Coherent and Exact Polygon-to-Polygon Visibility

F. Mora, L. Aveneau, M. Mériaux
SIC, CNRS FRE 2731, SP2MI
Bd Marie et Pierre Curie, BP 30179
80962 Futuroscope Chasseneuil Cedex – France
{mora,aveneau,meriaux}@sic.univ-poitiers.fr

ABSTRACT

Visibility computation is a classical problem in computer graphics. A wide variety of algorithms provides solutions with a different accuracy. However, the four dimensional nature of the 3D visibility has prevented for a long time from leading to exact from-polygon visibility algorithms. Recently, the two first tractable solutions were presented by Nirenstein, then Bittner. Their works give the opportunity to design exact visibility tools for applications that require a high level of accuracy. This paper presents an approach that takes advantage of both Nirenstein and Bittner methods. On the one hand, it relies on an optimisation of Nirenstein's algorithm that increases the visibility information coherence and the computation robustness. On the other hand, it provides an exact visibility data structure as Bittner does, but also suited for non-oriented polygon-to-polygon visibility queries.

Keywords

Exact visibility, CSG, Plücker space

1. INTRODUCTION

Visibility computation is a recurring problem in computer graphics applications. A wide variety of algorithms exists in the literature but they provide a different accuracy. This has led to a general algorithm classification :

- Aggressive : the visibility is underestimated.
- Conservative : the visibility is overestimated.
- Approximate : both aggressive and conservative.
- Exact : the visibility is exactly computed.

Solutions for these first three categories are usually fast. Most of them are designed in a context of visibility culling [Coh03a]. In contrast, exact algorithms require a significant computational effort, especially for from-polygon or from-region visibility. This problem has been considered for a long time as intractable due to the four dimensional nature of the visibility in 3D environment.

*C*onference proceedings ISBN 80-903100-7-9
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

Previous works attempt to compute a global and exact visibility information, as Pellegrini [Pel93a] or Durand [Dur02a] with the 3D visibility complex. But these solutions are not practicable.

The first tractable algorithm was recently published by Nirenstein [Nir02a]. It allows an exact computation of the visibility from a polygon. At the same time, Bittner[Bit02c] proposed another solution.

The exact visibility is not necessary for most application. However, it has potential to improve high quality rendering of complex scenes or realistic lighting effects. The works of Nirenstein and Bittner give the opportunity to design efficient visibility tools encoding an exact information.

This paper presents an exact visibility algorithm that takes advantage of both Nirenstein and Bittner methods. It relies on Nirenstein algorithm but provides in output a structured visibility information as Bittner does. Moreover, it presents an optimisation of Nirenstein algorithm that improves the visibility information coherence. As a consequence, it gets a noticeable property : By reducing the visibility result complexity, the number of performed operations decreases, improving the robustness.

The second section explains the mathematical underlying and the general approach used by Nirenstein and Bittner for exact visibility computation. The third one gives an overview of the two existing algorithms and underlines their differences. From this short study, the section four presents our approach emphasising our

optimisation that provides a coherent visibility information and improves robustness. At last, results are given in the section five.

2. BACKGROUND

The two solutions proposed by Nirenstein and Bittner both rely on the same approach. They solve the visibility problem between polygons by performing CSG operations on polytopes (convex "volume") in the Plücker space. This section begins with a presentation of the Plücker space where operates the solution. Next, it gives an overview of the approach allowing exact visibility computation from 3D polygons.

Plücker Space

The Plücker space [Som59a] is a five dimensional projective space $\mathbb{P}^5$. It provides an elegant parametrisation for dealing with directed lines in $\mathbb{R}^3$. Each line l passing through the point (p_x, p_y, p_z) and next through (q_x, q_y, q_z) is defined in $\mathbb{P}^5$ by $\pi_l = (\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$, with :

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

Notice that (π_0, π_1, π_2) is the direction of l and (π_3, π_4, π_5) encodes its location.
Next, let us consider the dual mapping within $\mathbb{P}^5$: Each $\pi \in \mathbb{P}^5$ can be associated with a dual hyperplane h_π defined by :

$$h_\pi = \{x \in \mathbb{P}^5 \quad | \quad \pi_3 x_0 + \pi_4 x_1 + \pi_5 x_2 + \pi_0 x_3 + \pi_1 x_4 + \pi_2 x_5 = 0\}$$

