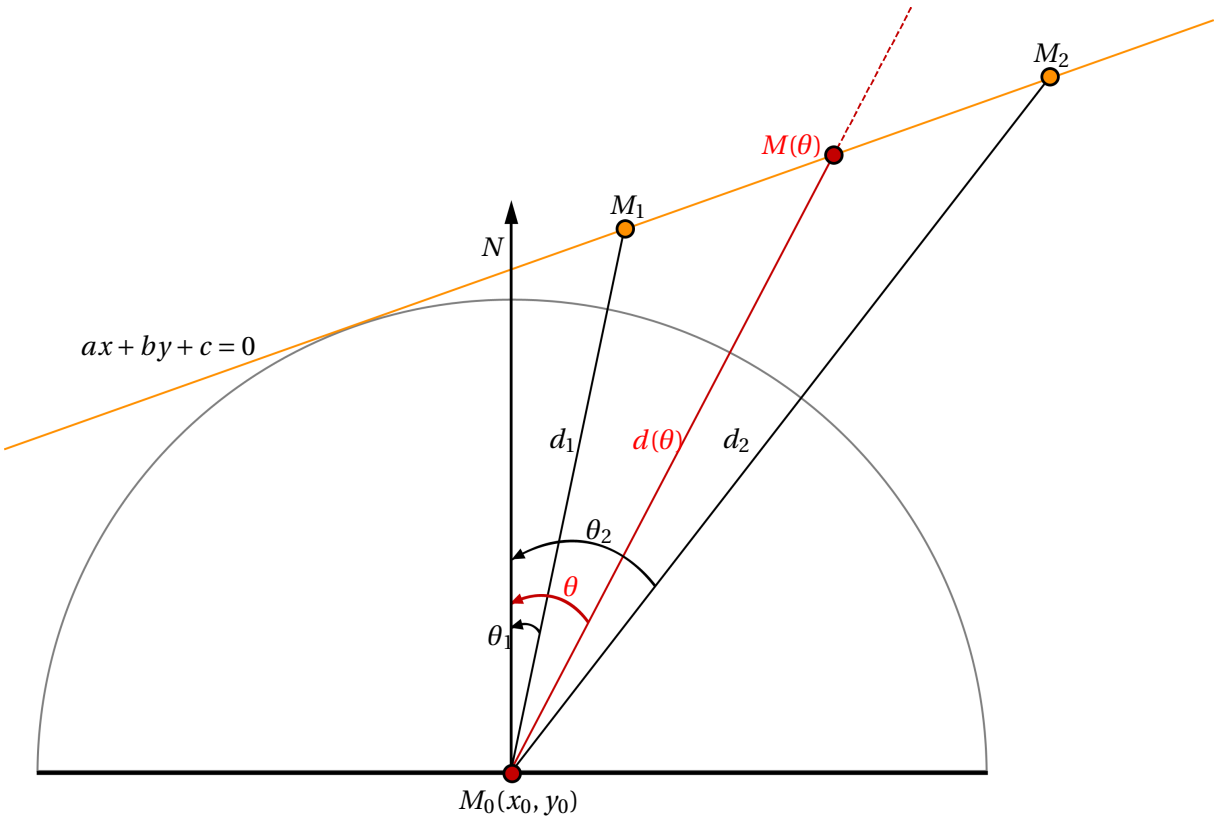


Figure 3.30: 2-dimensional illustration for the obscurances integral. Instead of integrating over a polygon, we integrate over a segment, $[M_1 M_2]$. $M(\theta)$ is the point on the segment in direction $\omega(\theta)$. This is the direction which forms a θ angle with the surface normal.

We can express $M(\theta)$ with respect to the first line, $l(\theta)$:

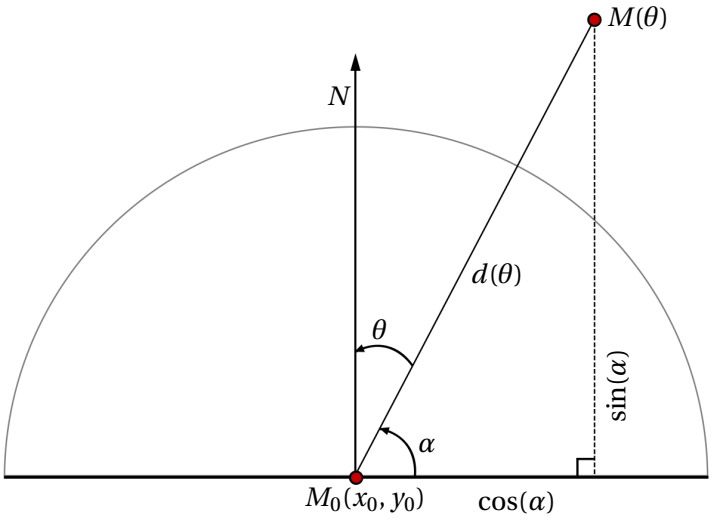
$$\begin{aligned} M(\theta) &= M_0(x_0,y_0) + d(\theta)\begin{pmatrix}\sin\theta\\ \cos\theta\end{pmatrix} \\ &= \begin{pmatrix}x_0 + d(\theta)\sin\theta\\ y_0 + d(\theta)\cos\theta\end{pmatrix} \end{aligned}$$

Figure 3.31 provides an illustration.

