Global Visibility

Jiří Bittner

Department of Computer Science, Czech Technical University,
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
e–mail: bittner@sgi.felk.cvut.cz

Abstract

Solving the hidden surface removal is an essential problem since the early time of computer graphics. Most of the algorithms designed, like z-buffer or BSP-tree, are not *output-sensitive* what means, that their time complexity is proportional to the total number of graphic primitives in the model. However, in many complex models most their parts are invisible to an observer while located in certain region. This observation yields a search to design output-sensitive algorithms with the run-time complexity (time of rendering) proportional to the number of visible graphic primitives. We present one such algorithm together withe an empirical evidence of the asset of the visibility determination. The method presented in this paper subdivides the model into smaller parts (*cells*) continuing with determination of visibility relations between them. To describe potentially visible sets a tree data structure is used, where edges correspond to tight approximations of *potentially visible frustrums*.

Keywords: *visibility, hidden surface removal, obscuration culling, spatial partitioning, computational geometry, virtual reality.*

1 Introduction

Visibility determination is one of the most important tasks in computer graphics. The goal of visibility determination (also known as hidden surface removal) is to determine parts of the *model of virtual environment* (MVE) which are visible, given a viewpoint and viewing direction.

Many algorithms to solve the hidden surface removal problem were developed, two of them most commonly used. An example of object space algorithm – the *Binary Space Partitioning* (BSP) preserves the topological information about the model in a binary tree. Knowing the position of the observer rendering order can be determined by appropriate traversal of the BSP-tree. The *z-buffer* is an image space oriented algorithm, solving the problem of visibility for each pixel of the screen. This is simple to implemented in hardware, thus commonly used in todays rendering systems. However, these algorithms are not *output-sensitive*, since they may spend significant time processing parts of the model actually invisible to an observer.

It seems to be worthwhile to study restrictions in visibility and to apply some more sophisticated methods of visibility preprocessing, to speedup queries for visible portion of the model given certain viewpoint or a set of viewpoints.

Teller and Séquin introduced a method to determine the *conservative visibility*, overestimating the visible portions of the model in [TS91, Tel92]. This method achieves very good performance when applied to densely occluded environments, such as models of architectural interiors. The algorithm is based on spatial subdivision and determination of potentially visible sets (PVS) (portions of the model) for each part of the subdivision.

Recently more general algorithms [CT96, GKM93] appeared, using an idea of obscuration culling instead of potentially visible sets. These algorithms are basically less complicated than Teller's approach and they are not so sensitive to the type of the model they are applied to. However, in the case of densely occluded interiors, the results obtained by obscuration culling does not seem to reach the performance of the PVS approach.

In this paper outlines of the visibility determination based on Teller's technique are described together with some results obtained by this method. Finally discussion about the effectiveness of visibility algorithms is presented, with some possible future directions in this field.

2 Algorithm Overview

Given a model of virtual environment, we attempt to subdivide it into smaller discrete parts called *cells*. Each cell is a convex polyhedral volume. When the spatial subdivision is completed, the *portal* enumeration is performed to find transparent polygonal regions upon the cell boundaries which connect adjacent cells. Obviously, the observer can see from one cell to another through portals on their shared boundary and to the more distant cells through *portal sequences*. In densely occluded environments, such as architectural models, cell boundaries often follow walls and partitions, so cells roughly correspond to rooms of the environment. The portals likewise correspond to doors and windows through which neighboring rooms are connected. Entities of the spatial subdivision, these are cells, boundaries and portals, together with their topological connections are represented by the *adjacency graph*. For each cell in the adjacency graph we determine a set of potentially visible regions. A hierarchical data structure – the *visibility tree* is used to store the visibility information. The root node of the tree corresponds to the *source cell*, which is actually the cell the tree is built for. The tree structure stores the *cell-to-cell* visibility relation, defining the order in which the cells are visited along sightline from the root cell (see figure 4). The visibility tree can be further extended to hold information about the *cell-to-frustrum* visibility, which estimates the potentially visible regions induced by the portal sequences. Using the static visibility relations several dynamic queries can be formulated, to find a *tight superset* of objects visible to the observer, knowing observer's location and viewing direction.