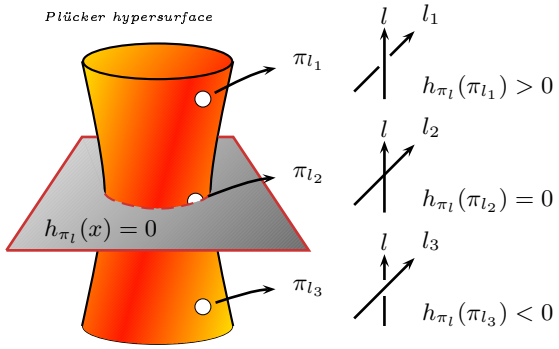

Figure 1: Line orientation in the Plücker space : There are three different cases for an oriented line to pass another : l_1 passes on the left of l_0, l_2 is incident on l_0 and l_3 passes l_0 on the right. The Plücker mapping of l_1, l_2, l_3 will respectively lie above, on and below the dual hyperplane of l_0.

Given two lines l_1 and l_2 and their Plücker mapping π_{l_1} and π_{l_2}, a crucial property is : l_1 and l_2 are incident if and only if π_{l_1} lies on the dual hyperplane of π_{l_2} (and vice versa). If $h_{\pi_{l_1}}(\pi_{l_2}) \neq 0$, the sign of $h_{\pi_{l_1}}(\pi_{l_2})$ determines the relative orientation of l_1 and l_2 as illustrated on figure 1.

At last, each line in $\mathbb{R}^3$ maps to a point in $\mathbb{P}^5$ but each point in $\mathbb{P}^5$ does not map to a line in $\mathbb{R}^3$. The mapping of all real lines in $\mathbb{P}^5$ forms a four-dimensional quadric surface called the *Plücker hypersurface*.

Exact From Polygon Visibility Principle

2.2.1 Lines stabbing polygons

Previous definitions are useful to characterise the set of lines stabbing convex polygons. In the Plücker space, these lines are a connected subset of points on the hypersurface. For computational convenience, it is easier to deal with a polyhedral representation of this subset by using the dual hyperplane mapping of each polygon edges. The intersection of this polyhedral structure with the Plücker hypersurface gives exactly the set of lines stabbing each polygons. Such an approach was already used by Teller [Tel92a] for computing the anti-penumbra of an area light source through a sequence of polygons.

Figure 2 illustrates a two triangles case since we are in a context of polygon to polygon visibility. More generally, if A and B are two polygons with n and m edges $e_1, ..., e_{n+m}$ consistently oriented, all the lines l passing through A then B satisfy :

$$\forall \, \mathrm{i} \in [1..n+m], \; h_{\pi_{e_i}}(\pi_l) \geq 0$$

This system of inequations is the hyperplane repre-

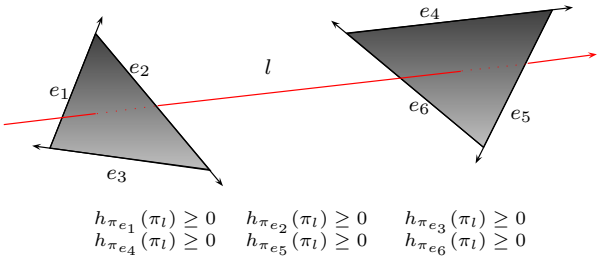

Figure 2: Lines stabbing two polygons : The Plücker mapping of the polygons edges induces the hyperplane representation of a polytope. Its intersection with the Plücker hypersurface is the set of all lines stabbing the two polygons.

sentation of an unbounded polyhedron in the Plücker Space. Both Nirenstein and Bittner add constraints to obtain a closed polyhedron : a polytope. Of course these additional constraints do not affect the intersection of the polyhedron with the Plücker hypersurface. The polytope representation allows to limit computations to the zone of the Plücker hypersurface.

2.2.2 Occluders removal

Let P_{AB} be the polytope that represents the set of lines stabbing A and B. Figure 3 gives a 2D illustration of the process that removes from P_{AB} the set of lines blocked by an occluder. This has to be applied to each occluder. The remaining parts of P_{AB} intersecting the hypersurface are exactly the set of lines that stabs A and B without stabbing any occluders. If no such a part remains, A and B are not visible.

