EUROGRAPHICS 2000 / M. Gross and F.R.A. Hopgood
(Guest Editors)

Volume 19 (*2000*), *Number 3*

Efficient Algorithms for Computing Conservative Portal Visibility Information

W. F. H. Jiménez, C. Esperança and A. A. F. Oliveira

L.C.G. - Laboratório de Computação Gráfica, COPPE - Sistemas / UFRJ, Rio de Janeiro, Brazil

Abstract

The number of polygons in realistic architectural models is many more than can be rendered at interactive frame rates. Typically, however, due to occlusion by opaque surfaces (e.g., walls), only small fractions of such models are visible from most viewpoints. This fact is used in many popular methods for preprocessing visibility information which assume a scene model subdivided into convex cells connected through convex portals. These methods need to establish which cells or parts thereof are visible to a generalized observer located within each cell. The geometry of this information is termed a 'visibility volume' and its computation is usually quite complex. Conservative approximations of viewing volumes, however, are simpler and less expensive to compute. In this paper we present techniques and algorithms which permit the computation of conservative viewing volumes incrementally. In particular, we describe an algorithm for computing the viewing volumes for a given cell through a sequence of 'm' portals containing a total of 'n' edges in $O(m \cdot n)$ *time.*

1. Introduction

Realistic scenes such as architectural models of furnished buildings may consist of several million polygons. They typically contain large connected elements of opaque material (e.g., walls), so that from most vantage points only a small fraction of the model can be seen. A visibility algorithm aimed at reducing the number of primitives rendered can exploit this property.

Following prior work[2, 6, 10], we make use of a spatial subdivision scheme that divides such models along the occluding primitives into cells and portals. A cell is a convex polyhedral volume of space; a portal is a transparent convex 2D region on a cell boundary that connects adjacent cells. Cells can only "see" other cells through portals. The portals are stored in a list along with the identifiers for the two neighboring cells to which the portal leads, and both cells have a reference to the portal. In this way we are constructing an adjacency graph in which two cells (vertices) are adjacent (share an edge) if and only if there is a portal connecting them.

It is frequently convenient to compute Visibility information offline, so it can be associated with each cell for later use in an interactive rendering phase. Given a *generalized observer*, i.e., an observer constrained to lie within the cell, but free to move anywhere inside it and to look in any direction, the visibility information can be categorized into *cell-to-cell*, *cell-to-region* and *cell-to-object* visibility, corresponding to the set of cells, the viewing volume, or the set of objects which are visible to that observer.

Computing the viewing volume – also known as the *antipenumbra* – for a sequence of portals means to characterize the volume illuminated by the light source (i.e., the first portal in the sequence) within the target cell (i.e., the cell reached through the last portal in the sequence). Clearly, a point belongs to the viewing volume if and only if it belongs to a line stabbing the sequence. Computing this volume for a sequence of a single portal is trivial: the antipenumbra in this case is the entire halfspace beyond the portal, intersected with the reached cell which is, of course, an immediate neighbor of the source cell. Since we are only considering convex cells, then this intersection yields exactly the volume of the reached cell. Computing the antipenumbra of portal sequences of length two or more involves interactions among vertices and edges of different portals and requires both linear and quadratic implicit primitives (line swaths) to correctly describe the illuminated volume. Teller[10] found an analytic solution for this portal-portal visibility problem but his approach is mathematically and computationally complex, requiring hours of preprocessing time for large mod-

els and it is not yet sufficiently robust for use on complex models. Consequently, he developed a simpler algorithm that computes a polygonal volume which conservatively estimates the exact visible region. In other words, this volume, which we will refer to henceforth as a *conservative viewing volume* is guaranteed contain the *exact* viewing volume.

In this paper we describe a new approach for simultaneously computing the conservative approximations of cell-to-cell and cell-to-region visibilities for each cell of the subdivision. This approach is simpler than that described by Teller[10] and, at the same time, makes it possible to compute the visibility information for the whole scene incrementally. It uses an adaptation of the Visibility Skeleton[3] to encode all possible visibility changes of a sequence of portals into a graph structure. The Visibility Skeleton is a powerful tool which can be used to solve numerous different problems which require global visibility information; it is easy to build and it is well-adapted to *on-demand* or *lazy* construction. The central components of the Visibility Skeleton are *line swaths* – sets of critical lines – and *extremal stabbing lines*, which are the foci of all visibility changes in a scene. All modifications of visibility in a polygonal scene can be described by *extremal lines*, and a set of *line swaths* which are necessarily adjacent to these lines.