Since $l(\theta)$ intersects $ax+by+c=0$ in $M(\theta)$, we obtain the following relation:

$$a(x_0 + d(\theta)\sin\theta) + b(y_0 + d(\theta)\cos\theta) + c = 0 \qquad \Rightarrow$$

$$\Rightarrow \qquad d(\theta) = \frac{-(ax_0 + by_0 + c)}{a\sin\theta + b\cos\theta}$$



$$
\begin{aligned}
M(\theta) = & \\
&= M_0(x_0, y_0) + \begin{pmatrix} d(\theta)\cos\alpha \\ d(\theta)\sin\alpha \end{pmatrix} \\
&= M_0(x_0, y_0) + \begin{pmatrix} d(\theta)\cos(\frac{\pi}{2} - \theta) \\ d(\theta)\sin(\frac{\pi}{2} - \theta) \end{pmatrix} \\
&= M_0(x_0, y_0) + \begin{pmatrix} d(\theta)\sin\theta \\ d(\theta)\cos\theta \end{pmatrix} \\
&= \begin{pmatrix} x_0 + d(\theta)\sin\theta \\ y_0 + d(\theta)\cos\theta \end{pmatrix}
\end{aligned}
$$

Figure 3.31: Calculating the coordinates of $M(\theta)$.

If we replace this in the initial integral, we obtain the following:

$$
\begin{aligned}
Obs(M_0, [M_1 M_2]) &= \frac{1}{\pi}\int_{\theta_1}^{\theta_2} \frac{1}{\sqrt{dmax}} \sqrt{\frac{-(ax_0 + by_0 + c)}{a\sin\theta + b\cos\theta}} \cos\theta d\theta \\
&= \frac{1}{\pi}\int_{\theta_1}^{\theta_2} \sqrt{\frac{-(ax_0 + by_0 + c)}{dmax}} \frac{\cos\theta}{\sqrt{a\sin\theta + b\cos\theta}} d\theta
\end{aligned}
$$

Since the first term is a constant, we would like to find a primitive of the function:

$$
f(\theta) = \frac{\cos\theta}{\sqrt{a\sin\theta + b\cos\theta}}
$$

In order to do this, we have used the $Maple^{TM}$ software for our calculations.

We note I_0 our initial integral:

$$
I_0 = \int \frac{\cos\theta}{\sqrt{a\sin\theta + b\cos\theta}} d\theta
$$

We perform the following change of variable:

$$
\theta = 2\phi + \alpha
$$

and the following substitution:

$$\alpha = arctan(\frac{a}{b})$$

I_0 becomes:

$$I_1 = \frac{2}{b\sqrt{\frac{a^2+b^2}{b^2}}} \int \left(\frac{2b\cos^2\phi - b - 2a\sin\phi\cos\phi}{\sqrt{\frac{(a^2+b^2)(2\cos^2\phi-1)}{b\sqrt{\frac{a^2+b^2}{b^2}}}}} \right) d\phi$$

After further simplification we obtain:

$$I_2 = \frac{2signum(b)}{(a^2+b^2)^{3/4}} \int \left(\frac{2b\cos^2\phi - b - 2a\sin\phi\cos\phi}{\sqrt{(2\cos^2\phi-1)signum(b)}} \right) d\phi$$

At this point we assume $b > 0$. The integral becomes:

$$\begin{aligned}
I_3 &= \frac{2b}{(a^2+b^2)^{3/4}} \int \left(\frac{2b\cos^2\phi - b - 2a\sin\phi\cos\phi}{\sqrt{(2\cos^2\phi-1)b}} \right) d\phi \\
&= \frac{2b}{(a^2+b^2)^{3/4}} \int \left(2b\frac{\cos^2\phi}{\sqrt{(2\cos^2\phi-1)b}} - b\frac{1}{\sqrt{(2\cos^2\phi-1)b}} - 2a\frac{\sin\phi\cos\phi}{\sqrt{(2\cos^2\phi-1)b}} \right) d\phi \\
&= \frac{2b}{(a^2+b^2)^{3/4}} * \\
&* \left(2b\int \frac{\cos^2\phi}{\sqrt{(2\cos^2\phi-1)b}} d\phi - b\int \frac{1}{\sqrt{(2\cos^2\phi-1)b}} d\phi - 2a\int \frac{\sin\phi\cos\phi}{\sqrt{(2\cos^2\phi-1)b}} d\phi \right) \\
&= \frac{2b}{(a^2+b^2)^{3/4}} \left(2b * I_{31} - b * I_{32} - 2a * I_{33} \right)
\end{aligned}$$

Next, we search for the closed form expressions for the integrals I_{31}, I_{32} and I_{33}.

$$\begin{aligned}
I_{31} &= \int \frac{\cos^2\phi}{\sqrt{(2\cos^2\phi)-1)b}} d\phi \\
&= \frac{1}{2} csgn(\cos\phi)(EllipticF(\sin\phi, \sqrt{2}) + EllipticE(\sin\phi, \sqrt{2}))
\end{aligned}$$

$$\begin{aligned}
I_{32} &= \int \frac{1}{\sqrt{(2\cos^2\phi-1)b}} d\phi \\
&= csgn(\cos\phi) EllipticF(\sin\phi, \sqrt{2})
\end{aligned}$$

$$\begin{aligned}
I_{33} &= \int \frac{\sin\phi\cos\phi}{\sqrt{(2\cos^2\phi-1)b}} d\phi \\
&= -\frac{1}{2}\sqrt{2\cos^2\phi - 1} \\
&= -\frac{1}{2}\sqrt{\cos(2\phi)}
\end{aligned}$$

