Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

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

D. Haumont[1] and O. Mäkinen[2] and S. Nirenstein[3]

[1]Université Libre de Bruxelles [2]Hybrid Graphics Ltd. and University of Helsinki [3]University of Cape Town

Abstract

Despite the importance of from-region visibility computation in computer graphics, efficient analytic methods are still lacking in the general 3D case. Recently, different algorithms have appeared that maintain occlusion as a complex of polytopes in Plücker space. However, they suffer from high implementation complexity, as well as high computational and memory costs, limiting their usefulness in practice.

In this paper, we present a new algorithm that simplifies implementation and computation by operating only on the skeletons of the polyhedra instead of the multi-dimensional face lattice usually used for exact occlusion queries in 3D. This algorithm is sensitive to complexity of the silhouette of each occluding object, rather than the entire polygonal mesh of each object. An intelligent feedback mechanism is presented that greatly enhances early termination by searching for apertures between query polygons. We demonstrate that our technique is several times faster than the state of the art.

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

1. Introduction

The *from-region* visibility problem refers to the determination of the set of 3D primitives visible from a volumetric region V, through a set of polygonal occluders O_i. This is a central problem in many computer graphics algorithms, such as global illumination and occlusion culling.

The most direct application of a from-region visibility solution is the computation of Potentially Visible Sets (PVS) [ARB90]. The navigable space of a scene is decomposed into volumetric *view cells*, while the objects that are visible from each view cell are computed using from-region visibility techniques and stored in the PVS data structure. During an interactive exploration of the scene, only the objects associated with the view cell containing the camera position need to be drawn. This can lead to notable speed gains when displaying scenes with high overdraw since the number of objects visible from each cell is usually much smaller than the size of the database.

Until recently, analytic from-region visibility algorithms have been considered impractical due to the costs involved and many alternative solutions have been proposed:

- Conservative solutions [LSCO03] usually make simplifying assumptions for computational efficiency, but almost always overestimate the visibility.
- Aggressive solutions [NB04] are typically based on a sampling process. They are fast and simple to implement, but the sampling practically always underestimates the visibility, leading to errors in the output image.

Detailed surveys about conservative and aggressive occlusion culling methods can be found in [COCSD02, PT02].

Analytic approaches were first introduced for $2D$ and $2\frac{1}{2}D$ scenes [KCoC01, BWW01]. Recent works have demonstrated that analytic computation is also possible in $3D$ by formulating visibility in Plücker space [NBG02, Bit02]. Instead of solving the from-region visibility problem directly, these methods replace it by several simpler from-surface visibility problems. Since any visible ray originating from the region intersects one of its boundary faces, the set of primitives visible from the region is equal to the union of the primitives seen by its boundary faces (and the primitives contained in the region), showing that the problems can be solved by using only from-surface techniques. Using this observation, existing analytic 3D methods compute

the from-surface occlusion information by using CSG operations on *polytopes* (i.e. bounded polyhedra) in Plücker space. However, current methods have a considerable complexity both in terms of computation time and implementation. In practice, these drawbacks have prevented them from being adopted by the computer graphics community and industry. In this paper, we address the issues of previous analytic from-region visibility methods by developing a simpler, more efficient and more robust mechanism for computing the visibility queries.

1.1. Contributions

Low dimensional algorithms in Plücker space: We show that maintaining the occlusion in Plücker space only requires maintaining the *1-skeleton* of the polytopes (i.e. the vertices and the edges of the polytopes), instead of the full face lattice used in previous work (Section 3).

Efficient polygon-to-polygon occlusion query: We employ the low dimensional algorithms in the context of polygon-to-polygon visibility query, and combine new techniques to further enhance the efficiency (Section 4):

- we propose the casting of rays into the apertures left by the already processed occluders. By construction, the method quickly detects any aperture existing between the polygons, and allows an early termination of the query in the case of mutual visibility. The rays are also used for occluder selection, to ensure that each occluder we process will block some part of the not yet processed *line space*.
- we propose a new occluder fusion mechanism specifically designed to pair with the Plücker-space mapping of the visibility query. By discarding the occluder's edges that are not part of the *from-region silhouettes* we show that many redundant computations can be avoided.

The rest of the paper is organized as follows. After presenting the general principle of previous analytic visibility approaches in Section 2, the low dimensional algorithms are described and used in a new polygon-to-polygon occlusion query framework in Sections 3 and 4. We evaluate this framework in Section 5 and conclude in Section 6.

2. Analytic from-region visibility

2.1. Introduction to Plücker coordinates

Let l be an oriented line in the 3D Euclidean space $\mathbb{E}^3$, passing first through point $P(p_x, p_y, p_z)$ and then through point $Q(q_x, q_y, q_z)$. This line is parameterized in the Plücker space by the Plücker coordinates π_i^l:

$$\begin{cases} \pi_0^l = q_x - p_x \\ \pi_1^l = q_y - p_y \\ \pi_2^l = q_z - p_z \\ \pi_3^l = q_z p_y - q_y p_z \\ \pi_4^l = q_x p_z - q_z p_x \\ \pi_5^l = q_y p_x - q_x p_y \end{cases} \quad (1)$$

The coordinates can be interpreted in two ways:

- as the homogeneous coordinates of a point $l^*(\pi_0^l, \pi_1^l, \pi_2^l, \pi_3^l, \pi_4^l, \pi_5^l)$
- as the coefficients of an hyperplane H_l of equation

$$H_l \equiv \pi_3^l x_0 + \pi_4^l x_1 + \pi_5^l x_2 + \pi_0^l x_3 + \pi_1^l x_4 + \pi_2^l x_5 = 0 \quad (2)$$

It is important to note that the Plücker space is a projective space $\mathbb{P}^5$, which means that the Plücker coordinates are equivalent within a positive multiplicative coefficient depending on the choice of P and Q to define the line. The point l^* of $\mathbb{P}^5$ can be seen as a ray through the origin in $\mathbb{R}^6$, and H_l as an hyperplane containing the origin in $\mathbb{R}^6$.

Let H_a be the dual hyperplane of the oriented line a and b^* the dual point of the oriented line b. The sign of the expression $H_a(b^*)$, gives the relative orientation of the lines a and b in 3D space $\mathbb{E}^3$. If $H_a(b^*) > 0$ the lines are skew and pass each other in the left handed way. If $H_a(b^*) < 0$, they are skew and right handed oriented. If $H_a(b^*) = 0$, the lines intersect each other. All lines in $\mathbb{E}^3$ map to points in $\mathbb{P}^5$, but the opposite is not true. The only points in $\mathbb{P}^5$ that have a correspondence in $\mathbb{E}^3$ belong to a manifold, called the Plücker quadric, given by the equation

$$G = \{H_x(x^*) = 0 : x \in \mathbb{P}^5\} \setminus \{0\} \quad (3)$$

The Plücker quadric is a 4D manifold, whose 3D analog would be a hyperboloid of one sheet. The other points in $\mathbb{P}^5$ correspond to lines with imaginary coefficients that do not exist in $\mathbb{E}^3$.

2.2. General principle of analytic visibility

The from-surface visibility problem refers to the determination of the set of polygonal primitives of a scene that are visible from the polygonal surface S through a set of polygonal occluders O_i. In the case of occlusion culling, S is the face of a view cell and R is a target of the visibility query: e.g., a scene primitive or a face of a bounding box. We call a *stabbing line* an oriented line that intersects a set of polygons. Let R be a convex polygonal scene primitive. To determine if R is visible from at least one point of S, analytic methods represent the sets of lines between S and R as polyhedra in dual space (see Figure 1). The set of lines stabbing simultaneously S and R is represented by the polyhedron $\mathcal{P}(S,R)$. Each occluder intercepts a set of lines that can also be represented by the polyhedron $\mathcal{O}_i(S,R)$.