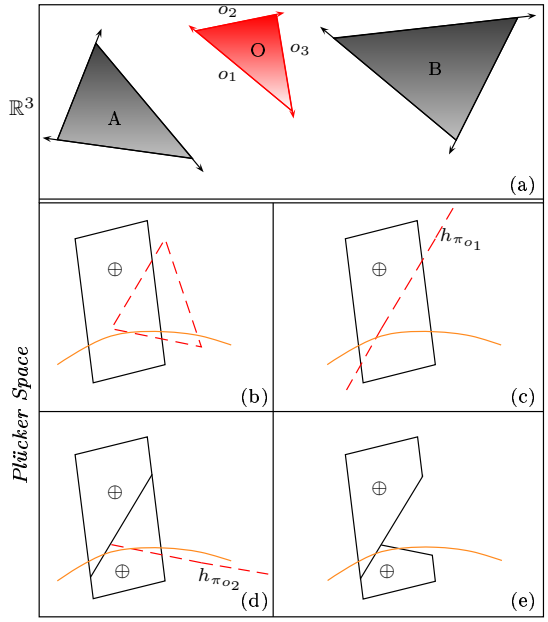

Figure 3: (a) An occluder O blocks some visibilities between two polygons A and B. (b) The Plücker representation of lines stabbing A and B and lines stabbing O. (c) (d) : To remove the subset of blocked lines, P_{AB} is successively split in sub-polytopes using the hyperplanes associated to the occluder edges. (e) The sub-polytope corresponding to blocked lines is removed.

3. EXISTING ALGORITHMS

Nirenstein and Bittner methods both rely on the approach explained in the previous section. However these two algorithms were not designed for providing the same visibility information. Moreover, they have noticeable differences between their CSG operations on polytopes.

Exact visibility requires a consequent computational effort, using non trivial n-dimensional geometric algorithms. An effective implementation of the process has to be carefully considered.

In this section, a short overview of each algorithm is given, including some comparisons on their processes. Then, we justify our choices from this study.

Algorithms overview

3.1.1 Nirenstein's algorithm

Nirenstein designed his algorithm to query if two polygons were visible or not. First, an initial polytope representing their stabbing lines is built. Next CSG operations are computed in the Plücker space to remove the lines blocked by each occluder. This is the same process as depicted by figure 3. Only the visible parts of the initial polytope are preserved during the process. However, this information is not organised and is only maintained as a set of sub-polytopes. As soon as the visibility or the invisibility is established, they are all dropped.

Exact visibility computation is sensitive to the number of occluders that have to be removed. Nirenstein makes a selection of the most effective occluders to be removed first. The occluded lines set can be removed using less occluders. This method minimises the number of intersection computation to perform. As a consequence, the algorithm termination is accelerated.

Nirenstein uses his algorithm to compute Potentially Visible Sets (PVS) for viewcells in 3D environment. In this context, he develops a framework including several optimisations that aim either to quickly find simple visibilities/invisibilities, or to choose an effective order for removing occluders. On the one hand, ray sampling handles trivial visibilities and finds effective occluders. On the other hand, a hierarchical subdivision of the scenes is used. Visibility queries are first applied to the cells of this hierarchy. Only visibility with polygons inside visible cells have to be computed, whereas invisible cells can be used as virtual occluders.

3.1.2 Bittner's algorithm

The purpose of Bittner's algorithm is different from Nirenstein's one. It was first developed in 2D [Bit01b] and then extended to three dimensional environments. It aims to encode all the visibilities with a scene from a source polygon. This information is encoded and structured by an occlusion tree [Bit98a]. Each leaf represents either a visibility or an invisibility set (when nothing can be seen). In particular, each in-leaf represents a set of lines that first stabs the same visible polygon.

The occlusion tree construction implies to treat occluders in a front to back order. This assumes the scene pre-processing to avoid overlapping occluders. For each of them, the associated polytope is inserted into the occlusion tree, from the root to the leaves, and is tested against each node met. If the hyperplane stored in a node splits the polytope, the algorithm continues in both subtree with the two relevant fragments. If an out-leaf is reached, the occluder is visible and the out-leaf is replaced by its fragment elementary occlusion

tree. If an in-leaves is reached, the fragment elementary occlusion tree is merged to update the visibility information.
Bittner also uses a hierarchical subdivision of the scene to enhance the occlusion tree construction. At the end of the process, the occlusion tree provides each part of the geometry that can be seen from the source polygon. Like Nirenstein, Bittner uses this information to compute PVS for viewcells. He also gives an example of virtual occluders extraction, valid from any viewpoint on the source polygon.