After substituting I_{31}, I_{32} and I_{33} in I_3 and further simplification, we obtain the following primitive:

$$I_4 = \frac{2(b * csgn(\cos\phi))EllipticE(\sin\phi, \sqrt{2}) + a\sqrt{\cos(2\phi)}}{(a^2+b^2)^{3/4}}$$

The $csgn$ function yields 1 or -1. For simplification, we can remove it.

$$I_5 = \frac{2(b * EllipticE(\sin\phi, \sqrt{2}) + a\sqrt{\cos(2\phi)}}{(a^2+b^2)^{3/4}}$$

For the initial integral, $Obs(M_0, [M_1 M_2])$, we obtain the following primitive:

$$I_5 = k_1 \frac{2(b * EllipticE(\sin\phi, \sqrt{2}) + a\sqrt{\cos(2\phi)}}{(a^2+b^2)^{3/4}} + k_2$$

where:

$$k_1 = \frac{1}{\pi}\sqrt{\frac{-(ax_0 + by_0 + c)}{dmax}}$$

and:

$$k_2 \in \mathbb{R}^3$$

$EllipticF$ [mat12b, map12b] is the incomplete elliptic integral of the first kind, and it is defined as

$$E(\phi, k) = \int_0^{\sin\phi} \sqrt{\frac{1}{(1-k^2t^2)(1-t^2)}}\, dt$$

$EllipticE$ [mat12a, map12a] is the incomplete elliptic integral of the second kind, and it is defined as

$$E(\phi, k) = \int_0^{\sin\phi} \sqrt{\frac{1-k^2t^2}{1-t^2}}\, dt$$

Bibliography

[AAM03] Ulf Assarsson and Tomas Akenine-Möller. *A geometry-based soft shadow volume algorithm using graphics hardware.* In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 511–520, New York, NY, USA, 2003. ACM.

[ACFM11] Lilian Aveneau, Sylvain Charneau, Laurent Fuchs, and Frederic Mora. *A Framework for n-Dimensional Visibility Computations.* In Leo Dorst and Joan Lasenby, editors, *Guide to Geometric Algebra in Practice*, pages 273–296. Springer, 2011.

[AF93] David Avis and Komei Fukuda. *Reverse Search for Enumeration. Discrete Applied Mathematics*, 65:21–46, 1993.

[AFO05] Okan Arikan, David A. Forsyth, and James F. O'Brien. *Fast and detailed approximate global illumination by irradiance decomposition.* In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1108–1114, New York, NY, USA, 2005. ACM.

[Ama84] John Amanatides. *Ray tracing with cones.* In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 129–135, New York, NY, USA, 1984. ACM.

[AMA02] Tomas Akenine-Möller and Ulf Assarsson. *Approximate soft shadows on arbitrary surfaces using penumbra wedges.* In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 297–306, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[AOV12] *Ambient Occlusion Volumes application.* http://graphics.cs.williams.edu/papers/AOVHPG10/index.html, 2012. Accessed: 23/05/2012.

[APS91] Boris Aronov, Marco Pellegrini, and Micha Sharir. *On the Zone of a Surface in a Hyperplane Arrangement. DISCRETE COMPUT. GEOM*, 9:177–186, 1991.

[BHS98] J. Bittner, V. Havran, and P. Slavik. *Hierarchical Visibility Culling with Occlusion Trees.* In *Proceedings of the Computer Graphics International 1998*, CGI '98, pages 207–, Washington, DC, USA, 1998. IEEE Computer Society.

[Bir98] Stan Birchfield. *An Introduction to Projective Geometry (for computer vision)*, April 1998.

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

[BP96] Chandrajit L. Bajaj and Valerio Pascucci. *Splitting a complex of convex polytopes in any dimension.* In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 88–97, New York, NY, USA, 1996. ACM.

[BP01] Jiří Bittner and Jan Prikryl. *Exact Regional Visibility using Line Space Partitioning.* Technical Report TR-186-2-01-06, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, March 2001. human contact: technical-report@cg.tuwien.ac.at.

[Bre02] Rob Bredow. *Renderman on film. SIGGRAPH 2002 Course Notes Course*, 16(6):7, 2002.

[BS09] Louis Bavoil and Miguel Sainz. *Multi-layer dual-resolution screen-space ambient occlusion.* In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, pages 45:1–45:1, New York, NY, USA, 2009. ACM.

[Bun05] Michael Bunnell. *Dynamic Ambient Occlusion and Indirect Lighting.* In Matt Pharr and Randima Fernando, editors, *GPU Gems 2*, pages 223–233. Addison-Wesley Professional, 2005.

[BW03] Jiri Bittner and Peter Wonka. *Visibility in Computer Graphics. JOURNAL OF ENVIRONMENTAL PLANNING*, 30:729–756, 2003.