- The convex polyhedron $\mathcal{P} \cap \mathcal{O}_i$ contains the subset of lines stabbing S and R, and **blocked** by O_i.
- The non-convex polyhedron $\mathcal{P} - \mathcal{O}_i$ contains the set of lines stabbing S and R, and **not blocked** by O_i. To avoid dealing with a non-convex polyhedra it is usually split into convex parts, forming together a *complex* of convex polyhedra (in the rest of the paper a complex $\mathcal{C}$ designates a set of convex polyhedra $\mathcal{P}_i$).

The set of lines that are not blocked by a set of n convex occluders O_i are computed by successive subtraction of the n occluder polyhedra $\mathcal{O}_i$ from $\mathcal{P}$, using CSG in dual space. The result is maintained as a complex of polyhedra $\mathcal{C}$, representing the set of lines that are not blocked by the already processed occluders. The subtraction operation itself is performed by splitting the polytope $\mathcal{P}$ by the hyperplanes of $\mathcal{O}_i$, and the subset of $\mathcal{P}$ that is located inside $\mathcal{O}_i$ is eliminated.

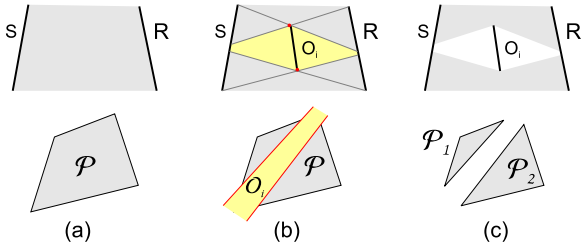


Figure 1: *(a) The polytope $\mathcal{P}$, representing in primal space the set of lines stabbing the query polygons. (b) The occluder O_i blocks a set of lines. (c) The complex $\mathcal{C} = \{\mathcal{P}_1, \mathcal{P}_2\}$ containing the result of the operation $\mathcal{P} - \mathcal{O}_i$. In primal space it represents the set of lines that were not blocked by the occluder O_i*

If the polytope $\mathcal{P}$ becomes entirely eliminated by the subtraction of a set of polytopes $\mathcal{O}_i$, R is hidden by the set of occluders O_i. Otherwise, some unblocked lines exist and R is visible from S (see Figure 2).

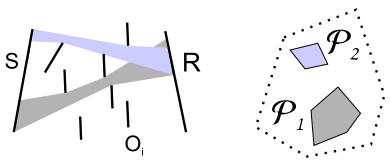


Figure 2: *After all the occluders have been processed, the remaining complex represents the set of lines that have not been blocked.*

In 2D, the dual space is also two-dimensional, and efficient algorithms exist [KCoC01, BWW01]. In 3D, the problem is more involved because the dual space is the five-dimensional projective Plücker space. The transformation of the problem into this dual space is obtained by using equation (2) to transform the lines containing the edges of the scene polygons to hyperplanes in Plücker space (see Figure 3). It can be shown that the set of stabbing lines through one polygon can be represented by a *polyhedron* (i.e. an unbounded convex region of space), and the set of stabbing lines between two polygons can be represented by a polytope (i.e. a bounded polyhedron) [Nir03] (the edges of the query polygons must be ordered so that any stabbing line b passes them with the same relative orientation). In both cases, the set of stabbing lines is represented by the portion of the Plücker quadric delimited by the polyhedron. In particular, the dual point b^* of the stabbing line b is located on the Plücker quadric surface and inside the polyhedron. The intersection points s^* of the edges with the Plücker quadric are the *extremal stabbing lines* of the polygons [Tel92]. In 3D, the line s is incident on four polygons edges (or more in degenerate configurations) (see Figure 3). A subtraction operation can possibly create a polytope that does not intersect with the Plücker quadric: this polytope can be deleted because it does not contain any real stabbing lines [Pu98]. The next section presents the previous analytic from-region visibility techniques using these general principles.

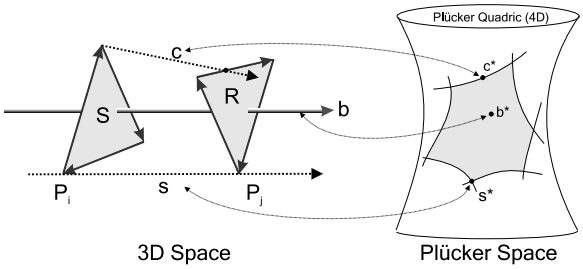


Figure 3: *Correspondence between 3D and Plücker Space (Note: this is an evocation of the reality, because the dual Plücker space is actually $\mathbb{P}^5$). The initial polygon edges are mapped to hyperplanes in Plücker space. The lines incident on four polygon edges are the extremal stabbing lines s. The curves on the Plücker quadric are the traces of the 2-faces of the polytope (i.e. the faces of dimension 2), and correspond to lines incident on 3 polygon edges.*

2.3. Previous work

In 2D, Koltun et al. proposed to determine the visibility of each geometric primitive individually by using polygon-to-polygon occlusion queries [KCoC01]. The dual space is also two dimensional, and the problem can be discretized, which gives the possibility to implement the subtraction operation with graphics hardware rasterization. The polygon-to-polygon query mechanism was extended to 3D by Nirenstein et al. [NBG02]. The visibility of a polygon R is determined by first constructing a bounded polytope $\mathcal{P}$, representing the set of stabbing lines crossing R and the source polygon S. Each occluder polytope is iteratively subtracted from $\mathcal{P}$. The polygons S and R are proven to be hidden if $\mathcal{P}$ gets completely removed. The authors proposed a framework including several optimizations for PVS computation.

Bittner et al. represent all the unblocked rays leaving the query polygon. The approach was first proposed in 2D [BWW01] and then extended to 3D [Bit02]. The rays are encoded in an occlusion tree [BHS98], which is a BSP tree in Plücker space. Each internal node contains a hyperplane equation that corresponds to the edge of an occluder. A leaf node represents either an unblocked region of Plücker space

(out-leaf) or a blocked line region (in-leaf). The occlusion tree construction necessitates a front-to-back ordering of the occluders. A bounded polytope $\mathcal{O}_i$ is constructed for each occluder O_i, representing the set of stabbing lines intersecting O_i and the source polygon S. The polytope $\mathcal{O}_i$ is filtered down in the tree, from the root to the leafs. Each internal node splits the initial polytope into two fragments that are processed in the node's two subtrees. When a fragment reaches an out-leaf node, the node is replaced by a subtree constructed from the polytope's hyperplanes. If an in-leaf is reached, the fragment is eliminated. After all the occluders have been inserted, the occlusion trees can be used to test the occlusion of the geometric primitives of the scene.

Recently, Mora et al. [MAM05] proposed to reduce the fragmentation of the polytope complex, induced by the method of Nirenstein et al. [NBG02], by detecting and discarding the redundant split operations.

The existing methods indicate that representing the occlusion by a complex of polytopes in Plücker space requires two fundamental operations: the creation of a polytope representing the set of lines stabbing two polygons, and a polytope split algorithm to perform the CSG operations.

The polytope representing the set of lines stabbing two polygons was obtained from a vertex enumeration algorithm [AF96] in all the previous approaches: the edges of the polygons are transformed to hyperplanes that correspond to the facets of the polytope. From these equations, the vertex enumeration algorithm outputs the 1-skeleton of the polytope. In his thesis, Nirenstein proposed a direct construction algorithm, but no implementation was evaluated [Nir03].