3 Spatial Subdivision

The spatial subdivision of the MVE is constructed incrementally based on the binary space partitioning. The *root cell* corresponds to the convex hull of the model. This cell is divided selecting the largest *occluder* along which a splitting plane is introduced, until

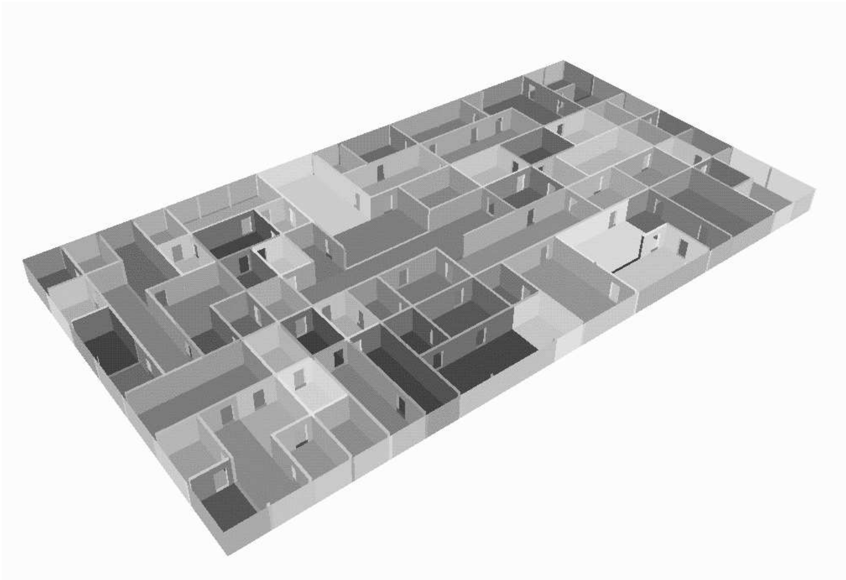

Figure 1: An example of the spatial subdivision in three dimensions. Different cells are shown in different colors.

no more candidates exist. The selection of the largest occluder in the current cell should prevent its descendants from "seeing" each other. In the current heuristics only polygons larger than some threshold are treated as occluders, assuming that all the other polygons form *detail-objects*. This choice makes the final cells of the subdivided model roughly correspond to actual rooms of the virtual environment. Thus, the spatial subdivision uses data structure known as BSP tree to store the intermediate results. The cell *boundaries* are created explicitly at each step of the spatial subdivision algorithm. When the spatial subdivision terminates the *portal* enumeration is invoked. The portals are computed as a convex hull of the set difference of the cell boundaries and the union of occluders coaffine to each particular boundary.

In order to preserve information about the spatial subdivision, the cell *adjacency graph* is created (see 2). Informally, the adjacency graph is a multi-graph structure, where nodes correspond to cells (leaves of the BSP tree) and where two cells are connected with an edge, if they share a portal on their common boundary. Thus edges of the adjacency graph correspond to the portals between cells of the spatial subdivision.

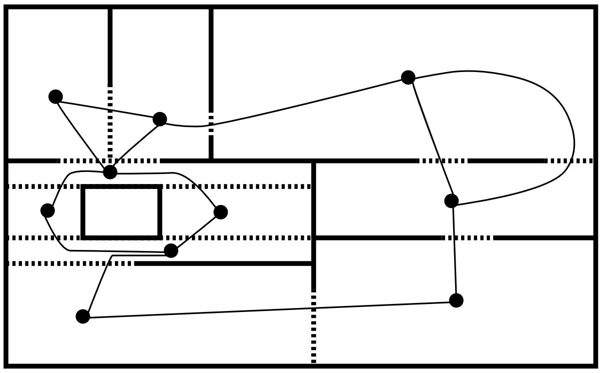

Figure 2: An example of the cell adjacency graph.

4 Static Visibility Preprocessing

The *cell-to-cell* visibility relation is based on the observation, that to see one cell from another an unobscured *sightline* between them must exist. The cell-to-cell visibility computation is performed as constrained depth first search on the adjacency graph. The existence of a sightline is established by involving the stabbing line algorithm for sequences of portals. This task is solved by transforming portal edges to *Plücker coordinates* (see e.g. [Tel92]), which suitably parameterize lines in 3D and involving six dimensional linear programming task. Furthermore the *cell-to-frustrum* visibility is determined. The computation of the potentially visible frustrum induced by the portal sequence is based on conservative approximation of the antipenumbra, when treating the first portal of the sequence as an areal light-source. The algorithm uses mapping to Plücker coordinates again and invokes the five dimensional convex polyhedra enumeration to produce a set of *extremal stabbing lines*. Some more computations are performed to conservatively approximate the potentially visible regions (see figure 3). We store the information about visibility relations using the extension of the Virtual Reality Modeling Language (VRML) file format, which is described in [Tre96].

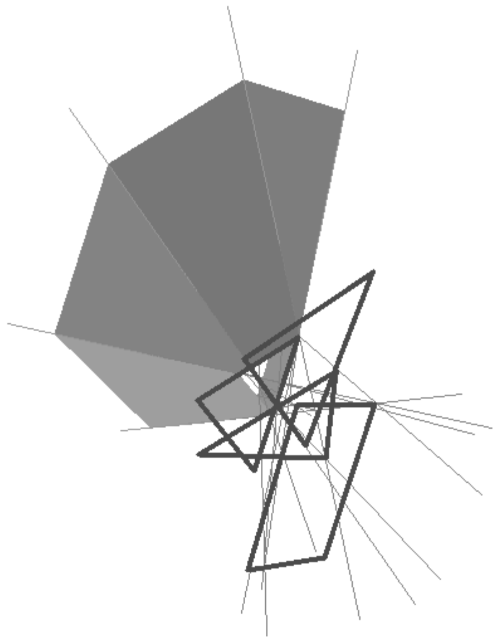

Figure 3: Potentially visible frustrum induced by the sequence of four portals. The extremal stabbing lines are marked as well.