[BWW05] Jiří Bittner, Peter Wonka, and Michael Wimmer. *Fast Exact From-Region Visibility in Urban Scenes.* In Kavita Bala and Philip DutrÃ©, editors, *Rendering Techniques 2005 (Proceedings Eurographics Symposium on Rendering)*, pages 223–230. Eurographics, Eurographics Association, June 2005.

[CEG+96] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, and Jorge Stolfi. *Lines in Space: Combinatorics and Algorithms. Algorithmica*, 15:428–447, 1996.

[Cha07] Sylvain Charneau. *Study and application of geometric algebras to the global visibility computation in a projective space of dimension n >= 2.* PhD thesis, Université de Poitiers, December 2007.

[Chr08] Per H. Christensen. *Point-Based Approximate Color Bleeding.* Technical report, Pixar, 2008.

[CNS00] Francesc Castro Castro, László Neumann, and Mateu Sbert. *Extended Ambient Term. journal of graphics, gpu, and game tools*, 5(4):1–7, 2000.

[Cro77] Franklin C. Crow. *Shadow algorithms for computer graphics.* In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '77, pages 242–248, New York, NY, USA, 1977. ACM.

[CWH93] Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and realistic image synthesis.* Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[DD02] Florent Duguet and George Drettakis. *Robust epsilon visibility. ACM Trans. Graph.*, 21:567–575, July 2002.

[DDP96] Frédo Durand, George Drettakis, and Claude Puech. *The 3D visibility complex: a new approach to the problems of accurate visibility.* In *Proceedings of the eurographics workshop on Rendering techniques 1996*, pages 245–256, London, UK, 1996. Springer-Verlag.

[DDP97] Frédo Durand, George Drettakis, and Claude Puech. *The visibility skeleton: a powerful and efficient multi-purpose global visibility tool.* In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 89–100, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[DDP98] Fredo Durand, George Drettakis, and Claude Puech. *Visibility driven hierarchical radiosity.* In *ACM SIGGRAPH 98 Conference abstracts and applications*, SIGGRAPH '98, pages 263–, New York, NY, USA, 1998. ACM.

[DDP99] Frédo Durand, George Drettakis, and Claude Puech. *Fast and accurate hierarchical radiosity using global visibility. ACM Trans. Graph.*, 18:128–170, April 1999.

[DDP02] Frédo Durand, George Drettakis, and Claude Puech. *The 3D visibility complex. ACM Trans. Graph.*, 21:176–206, April 2002.

[DHS+05] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X. Sillion. *A frequency analysis of light transport.* In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1115–1126, New York, NY, USA, 2005. ACM.

[DKW85] Norm Dadoun, David G. Kirkpatrick, and John P. Walsh. *The geometry of beam tracing.* In *Proceedings of the first annual symposium on Computational geometry*, SCG '85, pages 55–61, New York, NY, USA, 1985. ACM.

[DORP96] Frédo Durand, Rachel Orti, Stéphane Rivière, and Claude Puech. *Radiosity in flatland made visibly simple: using the visibility complex for lighting simulation of dynamic scenes in flatland.* In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 511–512, New York, NY, USA, 1996. ACM.

[Dug04] Florent Duguet. *Shadow Computations using Robust Epsilon Visibility.* Technical Report RR-5167, INRIA, REVES/INRIA Sophia-Antipolis, 2004.

[Dur99] Frédo Durand. *3D Visibility: analytical study and applications.* PhD thesis, Université Joseph Fourier, Grenoble I, July 1999. http://www-imagis.imag.fr.

[EASW09] Elmar Eisemann, Ulf Assarsson, Michael Schwarz, and Michael Wimmer. *Casting Shadows in Real Time.* In *ACM SIGGRAPH ASIA 2009 Courses*, SIGGRAPH ASIA '09, New York, NY, USA, 2009. ACM.

[EBD92] D. Eggert, K. Bowyer, and C. Dyer. *Aspect Graphs: State-of-the-Art and Applications in Digital Photogrammetry.* In *Proceedings of the 17th Congress of the International Society for Photogrammetry and Remote Sensing, Part B5*, pages 633–645, 1992.

[ED07] Elmar Eisemann and Xavier Décoret. *Visibility Sampling on GPU and Applications. Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3), 2007.

[EHDR11] Kevin Egan, Florian Hecht, Frédo Durand, and Ravi Ramamoorthi. *Frequency analysis and sheared filtering for shadow light fields of complex occluders. ACM Trans. Graph.*, 30(2):9:1–9:13, April 2011.

[EOS86] H Edelsbrunner, J O'Rouke, and R Seidel. *Constructing arrangements of lines and hyperplanes with applications. SIAM J. Comput.*, 15:341–363, May 1986.

[ESAW11] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. *Real-Time Shadows.* A.K. Peters, 2011.

[FBP08] Vincent Forest, LoÃ¯c Barthe, and Mathias Paulin. *Accurate Shadows by Depth Complexity Sampling. Computer Graphics Forum*, 27(2):663–674, 2008.

[GCS91] Ziv Gigus, John Canny, and Raimund Seidel. *Efficiently Computing and Representing Aspect Graphs of Polyhedral Objects. IEEE Trans. Pattern Anal. Mach. Intell.*, 13:542–551, June 1991.