Algorithms Implementation

As explained, computing exact visibility implies an important computational effort. This can not be trivially implemented. Some computational differences can be noticed between Nirenstein and Bittner implementations :

Polytope construction

The hyperplane representation of a polytope can be easily obtained from the Plücker mapping of polygons edges. However the intersection tests require the vertex representation. In any case, this can be achieved using an enumeration algorithm as in [Avi96a]. Such an algorithm is used by Bittner. For two given polygons, Nirenstein proposes in his thesis [Nir03a] a more efficient solution, including explicitly the additional constraints to cap the unbounded polyhedron. In contrast, the capping of the polyhedron in Bittner's algorithm requires more computation tests.

Intersection tests

Before splitting a polytope, intersection tests are made with hyperplanes. A common test to both methods is to compute whether polytope vertices fall in both half spaces induced by a hyperplane. In addition, Nirenstein implementation first makes a conservative test using the bounding sphere of a polytope. Then, a rejection test is applied using the other hyperplanes of the same occluder, as detailed in the next section. Contrary to Bittner's algorithm, this allows to limit the intersection computation to the zone of occluded lines.

Polytope splitting

The key for occluders removal is to compute the intersection of a polytope with a hyperplane. The implementation of Bittner computes implicit intersections. This means that a splitting hyperplane is added to the hyperplanes representation of a polytope, and all the vertices are enumerated again.
Nirenstein computes explicitly intersections using an algorithm similar to Bajaj and Pascucci's one [Baj96a]. As a requirement to this algorithm, the full face lattice of the polytope has to be computed. Nirenstein uses a combinatorial face enumeration as in [Fuk94a]. This is potentially more efficient since only the new vertices are computed, whereas Bittner enumerates all the vertices again.

Algorithms discussion

Our purpose is to compute a coherent and exact visibility information between two polygons. This information has to be structured to be available from the two queried polygons. The more coherent the visibility information will be, the more efficient its use will be.
Bittner's algorithm is interesting because it provides a structured visibility information. However this information is oriented since it is significant from the source polygon. As a consequence, it is not suited for non-oriented polygon-to-polygon visibility queries. We can notice that the occlusion tree construction can be restricted between two polygons. But it would still encodes the occluder fragments only visible from one polygon.
The computational part of the Nirenstein algorithm seems to be more efficient. In particular, the different tests made to limit the computation to the zone of an occluded lines set are interesting for our purpose. This should improve the coherence of the visibility information computed between two polygons. Besides, his algorithm is more flexible than Bittner algorithm with its fixed "front to back" substraction order. As an example, the optimisation presented in this paper could not be applied to his algorithm without corrupting the output. This will be explained in the next section. Nirenstein algorithm can provide a set of polytopes representing all the visibility between two polygons. This information is non-oriented. However it is not structured like Bittner.
From this study, we choose to take advantage of the Nirenstein algorithm for CSG computation on polytopes. But the algorithm output is modified to organise the visibility information, like Bittner does. Our structure is suited for non-oriented polygon-to-polygon visibility. The next section presents our approach and an optimisation of the Nirenstein algorithm. In particular, this optimisation minimises the fragmentation of the visibility information, and improves robustness.

4. PROPOSED APPROACH

Firstly, this section explains how useless polytope fragmentation can occur. Next an overview of our approach is given and illustrates how the visibility information is encoded. At last, we presents the "back splitting" optimisation that allows to minimise the polytope fragmentation and to improve the robustness.

Unnecessary Splitting

Two configurations exist where unnecessary splitting are computed. The first one appears when the splitting of a polytope P results in a first polytope having the

same intersection as P with the Plücker hypersurface, and a second one having no intersection with the hypersurface. Since only the Plücker surface intersection is of interest, it becomes clear that such a splitting is useless. However, we will see how to take advantage of this problem.

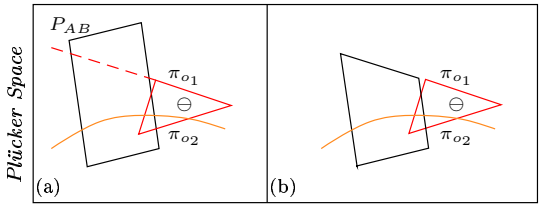

Figure 4: (a) A polytope P_{AB} that does not need to be intersected by each hyperplanes of an occluder. (b) As an example, the intersection with π_{o_1} does not modify the intersection of P_{AB} with the hypersurface. This would be the same with π_{o_2}.