The polytope splits are also performed using a vertex enumeration algorithm in [Bit02]: the two polytopes are obtained by adding the splitting hyperplane to the list of facets of the initial polytope. In [NBG02], Nirenstein et al. proposed a more efficient split algorithm adapted from [BP96], based on the face lattice of the polytope [FR94]. A polytope split consists of iterating through all the k-faces of the polytope in all dimensions, starting with the 1-dimensional faces and finishing with the d-dimensional faces, and performing symbolic and numerical computation on the polytope's face-lattice. This split operation was also used in [MAM05].

3. Low dimensional algorithms in Plücker space

Maintaining visibility relationships in Plücker space requires two fundamental operations: a polytope split procedure and the creation of a polytope representing the set of lines stabbing two polygons. In this section we propose new algorithms for performing these tasks using only the 1-skeleton of the polytopes (i.e. their vertices and edges) and the *combinatorial description* of their vertices (i.e. the list of facets they belong to).

This approach is conceptually similar to the introduction of the Visibility Skeleton for global visibility computation [DDP97], instead of the Visibility Complex [DDP96]. However, this comparison is not entirely correct, because the 1-skeleton of the polytopes contains different information than the visibility skeleton. The visibility skeleton encodes the *critical swaths* (i.e. the surfaces delimiting the visibility discontinuities) and the extremal stabbing lines, while the 1-skeleton only encodes the extremal stabbing lines explicitly. However, they are still grouped into polytopes, corresponding to higher dimensional cells of the visibility complex: this latter can still be reconstructed from the 1-skeleton representation.

Section 3.1 gives a general d-dimensional polytope split algorithm. In section 3.2, we show how to construct the 1-skeleton of a polytope $\mathcal{P}$ in Plücker space from its intersection with the Plücker quadric representing the set of lines stabbing two convex polygons S and R. A polygon-to-polygon occlusion query framework, based on top of these algorithms, is presented in the section .

3.1. d-dimensional polytope splitting algorithm

Let $\mathcal{P}$ be a bounded polytope in $\mathbb{R}^d$, H_i the hyperplane supporting its facet i, V_i one of its vertices and $E[V_i, V_j]$ one of its edges. We split this polytope by the hyperplane H_s to obtain the two polytopes $\mathcal{P}^-$ and $\mathcal{P}^+$.

Figure 4 shows an illustrative example in $2D$, in which case the convex polytope is a convex polygon, and its facets are equal to its edges. We indicate with each vertex its combinatorial description. Note that the algorithm is identical regardless of the dimensionality of the polytope. The splitting algorithm is divided into 3 steps: classification of the vertices, splitting of the edges, and finally linking of the new vertices:

Step 1: The first step is to classify each vertex V_i as $\{\boxed{-}, \boxed{=}, \boxed{+}\}$, with respect to its relative position with the hyperplane H_s. The vertices $\boxed{-}$ are copied into a new polytope $\mathcal{P}^-$, while the vertices $\boxed{+}$ are copied in a new polytope $\mathcal{P}^+$. If a vertex in $\mathcal{P}$ belongs to H_s, it is classified $\boxed{=}$, and added to $\mathcal{P}^-$ and $\mathcal{P}^+$. H_s is added to the combinatorial description of the vertex.

Step 2: For each edge $E[V_m, V_n]$ linking two vertices V_m and V_n of different signs, a new vertex V_s, labeled $\boxed{=}$, is added at the intersection of the edge and of the hyperplane H_s. The combinatorial description of this vertex is equal to the combinatorial description of the edge $E[V_m, V_n]$, augmented by the hyperplane H_s. The split edge $E[V_m, V_n]$ of $\mathcal{S}$ becomes the edge $E[V_m, V_s]$ of $\mathcal{S}^-$ and the edge $E[V_s, V_n]$ of $\mathcal{S}^+$. The edges of $\mathcal{S}$ linking two vertices $\boxed{-}$ (resp. $\boxed{+}$) are duplicated into $\mathcal{S}^-$ (resp. $\mathcal{S}^+$).

Step 3: The last step creates the new edges of $\mathcal{S}^-$ and $\mathcal{S}^+$. All these edges are located on the hyperplane H_s, and

link together vertices equal to [=]. We use the combinatorial description of the vertices [=] to create the new edges. If the polytope is simple, the vertices labeled [=] that must be linked by an edge are those that have $d-1$ common facets (d-1 is equal to 4 in Plücker space) in their combinatorial description. When the polytope is not simple, the edges can have more than $d-1$ facets in their combinatorial description. In that case, a new edge is created only if the common facet hyperplanes intersect in a line (i.e. the matrix built from their equations is of rank $d-1$).

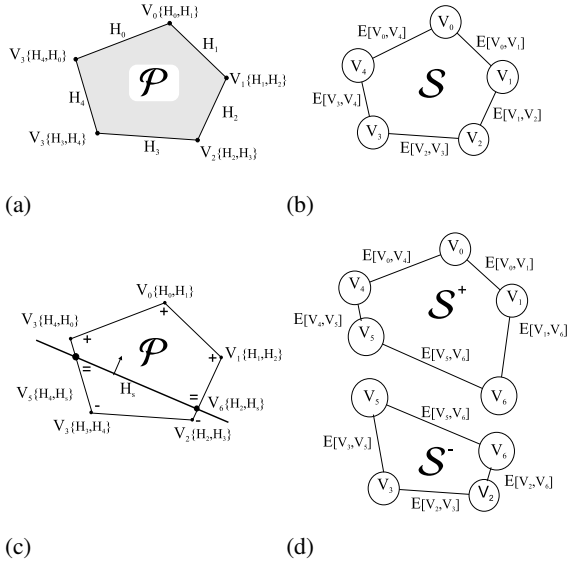


Figure 4: *(a) The polytope $\mathcal{P}$. (b) The 1-skeleton $\mathcal{S}$ of $\mathcal{P}$. (c) Classification of the vertices of $\mathcal{P}$ with respect to the splitting hyperplane H_s. (d) $\mathcal{S}^-$ and $\mathcal{S}^+$ after the split.*

3.2. Constructing the stabbing lines between two 3D polygons

In previous works [NBG02, Bit02], a vertex enumeration algorithm is used for the construction of initial polytopes. The approach is the most general possible, and is much slower than a dedicated algorithm, since it does not take the specifics of the problem into account. Furthermore, the approach is very sensitive to numerical imprecision, and does not always produce a correct solution in particular polygon configurations. In contrast, our approach is based on the explicit construction of the extremal stabbing lines between the two polygons, and always produces a valid result. A similar method was mentioned in [Nir03] and in [MAM05], but was neither evaluated nor detailed.

Let Pl_S and Pl_R be the oriented planes containing the query polygons S and R respectively. Before computing $\mathcal{P}$, we first clip S with Pl_R and R with Pl_S, and keep the parts of the polygons located on the positive side of the splitting planes. If one of the polygons is removed completely in the clipping stage, no stabbing lines exist, and the polytope is the empty set. Otherwise, line r is computed as the intersection of Pl_S and Pl_R (see Figure 5). (Note: if Pl_s and Pl_r are parallel, r^* is equal to $(0,0,0,-n_x,-n_y,-n_z)$, with $\overline{n}(n_x,n_y,n_z)$ the normal of Pl_s.)

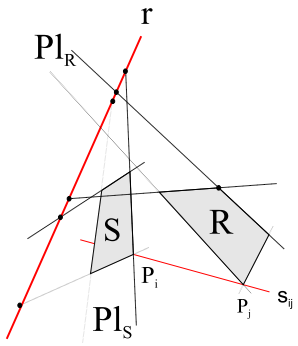


Figure 5: *Initial query polygons configuration. The line r is the intersection of planes Pl_S and Pl_R. All the edges of S and R are incident on r.*

The edges of the query polygons are mapped to hyperplanes in Plücker space, and form a projective polytope. After its projection on an arbitrary projection hyperplane, the polytope becomes an unbounded 'pyramid'. The line r maps to the point r^*, which is the apex of the pyramid, since r is the line on which all bounding edges are incident (cf. Figures 5 and 6).