[GH98] Djamchid Ghazanfarpour and Jean-Marc Hasenfratz. *A Beam Tracing with Precise Antialiasing for Polyhedral Scenes. Computer Graphics*, 22(1):103–115, 1998.

[GMM90] Ziv Gigus, Student Member, and Jitendra Malik. *Computing the Aspect Graph for Line Drawings Polyhedral Objects. IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:113–122, 1990.

[GSSK05] Ismael Garcia, Mateu Sbert, and L. Szirmay-Kalos. *Tree Rendering with Billboard Clouds.* In *Proceedings of Third Hungarian Conference on Computer Graphics and Geometry*, pages 9–15, Budapest, 2005.

[Hai00] Eric Haines. *A shaft culling tool. Journal of Graphic Tools (http://jgt.akpeters.com/papers/Haines00/)*, 5(1):23–26, 2000.

[HG98] Jean-Marc Hasenfratz and Djamchid Ghazanfarpour. *Une synthèse des variantes du lancer de rayons et du lancer de faisceaux. Revue Internationale de CFAO et d'informatique graphique et d'informatique graphique*, 13(3):235–264, 1998.

[HH84] Paul S. Heckbert and Pat Hanrahan. *Beam tracing polygonal objects.* In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 119–127, New York, NY, USA, 1984. ACM.

[HJ07] Jared Hoberock and Yuntao Jia. *High-Quality Ambient Occlusion.* In Hubert Nguyen, editor, *GPU Gems 3*, pages 257–274. Addison-Wesley Professional, 2007.

[HK85] Martial Hebert and Takeo Kanade. *The 3-D Profile Method for Object Recognition.* In *Proceedings of the 1985 Computer Vision and Pattern Recognition Conference (CVPR '85)*, pages 458–464, 1985.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. *A survey of Real-Time Soft Shadows Algorithms. Computer Graphics Forum*, 22(4):753–774, dec 2003.

[HMN05] Denis Haumont, Otso Makinen, and Shaun Nirenstein. *A Low Dimensional Framework for Exact Polygon-to-Polygon Occlusion Queries.* In *Eurographics Workshop on Rendering*, pages 211–222, 2005.

[HPAD06] Kyle Hegeman, Simon Premoze, Michael Ashikhmin, and George Drettakis. *Approximate Ambient Occlusion For Trees.* In C. Sequin and M. Olano, editors, *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM SIGGRAPH, March 2006.

[HRGZ97] M. Henk, J. Richter-Gebert, and G.M. Ziegler. *Basic Properties of Convex Polytopes.* In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry - 2nd edition*, pages 243–270. CRC Press, 1997.

[Hub93] Philip M. Hubbard. *Interactive Collision Detection.* In *In Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[IKSZ03] A. Iones, A. Krupkin, M. Sbert, and S. Zhukov. *Fast, realistic lighting for video games. Computer Graphics and Applications, IEEE*, 23(3):54 – 64, may-june 2003.

[KA06] Janne Kontkanen and Timo Aila. *Ambient Occlusion for Animated Characters.* In Thomas Akenine-Möller Wolfgang Heidrich, editor, *Rendering Techniques 2006 (Eurographics Symposium on Rendering)*. Eurographics, jun 2006.

[KA07] Adam G. Kirk and Okan Arikan. *Real-time ambient occlusion for dynamic character skins.* In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 47–52, New York, NY, USA, 2007. ACM.

[Kaj86] James T. Kajiya. *The rendering equation.* In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[KD76] J. J. Koenderink and A. J. Doorn. *The singularities of the visual mapping. Biological Cybernetics*, 24:51–59, 1976. 10.1007/BF00365595.

[KD79] J. J. Koenderink and A. J. Doorn. *The internal representation of solid shape with respect to vision. Biological Cybernetics*, 32:211–216, 1979. 10.1007/BF00337644.

[KH01] Alexander Keller and Wolfgang Heidrich. *Interleaved Sampling.* In *Proceedings of the Eurographics Workshop on Rendering*, 25–27 June 2001. To appear.

[KL05] Janne Kontkanen and Samuli Laine. *Ambient Occlusion Fields.* In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pages 41–48. ACM Press, 2005.

[KLA04] Jan Kautz, Jaakko Lehtinen, and Timo Aila. *Hemispherical Rasterization for Self-Shadowing of Dynamic Objects.* In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 179–184. Eurographics Association, 2004.

[LA05] Samuli Laine and Timo Aila. *Hierarchical Penumbra Casting. Computer Graphics Forum*, 24(3):313–322, 2005.

[LAA+05] Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen, and Tomas Akenine-Möller. *Soft shadow volumes for ray tracing.* In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1156–1165, New York, NY, USA, 2005. ACM.

[Lam60] Johann Heinrich Lambert. *I.H. Lambert Photometria, sive, De mensura et gradibus luminis, colorum et umbrae [microform].* V.E. Klett, Augustae Vindelicorum :, 1760.

[Lan02] Hayden Landis. *Production-Ready Global Illumination.* In *Siggraph Course Notes*, volume 16, 2002.

[LK10] Samuli Laine and Tero Karras. *Two Methods for Fast Ray-Cast Ambient Occlusion. Computer Graphics Forum*, 29(4):1325–1333, 2010.