The second configuration is within the rejection test of Nirenstein. It checks if a polytope is rejected by at least one hyperplane from a given occluder. This test, depicted in figure 5, can not always prevent unnecessary splitting operations. Figure 6 illustrates such a case. This explains why useless polytopes fragmentation still happens.

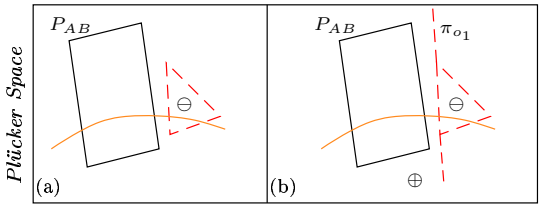

Figure 5: (a) A polytope P_{AB} that does not need to be intersected by any hyperplanes π_{o_1}, π_{o_2}, π_{o_3} of an occluder. (b) Because all the vertices of P_{AB} lie in the positive half space of π_{o_1}, P_{AB} will be rejected and unnecessary splitting will be avoided.

Useless splitting generates two main problems. Firstly, it seems obvious that the probability to face numerical instability grows with the number of successive splitting operations performed. Next it leads to an unnecessary polytopes fragmentation, and so to a fragmentation of the visibility information. These two problems are obviously correlated. As a consequence, reducing the polytopes fragmentation must be a solution to both of them.

Overview

Our approach uses a similar algorithm to Nirenstein's one. In this paper, notice we are not in a specific context. As a consequence, we do not use a framework as

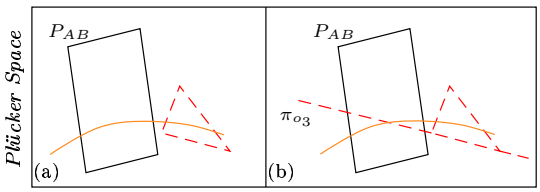

Figure 6: (a) Another configuration where P_{AB} does not have to be split. (b) Because each hyperplane π_{o_1}, π_{o_2}, π_{o_3} intersects P_{AB}, it will not be rejected. As a consequence, unnecessary splits will be perform. In particular the split by π_{o_3} will generate useless fragmentation of the visibility information.

developed by Nirenstein for PVS computation. Moreover some parts of this framework are out of matter. For example, ray sampling can not be used since we are interested in the whole visibility information computation.

We consider two polygons and obtain from their edges the hyperplanes representation for the associated polytope in the Plücker space. Our implementation takes advantage of the Nirenstein solution [Nir03a] for computing its vertices representation. Its full face lattice is obtained with [Kai02a] that provides a better complexity than [Fuk94a]. At last, explicit splitting operations are achieved using the approach of [Baj96a].

To store the visibility information, we build a "history tree". It has some similarities with an occlusion tree. It is a binary tree whose inner nodes are associated with splitting hyperplanes. A leaf represents either a set of blocked lines, or a set of visibilities between two polygons. In the later case, the polytope for this set is associated to the leaf.

But the history tree construction is different. It does not require a traversal from the root to the leaves. Intersection tests are made on visible leaves. When a leaf is split, it becomes an inner node associated with the splitting hyperplane. Its children represent each part of the intersected polytope. Initially, the root node is set with the polytope associated to the two queried polygons. A history tree can be understood as the history of the successive splitting operations.

At the end of the process, it encodes and represents all the visibility information between two polygons. The history tree is easy to build and does not require excessive computation time. This structure is suited to be used as a visibility tool.

Moreover, it gives the opportunity to minimise unnecessary splits. Since this induces an useless polygon fragmentation, we propose to detect and to cancel them using the history tree. This is the purpose of the "back splitting" algorithm. It also improves the visibility information coherence.

Back Splitting Algorithm

The back splitting optimisation takes advantage of the history tree to cancel useless operations. This implies the following modifications : Before splitting a polytope, its copy is left in its associated node of the history tree. An inner node is then associated with a polytope and a splitting hyperplane. Back splitting affects the construction of the history tree, with the combination of two rules.