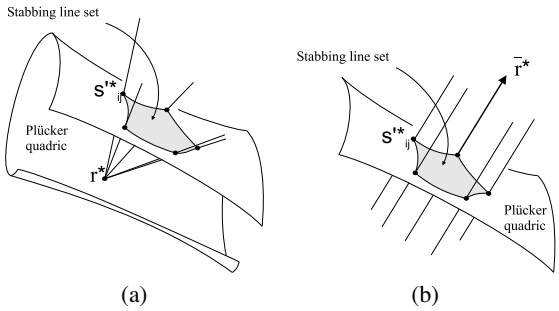


Figure 6: *Geometric properties of polytope $\mathcal{P}$. (a) After projection onto an arbitrary hyperplane: the apex of the pyramid r^* belongs to the quadric. (b) After projection onto $H_\perp$: r^* becomes a point at infinity. The intersection of the vertical edges with the Plücker quadric are the points s'^*_{ij} corresponding to the extremal stabbing lines. (Note: this is an evocation, the polytope is actually embedded in an hyperplane in $\mathbb{R}^6$. For clarity, we have represented only 5 of the 16 vertical edges that define the polytope in this configuration).*

The pyramid could possibly be degenerate if r^* became

a stabbing line, but this situation is avoided by the initial clipping of the query polygons. The set of stabbing lines between the query polygons map to points on the intersection of the pyramid with the Plücker quadric. The extremal stabbing lines are incident to four edges: they always contain a vertex P_i of S and a vertex P_j of R, and are noted s_{ij}. In projective space, they map to the rays s^*_{ij} on the intersection of the edges of the pyramid with the Plücker quadric, and become the points s'^*_{ij} after the projection (see Figure 6 (a)).

The split algorithm in Section 3.1 requires the polytope to be bounded, and we use the following steps to transform the projective polytope into a bounded polytope. First, we take a particular hyperplane $H_\perp$, chosen so that the point r^* becomes a point at infinity, corresponding to the direction $\overline{r^*}$ (the details can be found in the Appendix A). The projection onto $H_\perp$ effectively transforms the projective polytope into a prism with principal axis $\overline{r^*}$ and edges parallel to the direction $\overline{r^*}$. These edges are the *vertical* edges of the prism (see Figure 6 (b)). It is easier to clip this prism than to clip an unbounded pyramid obtained with an arbitrary projection hyperplane. Indeed, two capping hyperplanes H_c^+ and H_c^-, with normals $\overline{r^*}$ and $\overline{-r^*}$, are enough to obtain a closed polytope (see Figure 7). The independent terms of the capping hyperplanes, fixing their translation, are chosen so that they completely enclose the region of the Plücker quadric located inside the prism (the details are given below).

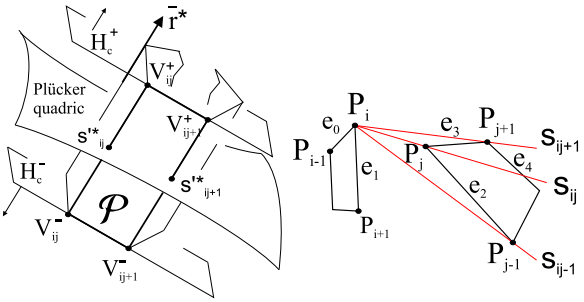


Figure 7: *On the capping hyperplanes H_c^+ and H_c^-, the two vertices V_{ij}^+ and V_{ij}^- created from s_{ij} share four hyperplanes with each other and with the vertices created from $\{s_{i+1j}, s_{i-1j}, s_{ij+1}, s_{ij-1}\}$.*

In Plücker space, a vertex of polytope $\mathcal{P}$ is located at the intersection of five hyperplanes (i.e. we suppose that the polytope is *simple* and that its vertices are contained in exactly five facets; this supposition is valid if S and R do not contain any degenerated edges). Four of these are the dual hyperplanes of the edges of the query polygons and the fifth is one of the capping hyperplanes. The vertex positions are computed as the intersections of the vertical edges of the prism and the capping hyperplanes. This can be efficiently done by forming the lines of the edges from the dual points s^*_{ij} of the extremal stabbing lines and the direction $\overline{r^*}$.
An edge is located at the intersection of four hyperplanes meaning that the edges of the skeleton should be created between pairs of vertices sharing four hyperplanes (see Figure 7).
The complete algorithm to create the 1-skeleton of $\mathcal{P}$ is as follows: compute the direction $\overline{r}^*$ and all the extremal stabbing lines s_{ij}, as well as their dual point s^*_{ij}. Project these points onto the projection hyperplane to obtain s'^*_{ij}. You are now able to find the capping hyperplanes H_c^+ and H_c^-: their normals are $\overline{r^*}$ and $\overline{-r^*}$ and their independent terms are computed so that all the s'^*_{ij} are inside the delimited polytope. Let d_{ij} be the orthogonal projection of s'^*_{ij} onto the direction $\overline{r^*}$, computed with the classical dot product in $\mathbb{R}^6$: $d_{ij} = < s'^*_{ij}, \overline{r^*} >$. The independent terms are the minimum and maximum values of all d_{ij}. To ensure the polytope contains all the stabbing lines, they must be clamped to 0 (the mathematical details are out of the scope of this paper, but this clamping enables to take the curvature of the Plücker quadric into account).

Then compute all the vertices V_{ij}^+ and V_{ij}^- as the intersections of the vertical edges passing through the points s'^*_{ij} with hyperplanes H_c^+ and H_c^-. The last step is to create the edges. For each extremal stabbing line s_{ij}, connect:

- vertex V_{ij}^+ and vertex V_{ij}^- (to form a vertical edge)
- vertex V_{ij}^+ and vertices V_{i-1j}^+,V_{i+1j}^+,V_{ij-1}^+,V_{ij+1}^+.
- vertex V_{ij}^- and vertices V_{i-1j}^-,V_{i+1j}^-,V_{ij-1}^-,V_{ij+1}^-.

4. Fast and Simple Polygon-Polygon Occlusion Query

A polygon-polygon occlusion query consists of determining whether the polygons S and R are visible through a set of convex polygonal occluders O_i. After the construction of the polytope $\mathcal{P}$ representing the set of lines stabbing S and R, the set of lines blocked by each occluder are subtracted incrementally, and the resulting polytopes are stored in the complex $\mathcal{C}$. Previous methods [MAM05] treat each polygonal occluder O_i individually and use every edge of every polygonal occluder as a splitting hyperplane, causing a lot of extra work. Furthermore, they perform poorly when the two polygons are partially visible because the visibility is only established after all the occluders have been subtracted. In this section, a new framework, using the low level algorithms presented in Section 3, is proposed to cope with these problems. We first present the two ideas it is based on: a new occluder selection process guided by visibility (Section 4.1), and a new from-region silhouette occluder aggregation (Section 4.2). The framework combining these ideas is presented in Section 4.3 and discussed in Section 4.4.

4.1. Visibility guided occluder selection

During a query, the polytope $\mathcal{P}$ is split by the hyperplanes of the occluder edges, and the polytopes corresponding to blocked line-space regions are eliminated. Each remaining

polytope represents a subset of the initial stabbing lines that are not blocked by the previously processed occluders. In other words, any line of this subset passes through an aperture left by the occluders (see Figure 8).