[LLA06] Jaakko Lehtinen, Samuli Laine, and Timo Aila. *An Improved Physically-Based Soft Shadow Volume Algorithm. Computer Graphics Forum*, 25(3):303–312, 2006.

[MA05] F. Mora and L. Aveneau. *Fast and Exact Direct Illumination.* June 2005. Proceedings of CGI'2005, New York, Stony Brooks.

[MAM05] F. Mora, L. Aveneau, and M. Mériaux. *Coherent and Exact Polygon-to-Polygon Visibility.* In *Proceedings of WSCG'05*, 2005.

[map12a] *EllipticE - Maple Help.* http://www.maplesoft.com/support/help/Maple/view.aspx?path=EllipticE, 2012. Accessed: 27/06/2012.

[map12b] *EllipticF - Maple Help.* http://www.maplesoft.com/support/help/Maple/view.aspx?path=EllipticF, 2012. Accessed: 27/06/2012.

[mat12a] *EllipticE.* http://mathworld.wolfram.com/EllipticIntegraloftheSecondKind.html, 2012. Accessed: 27/06/2012.

[mat12b] *EllipticF.* http://mathworld.wolfram.com/EllipticIntegraloftheFirstKind.html, 2012. Accessed: 27/06/2012.

[McG10] Morgan McGuire. *Ambient Occlusion Volumes.* In *Proceedings of High Performance Graphics 2010*, June 2010.

[MFS09] A. Méndez-Feliu and Mateu Sbert. *From obscurances to ambient occlusion: A survey. The Visual Computer*, 25:181–196, 2009. 10.1007/s00371-008-0213-4.

[MFSC+06] Alex Méndez Feliu, Mateu Sbert, Jordi Catà, Nicolau Sunyer, and Sergi Funtané. *Real-Time Obscurances with Color Bleeding (GPU Obscurances with Depth Peeling).* In Wolfgang Engel, editor, *ShaderX4: Advanced Rendering Techniques*, chapter 2.6. Charles River Media, 2006.

[Mil94] Gavin Miller. *Efficient algorithms for local and global accessibility shading.* In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 319–326, New York, NY, USA, 1994. ACM.

[Mit07] Martin Mittring. *Finding next gen: CryEngine 2.* In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[Mor06] Frédéric Mora. *Visibilité polygone à polygone : calcul, représentation, applications.* PhD thesis, Université de Poitiers, July 2006.

[MP99] David M. Mount and Fan-Tao Pu. *Binary Space Partitions in Plücker Space.* In *ALENEX '99: Selected papers from the International Workshop on Algorithm Engineering and Experimentation*, pages 94–113, London, UK, 1999. Springer-Verlag.

[MS04] A. Méndez and Mateu Sbert. *Comparing hemisphere sampling techniques for obscurances computation.* In *Proceedings of the International Conference on Computer Graphics and Artificial Intelligence (3IA 2004)*, 2004.

[MS06] A. Méndez and Mateu Sbert. *Obscurances in general environments.* In *Proceedings of Graphicon 2006*, 2006.

[MSC03] A. Méndez, Mateu Sbert, and Jordi Catá. *Real-time obscurances with color bleeding.* In *Proceedings of the 19th spring conference on Computer graphics*, SCCG '03, pages 171–176, New York, NY, USA, 2003. ACM.

[MSN03] A. Méndez, Mateu Sbert, and L Neumann. *Obscurances for ray-tracing.* EUROGRAPHICS 2003 Poster Presentation, 2003.

[NBG02] S. Nirenstein, E. Blake, and J. Gain. *Exact from-region visibility culling.* In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 191–202, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[Nie92] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[Nir03] Shaun Nirenstein. *Fast and accurate visibility preprocessing.* PhD thesis, University of Cape Town, October 2003. http://www.nirenstein.com.

[NN85] Tomoyuki Nishita and Eihachiro Nakamae. *Continuous tone representation of three-dimensional objects taking account of shadows and interreflection.* In *SIGGRAPH*, pages 23–30. ACM, 1985.

[ORDP96] Rachel Orti, Stéphane Rivière, Frédo Durand, and Claude Puech. *Radiosity for dynamic scenes in flatland with the visibility complex.* In Jarek Rossignac and François Sillion, editors, *Computer Graphics Forum (Proc. of Eurographics '96)*, volume 16, pages 237–249, Poitiers, France, Sep 1996.

[ORM07] Ryan Overbeck, Ravi Ramamoorthi, and William R. Mark. *A Real-time Beam Tracer with Application to Exact Soft Shadows.* In *Eurographics Symposium on Rendering*, Jun 2007.

[PD90] Harry Plantinga and Charles R. Dyer. *Visibility, occlusion, and the aspect graph. International Journal of Computer Vision*, 5:137–160, 1990. 10.1007/BF00054919.

[Pel91] Marco Pellegrini. *Ray-shooting and isotopy classes of lines in 3-dimensional space.* In Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 519 of *Lecture Notes in Computer Science*, pages 20–31. Springer Berlin / Heidelberg, 1991. 10.1007/BFb0028246.

[Pel93] M. Pellegrini. *Ray shooting on triangles in 3-space. Algorithmica*, 9:471–494, 1993.