The first one aims to reduce the number of splits to improve the robustness. It is applied during an occluder removal, after a splitting operation. If the intersection with the hypersurface has not been modified, this means the splitting was useless. However, to keep the operation benefit, the smaller polytope representing the same set of lines replaces the copy of the initial polytope. This may seem a contradiction to the robustness improvement. Our motivation is to work with polytopes as close as possible to the hypersurface, to improve further rejection tests, and so to decrease the number of splitting operations. The robustness is related to this number. Our tests have shown that keeping the smaller polytope gives finally a smaller splitting number than keeping the initial polytope.

The second rule tries to minimise the polytopes fragmentation. It has to be applied after each occluder removal. It relies on a quick analysis of each pair of leaves sharing the same father. This rule is applied each time one of the two following configuration occurs :

- If both leaves are visible, this means that the polytope set in the father node was unnecessary split, as shown in Figure 6(b). In this case, this operation is cancelled. Both children are removed and the father node restored as a visible leaf.

- If both leaves are invisible, children are removed and the father node is replaced by a leaf marked invisible. A similar implication was proposed by Bittner for its occlusion tree. However, due to the back to front order constraint, he could not apply the previous configuration.

This optimisation helps to minimise the polytopes fragmentation and the number of splitting operations. This may seem a contradiction since less fragmentation implies bigger polytopes and bigger polytopes is a contradiction to the first rule. However the justification is that a polytope can be “big” as long as it remains close to the Plücker hypersurface.

Moreover, this allows a smaller history tree that describes the same visibility information. As a consequence, we can expect a more efficient use of this information and to spare memory. Notice that all the polytopes associated with nodes can be removed after the construction of the history tree. The tree with the splitting hyperplanes in inner nodes is then sufficient.

In the next section, we present some experimental results emphasising the back splitting improvement. We test different configurations depending on the visual complexity.

5. RESULTS

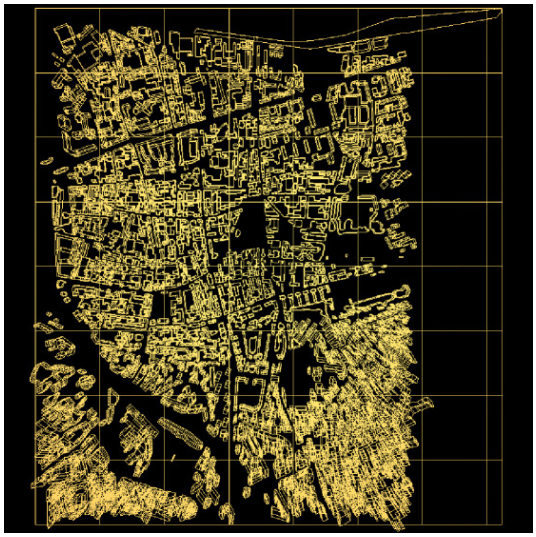

Figure 7: Test scene

To test the back splitting optimisation we use an urban environment composed by 38834 polygons as depicted in figure 7. This scene was chosen for the various configurations proposed in terms of occlusion and visual complexity. The hardware used is an Athlon XP1800+ (1.5 GHz) with 512Mo RAM.

From the test scene, we choose three sets of buildings, each one representing a different configuration for occlusion and visual complexity.

Set 1 The first set is composed of the ten smallest buildings in the scene. Here, the occlusion is strong, and the visual complexity is low.

Set 2 In opposition to the first set, the second set is composed of the ten highest buildings in the scene. This implies many visibilities and few occlusion.

Set 3 The third set contains ten buildings with an average height. It combines both visual complexity and depth visibility. This means that occlusion can not be defined by a small subset of occluders.

From each set, the exact visibility between each building wall and the other scene polygons are computed. Table 1 shows results for the three sets.

For the first set, the back splitting has no contribution. We can notice a small time over cost for a query using back splitting. The explanation lies in the fact that the

Set 1	AVG Occluders	AVG Polytopes	AVG Splitting	Time/query (ms)
Without BS	66.36	1.67	1.02	48.63
With BS	66.36	1.67	1.02	49.74
Set 2	AVG Occluders	AVG Polytopes	AVG Splitting	Time/query (ms)
Without BS	136.7	40.5	62.26	203.11
With BS	136.7	20.44	37.25	132.09
Set 3	AVG Occluders	AVG Polytopes	AVG Splitting	Time/query (ms)
Without BS	157.3	17.39	39.87	357.55
With BS	157.3	7.55	27.36	244.37

Table 1: Results without and with back splitting (BS). AVG Occluders is the average number of occluders per query. AVG Polytopes is the average number of polytopes representing the visibility information between two visible polygons. AVG Splitting is on average the number of splitting operation for each query.