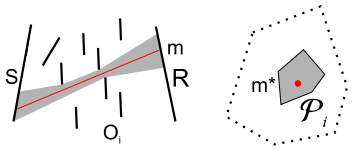


Figure 8: *$\mathcal{P}_i$ represents a set of lines through the aperture left by the occluders. m is the representative line of $\mathcal{P}_i$.*

For each polytope $\mathcal{P}_i$, such a line m is extracted and intersected with the geometry. If m does not intersect any occluders, S and R are mutually visible and the query stops. Otherwise, the next occluder O_i to be subtracted from $\mathcal{P}_i$ is selected among the polygons intersected by m. Once O_i has been chosen, its polytope $\mathcal{O}_i$, however, will not be subtracted from every polytope in the complex as was done in previous work [NBG02]. Instead, $\mathcal{O}_i$ is only subtracted from the polytope $\mathcal{P}_i$. Since the occluder was chosen by intersecting the representative line m with the geometry, this guarantees that $\mathcal{O}_i$ contains at least the point m^* of $\mathcal{P}_i$, and that the intersection of $\mathcal{P}_i$ and $\mathcal{O}_i$ is never empty (see Figure 9). Furthermore, every subtraction operation alters the intersection of the polytope $\mathcal{P}$ with the Plücker quadric, and all splits which would not remove any real stabbing lines are avoided. In constrast to the method proposed recently by Mora et al. [MAM05], the advantage of this occluder selection mechanism is that the unnecessary split operations are discarded *before* being performed.

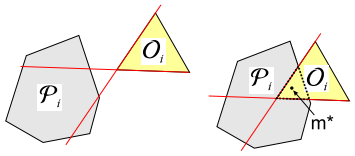


Figure 9: *(a) Even if the hyperplanes of $\mathcal{O}_i$ intersect $\mathcal{P}_i$, these two splits are redundant because $\mathcal{O}_i \cap \mathcal{P}_i$ is empty. (b) With our ray sampling strategy, $\mathcal{O}_i \cap \mathcal{P}_i$ is never empty, and the intersection contains at least m^*.*

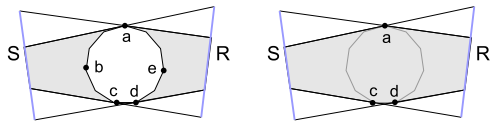
The representative line m of the polytope $\mathcal{P}_i$ is computed in Plücker space, where the polytopes are convex sets, rather than in $\mathbb{E}^3$, where they often represent complicated sets of lines (these line sets are delimited by swath surfaces that are not necessarily planar nor convex). In dual space, the point m^* must respect the following constraints:

- located inside $\mathcal{P}_i$: every linear convex combination of its vertices defines a valid point.
- belongs to the Plücker quadric.

To find m^*, we compute two convex linear combinations of vertices on each side of the Plücker quadric that together define a line segment. The intersection of this line segment with the Plücker quadric is the point m^*. If it is not possible to find a point for both sides of the quadric, an extremal stabbing line is taken as a representative line (this happens in degenerate configurations, when all the polytope' vertices are on the same side of the quadric or belong to the quadric). The problem of selecting one occluder among the occluders intersected by the line m is solved by using a line counting strategy, like the one used in the case of random ray sampling in [NBG02]. Each occluder maintains a counter representing the number of representative lines it intersects, and the chosen occluder is the one that intersects the most lines. For a given polytope $\mathcal{P}_i$, only a subset of the representative lines is taken into account: the ones having a dual point inside $\mathcal{P}_i$. Of course, every time a polytope is deleted, its representative line is also discarded. It is possible to reduce the number of line-geometry intersection tests by 'recycling' the representative lines: each split creates two sub-polytopes, and the line m can be used again as a representative line for one of them (i.e. for the sub-polytope located on the same side of the splitting hyperplane as m^*).

4.2. Silhouette occluder aggregation

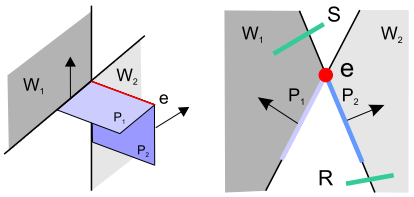
We define the *from-region silhouette* as the set of edges that are from-point silhouette edges simultaneously for at least one point of the polygon S and one point of the polygon R. In the case of connected polygonal occluders, only the from-region silhouette edges of the objects can cause visibility events (i.e. separate the Plücker space between blocked and free set of lines). The other ones, called the *internal edges*, are redundant for the visibility determination (see Figure 10).



(a) (b)

Figure 10: *(a) The shaded region represents the set of rays blocked by the occluders. b and e are internal edges, while a, c and d are from-region silhouette edges. (b) Only the silhouette edges have to be used to obtain the set of blocked rays.*

Rather than splitting the complex of polytopes by all the edges of every occluder, it is possible to avoid many redundant splits by verifying that the splitting edge is effectively a from-region silhouette edge. Let e be the edge that connects the occluder polygons P_1 and P_2. The planes containing P_1 and P_2 define a double wedge which delimits the regions W_1 and W_2 (see Figure 11).



(a) (b)

Figure 11: *The edge e connects the occluder polygons P_1 and P_2. The planes containing P_1 and P_2 define a double wedge region. (a) 3D view. (b) 2D view (e is perpendicular to the sheet of paper).*

The wedge of the from-region silhouette edges have an intersection with both query polygons S and R. We deduce the from-region silhouette edge condition (see Figure 12 for some examples):

From-region silhouette edge. *e is a from-region silhouette edge if and only if S has at least one point in W_1 (resp. in W_2) and R have at least one point in W_2 (resp. in W_1).*

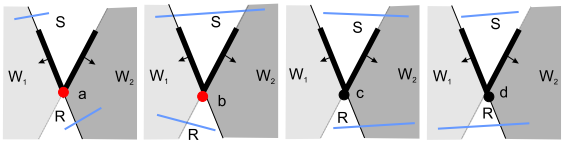
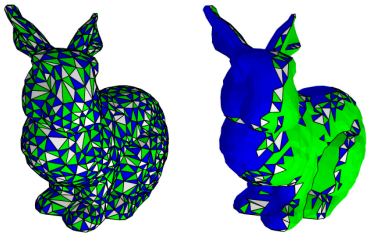


Figure 12: *Illustrative example of the silhouette condition. The edges a and b belong to the from-region silhouette, contrary to the edges c and d.*

The from-region silhouette edges partition the occluder mesh into connected patches F_i (i.e. each occluder polygon belongs to one and only one F_i, see Figure 13).



(a) (b)

Figure 13: *Using a connected polygonal mesh as an occluder for a query (S and R are not shown, but they are located on both sides of each bunny) (a) Classical approach: the polygonal primitives are treated individually and all the edges are used. (b) Silhouette occluder aggregation: the polygons are grouped into patches, and only the from-region silhouette edges are used .*

In our line counting mechanism used for occluder selection, each patch is considered an occluder. Since the silhouettes depend on the configuration of the query polygons, the patches F_i are explicitly extracted for each query the first time a polygon occluder is hit by a ray. An extraction is a flood-fill traversal of the initial mesh's face-adjacency graph that stops when from-region silhouette edges are met.