[Pel97] Marco Pellegrini. *Ray shooting and lines in space.* In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 599–614. CRC Press, Boca Raton-New York, 1997.

[Pel04] M. Pellegrini. *Handbook of Discrete and Computanional Geometry - second edition*, pages 839–856. 2004. Ray shooting and lines in space.

[PSD90] Harry Plantinga, W. Brent Seales, and Charles R. Dyer. *Real-time hidden-line elimination for a rotating polyhedral scene using the aspect representation.* In *In Proceedings of Graphics Interface 1990*, pages 9–16, 1990.

[PV93] Michel Pocchiola and Gert Vegter. *The visibility complex.* In *Proceedings of the ninth annual symposium on Computational geometry*, SCG '93, pages 328–337, New York, NY, USA, 1993. ACM.

[PV96] Michel Pocchiola and Gert Vegter. *Topologically sweeping visibility complexes via pseudotriangulations. Discrete Computational Geometry*, 16(4):419–453, December 1996. Special issue devoted to the proceedings of the 11th Annu. ACM Sympos. Comput. Geom. (SCG'95).

[RAMN12] Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. *A Theory of Monte Carlo Visibility Sampling. ACM Transactions on Graphics*, 2012.

[RB85] William T. Reeves and Ricki Blau. *Approximate and probabilistic algorithms for shading and rendering structured particle systems. SIGGRAPH Comput. Graph.*, 19:313–322, July 1985.

[RGS09] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. *Approximating dynamic global illumination in image space.* In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 75–82, New York, NY, USA, 2009. ACM.

[Riv95] Stéphane Rivière. *Topologically sweeping the visibility complex of polygonal scenes.* In *Proceedings of the eleventh annual symposium on Computational geometry*, SCG '95, pages 436–437, New York, NY, USA, 1995. ACM.

[Riv97a] Stéphane Rivière. *Dynamic visibility in polygonal scenes with the visibility complex.* In *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, pages 421–423, New York, NY, USA, 1997. ACM.

[Riv97b] Stéphane Rivière. *Walking in the visibility complex with applications to visibility polygons and dynamic visibility.* In *9th Canadian Conference on Computational Geometry, CCCG97,*, Kingston, Canada, 1997.

[RSH05] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. *Multi-level ray tracing algorithm.* In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1176–1185, New York, NY, USA, 2005. ACM.

[SA07] Perumaal Shanmugam and Okan Arikan. *Hardware accelerated ambient occlusion techniques on GPUs.* In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 73–80, New York, NY, USA, 2007. ACM.

[SCLR99] Michael M. Stark, Elaine Cohen, Tom Lyche, and Richard F. Riesenfeld. *Computing exact shadow irradiance using splines.* In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 155–164, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[SEA08] Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. *Sample-based Visibility for Soft Shadows Using Alias-free Shadow Maps. Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2008)*, 27(4):1285–1292, June 2008.

[SP94] Francois X. Sillion and Claude Puech. *Radiosity and Global Illumination.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[SR00] Michael M. Stark and Richard F. Reisenfield. *Exact Illumination in Polygonal Environments using Vertex Tracing.* In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 149–160, London, UK, UK, 2000. Springer-Verlag.

[SR01] Michael M. Stark and Richard F. Riesenfeld. *Reflected and Transmitted Irradiance from Area Sources Using Vertex Tracing.* In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 13–24, London, UK, UK, 2001. Springer-Verlag.

[SSS74] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. *A Characterization of Ten Hidden-Surface Algorithms. ACM Comput. Surv.*, 6:1–55, March 1974.

[SSSK04] László Szécsi, Mateu Sbert, and László Szirmay-Kalos. *Combined Correlated and Importance Sampling in Direct Light Source Computation and Environment Mapping. Comput. Graph. Forum*, 23(3):585–594, 2004.

[SSZK04] M. Sattler, R. Sarlette, G. Zachmann, and R. Klein. *Hardware-Accelerated Ambient Occlusion Computation.* In *9th Int'l Fall Workshop VISION, MODELING, AND VISUALIZATION (VMV)*, pages 119–135, Stanford (California), USA, November16–18 2004.

[Sta02] Michael M. Stark. *Analytic Illumination in Polyhedral Environments (extended version).* PhD thesis, The University in Utah, May 2002.

[STN87] Mikio Shinya, T. Takahashi, and Seiichiro Naito. *Principles and applications of pencil tracing.* In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 45–54, New York, NY, USA, 1987. ACM.

[Sto87] Jorge Stolfi. *Oriented Projective Geometry.* In *Proceedings of the 3rd Symposium on Computational Geometry (SOCG)*, pages 76–85, June 1987.

[Sto91] Jorge Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations.* Academic Press, 1991.

[SWZ96] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. *Monte Carlo techniques for direct lighting calculations. ACM Trans. Graph.*, 15(1):1–36, January 1996.

[TCM06] Marco Tarini, Paolo Cignoni, and Claudio Montani. *Ambient occlusion and edge cueing to enhance real time molecular visualization. IEEE Transaction on Visualization and Computer Graphics*, 12(6), sep/oct 2006.

[Tel92] Seth Jared Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, University of California at Berkeley, October 1992.