(in)visibilities are quickly determined, before the two rules could have an impact on the computation.
For the second set, a significant contribution appears : 40% of the splitting operations are avoided. This leads to an improvement (35%) of the time computation per query. However, the most interesting result is the number of polytopes reduced from 40.5 to 20.44. As the size of the history tree is connected to the fragmentation of the polytopes, this implies a better coherence of the information and a reduction of the memory requirement.
In spite of an important number of occluders and a significant visual complexity, back splitting remains efficient on the third set, where 31% of the splitting operation are avoided. We note the same reduction for the computation time per query. Once again, the main result is the minimisation of the fragmentation of the polytopes. Back splitting reduces fragmentation from more than 56%.
This result illustrates the back splitting efficiency for reducing the fragmentation of the polytopes. Moreover, since less splitting operations are performed, this increases the robustness of the visibility computation. As a secondary result, the time per query is improved.

6. CONCLUSION

This paper has presented a solution to compute a coherent and exact visibility information between two polygons. The fundamental principles to achieve exact visibility computation has been recalled. From the first two tractable solutions study, we have proposed an unified approach taking advantage of both of them. It modifies the Nirenstein algorithm to provide a history tree. This allows to organise the visibility information similarly to Bittner, but suited for non-oriented polygon-to-polygon queries. Moreover, we have presented the back splitting optimisation that improves the visibility coherence and the robustness. As the information is described using a smaller set of polytopes, memory can also be spared.
Results show that our approach is mainly efficient with a consequent visibility complexity, which is the most challenging configuration in graphic applications such as realistic image synthesis.
As a future work, we plane to enhance such applications by taking advantage of the history tree as a visibility tool. Moreover, a collaboration with a telecommunication department is already in progress for an accurate and fast visualisation of electromagnetic waves.

7. REFERENCES

[Avi96a] David Avis and Komei Fukuda. *Reverse search for enumeration.* Discrete Appl. Math., vol 65, 21-46, 0166-218X, Elsevier Science Publishers B. V.

[Baj96a] C. L. Bajaj and V. Pascucci. *Splitting a Complex of Convex Polytopes in any Dimension.* In Proceedings of the 12th Annual Symposium on Computational Geometry, ACM, 88-97, May 1996

[Bit98a] J. Bittner and V. Havran and P. Slavik. *Hierarchical Visibility Culling with Occlusion Trees.* Proceedings of Computer Graphics International '98 (CGI'98), 207-219, 1998.

[Bit01b] Bittner and Jan Prikryl. *Exact Regional Visibility using Line Space Partitioning.* TR-186-2-01-06, 1-13, 2001

[Bit02c] J. Bittner. *Phd dissertation : Hierarchical Techniques for Visibility Computations.* Czech Technical University in Prague, October 2002.

[Coh03a] Cohen-Or,Chrysanthou,Silva, Durand. *A survey of visibility for walkthrought applications.* IEEE TVCG, pages 412-431, september 2003

[Dur02a] F. Durand, G.Drettakis, C. Puech. *The 3D visibility complex.* ACM Trans. Graph., vol. 21, 2, pages 176-206, ACM Press.

[Fuk94a] K. Fukuda and V. Rosta. *Combinatorial face enumeration in convex polytopes.* Computational Geometry: Theory and Application, vol. 4, 4, pages 191-198, 1994

[Kai02a] V. Kaibel and M. E. Pfetsch. *Computing*

the face lattice of a polytope from its vertex-facet incidences. Computational Geometry: Theory and Applications, vol. 23, 3, pages 281-290, November 2002

[Nir02a] S. Nirenstein and E. Blake and J. Gain. *Exact from-region visibility culling.* Proceedings of the 13th Eurographics workshop on Rendering, pages 192-202, 2002.

[Nir03a] S. Nirenstein. *Fast and accurate visibility preprocessing.* University of Cape Town, South Africa, October 2003.

[Pel93a] M. Pellegrini. *Ray Shooting on Triangles in 3-Space.* Algorithmica, vol. 9, 5, pages 471-494, 1993

[Tel92a] Seth J. Teller. *Computing the antipenumbra of an area light source.* Computer Graphics, (Proc. Siggraph '92), 26:139-148, July 1992.

[Som59a] Sommerville. *Analytical Geometry in Three Dimension.* Cambridge University Press, 1959.