4.3. Framework for analytic occlusion query

Our framework builds on the described new techniques and can be summarized by the following pseudo-code:

```
1:  Procedure areVisible(S, R)
2:  Construct the initial polytope P for S and R
3:  return recursiveSplit(P)
4:
5:  Procedure recursiveSplit (P)
6:  m = representative line of P
7:  X = set of patches intersected by m
8:  if X = empty then
9:     return Visible //early termination
10: Select a patch F_i from X
11: Search for a valid edge e in F_i
12: if e ≠ empty then
13:    {P⁻,P⁺} = Split (P, H_e)
14:    if (recursiveSplit(P⁻)=Visible then
15:       return Visible
16:    if (recursiveSplit(P⁺)=Visible then
17:       return Visible
18: return Hidden
```

The complex $\mathcal{C}$ is not represented explicitly, but implicitly by successive calls to the recursive procedure *recursiveSplit($\mathcal{P}$)*. For each polytope, a representative line m is extracted and intersected with the scene geometry to select F_i (lines 6 and 7). As described in Section 4.1, m passes through an aperture remaining after the already processed occluders. If m does not intersect any occluders, the query polygons are mutually visible and the query terminates early (line 9). Otherwise the patch F_i is selected from the set of patches intersected by m (line 10). Its boundary edges are inspected, until an edge e verifying the following *split condition* is found (line 11):

$$\begin{cases} H_e, \text{ the dual hyperplane of } e, \text{ is not a facet of } \mathcal{P} \\ e \text{ belongs to the from-region silhouette (Section 4.2)} \\ e \text{ divides the set of lines inside polytope } \mathcal{P} \end{cases}$$

The last test is not trivial, and we evaluate it conservatively by testing the edge e for intersection with the convex hull of S and R. If no such edge is found, the current polytope represents a set of lines completely blocked by the patch F_i: the recursion stops, and the polytope is eliminated from the complex. Otherwise, the polytope is split into the polytopes $\mathcal{P}^-$ and $\mathcal{P}^+$ (line 13) and the procedure is recursively

applied to both polytopes (lines 14-17). The recursive procedure returns when all the polytopes have been blocked: S and R are mutually occluded (line 18).

4.4. Framework Analysis

The main advantage of our framework is that all splits by internal edges are avoided. The best case is a single convex occluder blocking all the rays between the query polygons: all the edges are internal and the query is proven hidden without performing any split operations. In a sense, the framework can be seen as an extension to the general case of multiple occluders of the predicates presented by Navazo et al. to detect the occlusion created by a single polygonal occluder [NRJS03].

In terms of occlusion layers [KS00], previous analytic frameworks were only able to fuse the occlusion for the second or greater occluded layer. This is achieved by using hidden cells as virtual occluders [NBG02]. In addition, our occluder aggregation method is able to combine occlusion from the *first visible layer* of occluders: this property is central for the scalability of the approach in scenes containing complex objects.

The possibility of early termination is increased greatly in the case of mutual visibility, since a representative line m is tested progressively for each aperture. The algorithm effectively converges on aperture and terminates. Furthermore, the complexity of the method is no longer a function of the occluders within the polygon-polygon query shaft, but a function of the complexity of the far simpler silhouette edge. Furthermore, the conjunction of the silhouette condition and the representative line strategy leads to a very efficient aperture detection, because the lines are guided towards the silhouette boundaries of objects.

5. Results

We have implemented the described algorithms in the 'VisiLib' library; the test computer is a laptop Pentium 4 computer (1.9Ghz) with 1.28 Gb of memory.

5.1. Framework evaluation

The most direct application of the polygon-to-polygon query framework is the computation of a PVS. However, the complete visibility pre-processing of a scene requires additional algorithms that are out of the scope of this paper (see for example [Lai05]). For this reason we have evaluated the query framework in context of PVS computation by using a random sampling process.

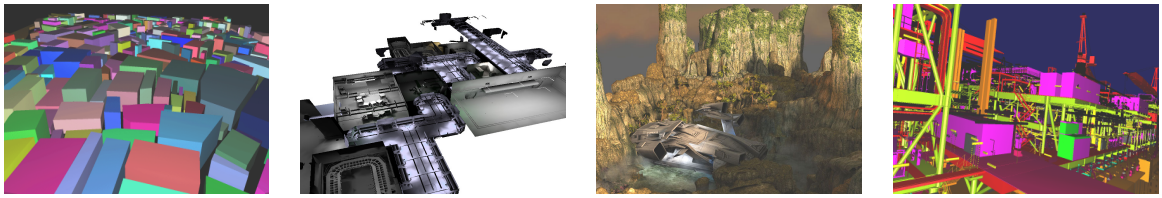
The framework was tested by placing axis-aligned cubical view cells of equal size along a path traversing the scene geometry. For realistic results, scenes of varying complexity were used. For each path the mutual occlusion of 100000 randomly chosen pairs of bounding boxes were tested, the first box corresponding to one of the viewcells and the other one enclosing an object of the scene. At first, the two boxes were replaced by a conservative polygonal approximation representing the union of all the possible views of the box from the other one, and a polygon-to-polygon query was applied to determine if they were hidden. If so, the boxes were also proven hidden. Otherwise, the mutual visibility of the 36 pairs of box faces were tested, and the boxes were marked hidden if all the face combinations were mutually hidden. For efficient processing, the scenes were stored in an octree and shaft culling was used to limit the intersection tests to the objects inside the shaft of the two bounding boxes [HW91]. The results are summarized in Figure 14. Because of the early termination mechanism, the distributions of the visible and hidden queries are slightly different and they are presented separately.

Each query begins with the creation of the initial polytope $\mathcal{P}$. Typically, when S and R are two quadrilaterals, the double description method implementation [Fuk] used in previous work [NBG02] takes about $8 * 10^{-3}$ sec. (this time also includes the computation of the full face-lattice of $\mathcal{P}$), and was a serious bottleneck (the situation is even worse in the case of [Bit02], because the method is used for each subtracted occluder). In the same situation, our direct construction method is more than 300 times faster, and performs in about $24.5 * 10^{-6}$ sec.

Depending on the scene, the average time for silhouette extraction ranges from 1 to 36 ms. Even if it is small in comparison to the time that would have been needed to perform all the split operations, it can reach up to 95% of the total query time in worst case situations. A caching scheme similar to the one proposed by Aila et al. [AM04] should be used to limit the cost of the extractions (optimization not implemented).

To identify the bottlenecks of the framework, we measured the average time spent in different parts of the algorithm. The results are presented in Figure 17(a), as a function of the number of occluders effectively subtracted during the query (i.e. the number of *effective occluders*). As expected, the most costly operations are the CSG in Plücker space and the representative line intersection tests. The latter are considerably more expensive than simple ray casting because the intersection testing does not stop when the first object is hit. With an average of 4000 representative lines treated by second, our implementation is considerably slower than the current standard in ray tracing and should certainly be improved. Another possibility could be to stop the intersection tests when a given number of occluders has been intersected. Note that the number of representative lines was already reduced by 'recycling' them among the polytopes created in splitting (Section 4.1).