[TH92] Seth J. Teller and Michael E. Hohmeyer. *Stabbing Oriented Convex Polygons in Randomized O(n^2) Time.* Technical Report UCB/CSD-92-669, EECS Department, University of California, Berkeley, Jan 1992.

[TH93] Seth Teller and Pat Hanrahan. *Global visibility algorithms for illumination computations.* In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 239–246, New York, NY, USA, 1993. ACM.

[TH99] Seth Teller and Michael Hohmeyer. *Determining the lines through four lines. J. Graph. Tools*, 4:11–22, November 1999.

[TS91] Seth J. Teller and Carlo H. Séquin. *Visibility preprocessing for interactive walkthroughs.* In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 61–70, New York, NY, USA, 1991. ACM.

[Wal07] Ingo Wald. *On fast Construction of SAH-based Bounding Volume Hierarchies.* In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 33–40, Washington, DC, USA, 2007. IEEE Computer Society.

[WBWS01] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. *Interactive Rendering with Coherent Ray Tracing*. In A. Chalmers and T. Rhyne, editors, *Computer Graphics Forum*, volume 20, pages 153–164. Blackwell, 2001.

[WIK+06] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. *Ray tracing animated scenes using coherent grid traversal*. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 485–493, New York, NY, USA, 2006. ACM.

[wik12a] *Degrees of Freedom*. http://www.beam-wiki.org/wiki/Degrees_of_Freedom, 2012. Accessed: 29/02/2012.

[wik12b] *Importance Sampling*. http://en.wikipedia.org/wiki/Importance_sampling, 2012. Accessed: 20/03/2012.

[wik12c] *Polytope*. http://en.wikipedia.org/wiki/Polytope, 2012. Accessed: 20/02/2012.

[WK06] Carsten Wächter and Alexander Keller. *Instant Ray Tracing: The Bounding Interval Hierarchy*. In *In Rendering Techniques 2006 - Proceedings of the 17th Eurographics Symposium on Rendering*, pages 139–149, 2006.

[WSBW01] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. *Interactive Rendering with Coherent Ray Tracing*. In *Computer Graphics Forum*, pages 153–164, 2001.

[ZIK98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. *An Ambient Light Illumination Model*. In *Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques*, pages 45–56, 1998.

Visibilité analytique dans l'espace de Plücker: De la théorie aux applications pratiques

Résumé

Cette thèse aborde le problème du calcul analytique de la visibilité depuis un polygone, dans un contexte applicatif. Premièrement, nous mettons en évidence le fossé existant entre la théorie qui permet une élégante description du problème, et la pratique qui peine à proposer des implémentations robustes. Par la suite, nous nous appuyons sur le concept de classes d'équivalence de droites orientées afin de proposer deux nouveaux algorithmes qui encodent l'occlusion et la visibilité depuis un polygone dans l'espace de Plücker. Les deux algorithmes utilisent cette information pour exploiter la cohérence visuelle qui existe entre des points voisins sur un même polygone et accélérer les calculs de visibilité depuis ces points. Contrairement aux méthodes existantes, notre approche est très robuste, et l'information est encodée de manière paresseuse à l'exécution. Les deux contextes applicatifs sont le calcul des ombres douces et de l'occlusion ambiante en synthèse d'images. Notre algorithme d'occlusion distingue les droites occultées et non occultées. Il est utilisé pour calculer les fragments exacts d'une source surfacique visibles depuis les points à ombrer. Notre algorithme de visibilité enrichit cette définition et regroupe de manière analytique les rayons issus d'une surface selon les premiers objets intersectés. Les deux algorithmes obtiennent des résultats de haute qualité sans bruit ou autre artéfact, contrairement aux méthodes stochastiques ou basées sur une approximation de la visibilité. De plus, nos algorithmes sont compétitifs par rapport à une solution de production standard.

Mots clés : visibilité analytique depuis un polygone, espace de Plücker, ombres douces, occlusion ambiante

Analytic visibility in Plücker space: From theory to practical applications

Abstract

This thesis addresses the from-polygon visibility problem in an applicative context. First of all, we underline the void which exists between the theory which allows an elegant description of this problem and the practice which struggles to provide efficient implementations. Then, we build on the theoretical concept of equivalence classes of oriented lines in order to propose two new algorithms which analytically encode from-polygon occlusion and from-polygon visibility in Plücker space. Both algorithms use this information to take advantage of the visual coherence which exists between neighbor points on the same polygon and speed up individual from-point queries. Contrary to previous methods, our approach is very robust, and the information is encoded lazily at run time, as directed by the visibility queries. The two applicative contexts are the generation of soft shadows and the calculation of ambient occlusion in image synthesis. Our from-polygon occlusion algorithm distinguishes between occluded and unoccluded lines and it is used to calculate the exact fragments of a light source which are visible from the points to shade. Our from-polygon visibility algorithm improves this definition and analytically groups the view rays issued from a surface according to the first geometry they intersect. Both our algorithms achieve high quality results that are not sensitive to noise or other visual imperfections, contrary to other methods based on either sampling or visibility approximations. Moreover, our algorithms are competitive with respect to a standard production solution.

Keywords : analytic from-polygon visibility, Plücker space, soft shadows, ambient occlusion

XLIM - UMR CNRS n^{o}7252
123, avenue Albert Thomas - 87060 LIMOGES CEDEX