5 Dynamic Visibility Queries

The static visibility relation generally overestimates the visible portion of the model to an observer, since it describes potentially visible portions of the model for a closed, but unlimited set of viewpoints in any direction. The *dynamic visibility* query is a process of real time determination of visible entities, which is dependent on the current location and viewing direction of the observer. To establish the *eye-to-cell* visibility we perform a depth first search on the visibility tree, and at each encountered portal we check its incidence with the observer's viewing frustrum (see figure 4). During the *eye-to-frustrum* visibility query, the DFS is constrained also by the *areas of interest* induced by encountered portal sequences. Note, that the area of interest corresponds to the antipenumbra on the "reverse" side of the portal sequence. Finally, reaching the cell during the cell-to-frustrum visibility, all the objects enclosed inside the frustrum are passed to the rendering subsystem.

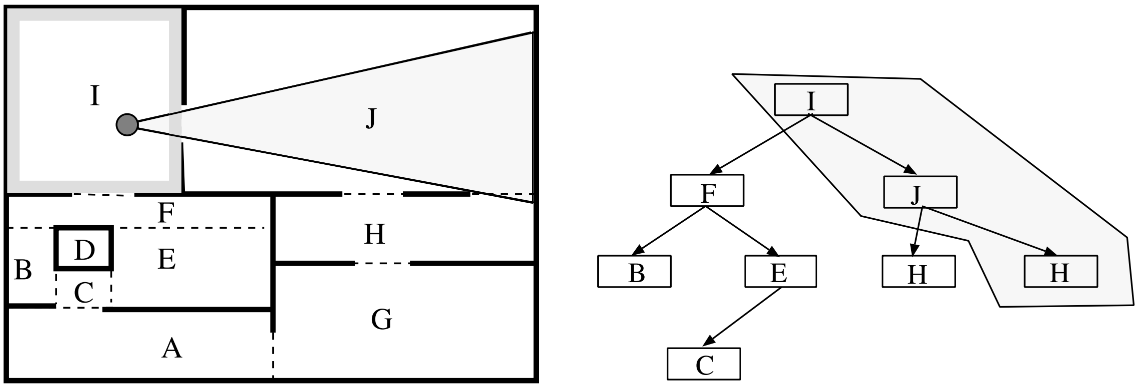

Figure 4: Viewing direction dependent pruning of the visibility tree during the dynamic visibility query.

6 Results and Discussion

We have tested the briefly described algorithms to establish visibility relations on several different models of architectural interiors. Table 1 depicts times spend by the visibility preprocessing as well as the asset of visibility computations. The percentage of cells classified as visible shows, that the visibility computations significantly reduce the amount of data necessary to process during rendering. However, efficiency of the visibility preprocessing tends to decrease rapidly with lower density of occluders in the environment (scene named `6supr`). In the case of sparsely occluded environments the size of visibility trees grows, because of the combinatorial explosion of the passable portal sequences. Now, let us provide a short discussion as a subject of the following paragraphs.

We have showed that rendering can be significantly speeded-up using an algorithm to determine the visible portion of the model in advance (i.e. before actually rendering it). However, a question remains how much work should be done as a preprocessing and what should be left for real-time calculations. Obviously, more precise preprocessing yields higher storage complexity for the precomputed information, while on the other hand reduces the complexity of on-line calculations and possibly increases their efficiency. Another question, which has not been answered yet, is when the usage of the portal based methods starts to be inefficient, leaving a place for the obscuration culling to be applied.

As already mentioned, the PVS based approach is efficient for processing densely occluded environments with cells and portals inherited in their structure. For such types of MVEs this method gives very good results and the preprocessed information is very close to the actually visible portion of the model. Moreover the dynamic queries can be performed very simply and quickly. However, if the model is not feasible enough

Scene name	room	6supr	bez2
Number of occluder polygons	30	358	1884
Subdivision time [s]	0.2	1.0	2.2
Number of occluders after SSD	88	1122	2374
Number of cells	24	140	223
Number of boundary polygons	119	622	977
Number of portals	36	120	245
Number of connected cells	2	35	23
VRML file size [kBytes]	85	1200	1200
Static visibility preprocessing [s]	2	45	23
Static visibility [% of cells]	32.9	12.0	5.0
Eye-to-cell [% of cells]	8.3	10.0	3.9
Eye-to-frustrum [% of cells]	7.0	5.0	2.1

Table 1: Results of the spatial subdivision and visibility computations. The table shows the time spend by the preprocessing as well as the average percentage of the portion of the model classified as visible. Measured on K5/100MHz, 32MB RAM, Linux OS.

preprocessing based on the PVS approach leads to enormous storage requirements. In such cases even more time can be spend during the dynamic queries than it would be needed using some most naive algorithm.