For comparison purposes, we have also implemented the exact polygon-to-polygon occlusion query method presented in [NBG02], modified to reduce the fragmentation by detecting and discarding the useless splits as described in [MAM05]. To allow a direct comparison, this method



(a)Vienne (26k tri) (b) Level 12 (104k tri) (c) Level 1 (187k tri) (d) Bonga (4.6Mo tri)

		Visible	Visible	Visible	Visible	Visible	Hidden	Hidden	Hidden	Hidden	Hidden
	Visible (Time)	# Splits	# Rep. Lines	#Eff. Occ.	Sil. Ex.(ms)	Time (ms)	# Splits	# Rep. Lines	#Eff. Occ.	Sil. Ex.(ms)	Time (ms)
a	8%(5.3%)	0/7.5/169	1/7.5/151	0/8.1/148	0/0.2/22.8	0.2/4.3/112.7	0/5.1/162	1/6.5/138	1/8.8/122	0/1.1/44.2	0.3/6.6/106.7
b	16%(5.6%)	0/19.3/2.4k	1/11.8/1.2k	0/16.8/642	0/0.2/21.7	0.2/10.7/1.5k	0/51.8/6.7k	1/32.6/3.6k	1/55.5/1.1k	0/1.5/78.3	0.3/34.5/5.4k
c	46.6%(32.3%)	0/70.1/42.9k	1/34.7/20k	0/55.4/5k	0/1.3/394.4	0.3/62/56.4k	0/117.7/20.3k	1/60.6/9.8k	0/88.9/3k	0/2.8/335.3	0.4/113.3/30.5k
d	22.3%(17.7%)	0/17.7/6.9k	1/11/3.6k	0/14.7/3k	0/17.2/7.9k	0.6/80.5/24.5k	0/15.2/15.4k	1/8.9/8.1k	0/11.6/2.9k	0/36/10.4k	0.6/107.6/35.4k

Figure 14: *Vienna is a town model of 458 objects and 26k triangles; Level 12 and Level 1 are two computer game scenes: Level 12 is an interior scene of 1177 objects and 104k triangles, and Level 1 is an outdoor scene with 2160 objects and 187k triangles. Bonga is an industrial CAD model, counting 430k objects and 4.6 millions of triangles. For each scene, 100 000 random box-to-box occlusion queries where performed. For each data, the minimum/average/maximum value observed is given. The first column gives the percentage of visible queries and the global time spent to perform them (in parenthesis). The 3 next columns give resp. the number of splits, of representative lines, and of effective occluders performed during the query. The column 'Sil. Ex.' gives the time spent in the silhouette extraction process. Finally, the total time needed to perform one polygon-to-polygon occlusion query is given in the column 'Time'.*

was implemented using the low dimensional algorithms presented in Section 3 and the occluder selection presented in the Section 4.1. Despite all the modifications made to the original method, we refer to this implementation as the *classical framework*. In addition to the occluder silhouette aggregation technique, the main difference between the classical framework and the framework presented in the Section 4.3 is that once an occluder has been selected, all the polytopes of the complex are split by all the edges of its dual polytope. Time needed to perform one query, as a function of the number of effective occluders, is given in Figure 15 (Level 12 scene).

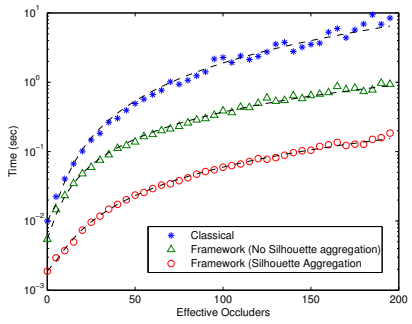


Figure 15: *Comparison of different methods for Level 12.*

For sufficiently large queries, the classical framework results are compatible to those presented in [NBG02]. However, our modified implementation is faster for smaller queries because it benefits from the acceleration of the initial polytope creation algorithm. With a least square fitting, we evaluated the complexity of the classical framework curve: $O(n^{1.84})$, where n is the number of effective occluders. The second curve is for our framework presented in Section 4.3 used *without* the silhouette occluder aggregation (i.e. all the edges are considered as splitting edges). In comparison to the classical framework, the occluders are only used locally, and the useless splits are avoided before being performed. For 100 effective occluders, the framework is about 5 times faster. The last curve is for our framework used *with* the silhouette occluder aggregation. Since most of the splits by internal edges are avoided, it provides a nearly constant acceleration over the precedent curve, equaling to about 6. For 100 effective occluders, the acceleration between the classical framework and the silhouette aggregation framework is about 30. The complexity of the silhouette occluder aggregation curve is $O(n^{1.44})$.

5.2. Occluder selection and aperture detection

To study the occluder selection and the aperture detection a synthetic scene was used to control the test parameters. The query polygons S and R are two equilateral triangles of the same size, parallel to each other while random sized equilateral triangles are incrementally inserted between them. After inserting each occluder, the mutual visibility of S and R was queried (see Figure 16).
Since the occluders are disconnected, no silhouette occluder aggregation occurs. The results are presented in the Figure 17(b) (average over 1000 experiments). The curves are decomposed between a visible and a hidden phase. During the visible phase, they increase with the number of trian-

gles: the query polygons get less visible, and early termination happens later as the size of the aperture decreases. The curves reach a maximum point when the shaft gets blocked (with 310 occluders on average). After this, adding more triangles gives more choices for the occluder selection heuristic, and the curves begin to decrease. This shows that the occlusion query cost is not proportional to the number of occluders present in the shaft: as soon as the shaft is blocked, adding more occluders can even have a positive effect on the computation time (except for the representative line intersection tests that are more and more costly in our implementation).

In previous work, early termination was obtained by testing if free rays exist among a set of randomly chosen rays between S and R [NBG02]. If the random sampling missed the aperture, all the occluders inside the shaft had to be treated before the mutual visibility of the polygons could be established. The visible queries were potentially more costly than the hidden ones that only subtracted the subset of effective occluders needed to prove the occlusion. Let p be the probability of detecting an aperture by using random rays. Figure 17(c) represents $\frac{1}{p}$, the average number of random rays needed to discover the aperture, estimated by casting a large number of random rays. This number becomes very large as the aperture size decreases, while our visibility guided aperture detection always determines the mutual visibility with a bounded number of representative lines. Using our framework, the need to subtract all the occluders inside the shaft to prove visibility is avoided.

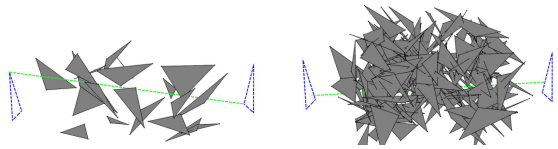


Figure 16: *Random occluders experiment: random sized equilateral triangles are iteratively inserted between the query polygons.*

6. Conclusion and Future Work

We have presented a new framework for polygon-to-polygon occlusion query that decreases the average complexity of the previous analytic approaches on realistic scenes. Maintaining the full face-lattice of the polytopes, as done in previous work, is similar to the construction of a localized subset of the visibility complex. Our approach, using only the 1-skeletons of the polytopes, is similar to the construction of a localized subset of the visibility skeleton. The first benefit of this framework is to reduce the complexity of the query by choosing the occluder locally for each polytope: the useless splits are avoided. Furthermore, the chance of early termination is greatly enhanced in the case of mutual visibility. When the framework is used to perform the silhouette occluder aggregation, it becomes sensitive to the complexity of the from-region silhouette of each occluding object, rather than to its entire polygonal mesh. The drawbacks are linked to the extraction of the silhouettes: the triangle adjacencies have to be stored for each mesh, and the silhouettes must be extracted for each occluder for each query. However, the time needed to perform this task is small in comparison to the time that would be needed to perform all the split operations. The algorithm evaluates a silhouette condition and a split condition for each occluder edge. These tests are not trivial, and were replaced by simpler conservative tests in our implementation, leading to some amount of useless split operations. As future work, the from-region silhouette could be extracted locally for each polytope, instead of globally as it was presented here. We also plan to develop an accurate test to determine if e effectively divides the set of lines inside each polytope to replace our conservative test using the convex hull of S and R.

Since occluder selection is an NP-hard problem, our framework uses a representative line counting strategy as a selection heuristic. This greedy process does not always provide the best ordering [NBG02], and we plan to improve this heuristic in future work.

Finally, we intend to incorporate the polygon-to-polygon mechanism in a complete PVS computation algorithm, such as the one presented recently in [Lai05].

Acknowledgements We would like to thank T. Aila, T. Karras, T. Haanpää, C. Thays, X. Baele, C. Laugerotte and O. Debeir for all their help. This work is partially supported by a grant of the 'Région wallonne' (Belgium).

References

[AF96] AVIS D., FUKUDA K.: Reverse search for enumeration. *Discrete Applied Mathematics 65*, 1-3 (1996), 21–46. 4

[AM04] AILA T., MIETTINEN V.: dpvs: An occlusion culling system for massive dynamic environments. *IEEE Computer Graphics & Applications* (2004), 86–97. 9

[ARB90] AIREY J. M., ROHLF J. H., BROOKS, JR. F. P.: Towards image realism with interactive update rates in complex virtual building environments. In *Proc. of the Symp. on Interactive 3D graphics* (March 1990), pp. 41–50. 1

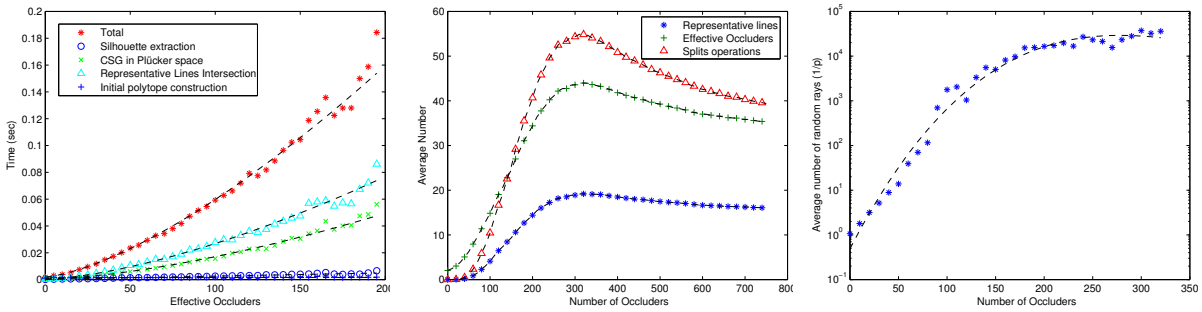
[BHS98] BITTNER J., HAVRAN V., SLAVÍK P.: Hierarchical visibility culling with occlusion trees. In *Proc. of Computer Graphics International* (June 1998), pp. 207–219. 3

[Bit02] BITTNER J.: *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University in Prague, 2002. 1, 3, 4, 5, 9

[BP96] BAJAJ C. L., PASCUCCI V.: Splitting a complex of convex polytopes in any dimension. In *Proc. of SoCG* (1996), pp. 88–97. 4

[BWW01] BITTNER J., WONKA P., WIMMER M.: Visibility preprocessing for urban scenes using line space subdivision. In *Proc. of Pacific Graphics* (2001), pp. 276–284. 1, 3

[COCSD02] COHEN-OR D., CHRYSANTHOU Y., SILVA C. T.,



(a) (b) (c)

Figure 17: *(a) Silhouette occluder aggregation: time for the different algorithm steps vs. number of* effective *occluders (for the Level 12) b) Number of representative lines, splits and effective occluders for the random occluders experiment (c)* $\frac{1}{p}$*, the average number of random rays needed to discover the aperture, in function of the number of occluders for the random occluders experiment.*

DURAND F.: A survey of visibility for walkthrough applications. *IEEE Transaction on Visualization and Computer Graphics 9* (2002), 412–431. 1

[DDP96] DURAND F., DRETTAKIS G., PUECH C.: The 3d visibility complex, a new approach to the problems of accurate visibility. In *Proceedings of Eurographics Workshop on Rendering* (June 1996), pp. 245–257. 4

[DDP97] DURAND F., DRETTAKIS G., PUECH C.: The visibility skeleton: a powerful and efficient multi-purpose global visibility tool. In *Proc. of Siggraph* (1997), pp. 89–100. 4

[FR94] FUKUDA K., ROSTA V.: Combinatorial face enumeration in convex polytopes. *Computational Geometry 4* (1994), 191–198. 4

[Fuk] FUKUDA K.: cdd package. http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html. 9

[HW91] HAINES E., WALLACE J. R.: Shaft culling for efficient ray-cast radiosity. In *Proc. of Eurographics Rendering Workshop* (1991). 9

[KCoC01] KOLTUN V., COHEN-OR D., CHRYSANTHOU Y.: Hardware-accelerated from-region visibility using a dual ray space. In *Proc. of Eurographics Rendering Workshop* (2001), pp. 205 – 216. 1, 3

[KS00] KLOSOWSKI J. T., SILVA C. T.: The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 108–123. 9

[Lai05] LAINE S.: A general algorithm for output-sensitive visibility preprocessing. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005), ACM Press, pp. 31–39. 9, 11

[LSCO03] LEYVAND T., SORKINE O., COHEN-OR D.: Ray space factorization for from-region visibility. In *Proc. of Siggraph* (2003), pp. 595–604. 1

[MAM05] MORA F., AVENEAU L., MÉRIAUX M.: Coherent and exact polygon-to-polygon visibility. In *Proceedings of Winter School on Computer Graphics* (2005). 4, 5, 6, 7, 9

[NB04] NIRENSTEIN S., BLAKE E.: Hardware accelerated aggressive visibility preprocessing using adaptive sampling. In *Proc. of the Eurographics Symposium on Rendering* (2004), pp. 207–216. 1

[NBG02] NIRENSTEIN S., BLAKE E., GAIN J.: Exact from-region visibility culling. In *Proc. of Eurographics Rendering Workshop* (2002). 1, 3, 4, 5, 7, 9, 10, 11

[Nir03] NIRENSTEIN S.: *Fast and accurate visibility preprocessing*. PhD thesis, University of Cape Town, 2003. 3, 4, 5, 12

[NRJS03] NAVAZO I., ROSSIGNAC J., JOU J., SHARIFF R.: Shieldtester: Cell-to-cell visibility test for surface occluders. *Computer Graphics Forum* (2003), 291–302. 9

[PT02] PANTAZOPOULOS I., TZAFESTAS S.: Occlusion culling algorithms: A comprehensive survey. *Journal of Intelligent and Robotic Systems* (2002), 123–156. 1

[Pu98] PU F.-T.: *Data structures for global illumination computation and visibility queries in 3-space*. PhD thesis, University of Maryland, 1998. 3

[Tel92] TELLER S.: Computing the antipenumbra of an area light source. In *Proc. of the Symp. on Interactive 3D graphics* (1992), pp. 139 – 148. 3

Appendix A: Transforming r^* into a point at infinity

Let $r^*(r_0^*,...,r_5^*)$ be a point in $\mathcal{P}^5$ and H a projection hyperplane of equation $\sum_{i=0}^5 a_i x_i = b$. The point r^* can be seen as a ray $(tr_0^*,...,tr_5^*)$ going through the origin in $\mathbb{R}^6$. The projection of r^* onto the hyperplane H is equal to the intersection point of this ray with H, obtained when t is equal to $t_{int} = -b/(\sum_{i=0}^5 a_i r_i^*)$. We are free to choose the projection hyperplane. In order to project r^* to a point at infinity, we take the projection hyperplane of equation $H_\perp \equiv \pi_3^r x_0 + \pi_4^r x_1 + \pi_5^r x_2 + \pi_0^r x_3 + \pi_1^r x_4 + \pi_2^r x_5 = 1$. Since r is a real line, r^* belongs to the Plücker quadric, and we have $\sum_{i=0}^5 a_i r_i^* = H_r(r^*) = 0$ (by equation (3)): with this projection hyperplane, r^* is effectively projected to a point at infinity. A similar projection was also used in [Nir03], but the scene had to be rotated for each query to ensure that the line r was included in the plane $yz = 0$; our choice of projection hyperplane avoid this rotation step.