Obscuration culling methods are a general scheme, since practically no assumptions about the scene structure are required. This arises from the fact, that no search for the topology of the model has to be performed, namely no portals and cells need to be found. Obscuration culling methods use the "already defined" primitives such as polygons, to cull away the invisible portions of the model in contrast to the determination of the potentially visible sets induced by the transparent(!) portions of the model. Their potential weakness is the lack of knowledge about connectivity of primitives causing an occlusion, namely for densely occluded models (note the contrast to cell-portal approach).

It seems that most of todays research focuses on the improvement of the obscuration culling methods, since they are not so sensitive to the type of model being visualized and provide more freedom in terms of balance between preprocessing and real-time computations. Moreover as shown in [CT96], when more calculations are left for the real-time processing, the use of spatial and temporal coherence can significantly reduce their complexity.

7 Conclusion

In previous sections it was shown that the visibility computations can significantly speedup the visualization process. Furthermore, the information about restrictions in visibility can be applied to some other tasks, such as multi-user distributed virtual reality or photo-realistic visualization by the radiosity method.

Future work could be focused on algorithms, which take an advantage of both of the currently popular approaches – potentially visible sets and the "online" obscuration culling methods. The aim is to design algorithms less sensitive to the type of the MVE with a reasonable efficiency and possible adaptive balance between the preprocessing and calculations performed on the fly.

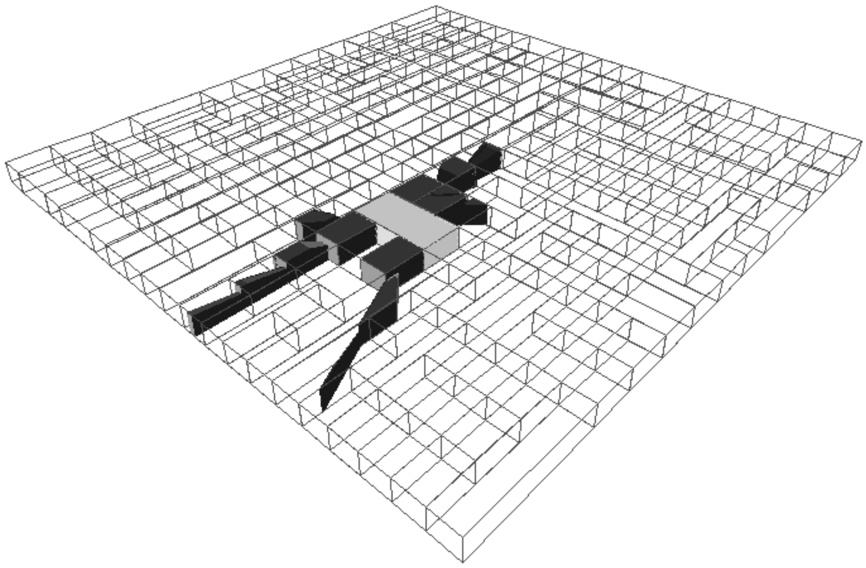

Figure 5: Cell-to-frustrum static visibility. Potentially visible regions shown in dark color.

References

[CT96] Satyan Coorg and Seth Teller. Temporally coherent conservative visibility. In *Proceedings of the Twelfth Annual ACM Symposium on Computational Geometry*, Philadelphia, PA, May 1996.

[DDP96] Frédéric Durand, George Drettakis, and Claude Puech. The 3D Visibility Complex, a new approach to the problems of accurate visibility. In *Eurographics Workshop on Rendering*, June 1996.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In L. Valastyan and L. Walsh, editors, *Proceedings of the Annual Conference on Computer Graphics*, pages 231–238. ACM Press, August 1993.

[LG95] David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, April 1995.

[Tel92] Seth Jared Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992. Also available as Technical Report UCB//CSD-92-708.

[TH93] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. In *Proc. SIGGRAPH '93*, pages 239–246, 1993.

[Tre96] M. Trefný. Formálni popis 3d scén. Master's thesis, Czech Technical University - Prague, Department of Computers, 1996.

[TS91] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walk-throughs. *Computer Graphics (SIGGRAPH '91 Proceedings)*, pages 61–69, July 1991.

[YR95] R. Yagel and W. Ray. Visibility computation for efficient walkkthrough of complex environments. *Presence: Teleoperators and Virtual Environments*, 5(1), 1995.