We present a description of our adaptation of the Visibility Skeleton in Section 2. In Section 3 we describe in detail: an algorithm to compute the viewing volume for two portals, an algorithm to compute a conservative approximation of the viewing volume for three portals and, finally, an algorithm that handles sequence of four or more portals. In Section 4 we show the results obtained by applying these to a single synthetic scene. Our conclusions and suggestions for further work are presented in Section 5.

2. The Portal Visibility Skeleton

The Visibility Skeleton (VS) devised by Durand, Drettakis and Puech[3] is a tool for solving many different problems which require global visibility information. The Portal Visibility Skeleton (PVS) introduced here borrows the idea of encoding visual events into a graph structure, but whereas the VS tries to handle general polyhedral scenes, the PVS focuses on portals and cells. Not all of the visual events encoded by VS and by PVS are the same. Thus, some visual events exist in VS but not in PVS and vice-versa. Also, some visual events are encoded in a different way.

Many global visibility algorithms have been proposed in the recent literature, in particular those which compute aspect graphs[5, 11, 12], antipenumbra[9] or discontinuity meshing[1, 4]. In many of those, visibility changes, also known as visual events, have been characterized by *critical line sets* or *line swaths* and by *extremal stabbing lines*. Following the procedure used for the Visibility Skeleton[3], we encode these visual events in a graph structure that can describe completely all possible visibility relationships along a sequence of portals. The nodes of the graph structure represent extremal stabbing lines while arcs correspond to line swaths.

In this paper, we do not describe the PVS in full, but only those parts of the PVS which are needed for the analysis of interactions between pairs of portals. As is explained in Section 3, the conservative antipenumbra estimations for arbitrary sequences of portals can be derived incrementally from such pairwise analysis.

2.1. Extremal stabbing lines

Following prior work[3, 8, 9], we define an *extremal stabbing line* to be incident on four portal edges. Although the Visibility Skeleton defines a host of extremal stabbing line types, we must consider only two: *vertex-vertex* (or VV) lines, and *portal-vertex-vertex* (PVV) lines. Notice that this simplification is possible because our algorithms will examine only interactions between pairs of portals. Thus, for instance, extremal stabbing lines that touch four edges (type EEEE) are not explicitly computed. The VV lines (represented by VV nodes in the graph structure) correspond to the interaction of two vertices (see Figure 1). The PVV lines (represented by PVV Nodes) correspond to the interaction of two vertices v_1 and v_2, each lying on a different portal, but both on the same plane as one of the two portals, which we refer to as p. Without loss of generality, we assume that v_1 is one of the vertices of p. There are five possible configurations for PVV lines, depending on the relative position of the line with respect to the vertices of p (see Figure 2).

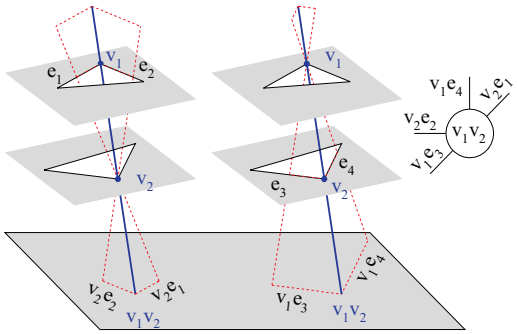


Figure 1: *A VV node v_1v_2 is adjacent to four EV arcs defined by a vertex and edges of the other vertex: v_1e_3 and v_1e_4 and v_2e_1 and v_2e_2.*

2.2. Line swaths

A *swath* is the surface swept by extremal stabbing lines when they are moved after relaxing exactly one of the four edge constraints defining the line. In general, a swath can either be planar (if the line remains tight on a vertex) or a regulus, whose three generator lines embed three edges. Once again, we can simplify our discussion of the Portal Visibility

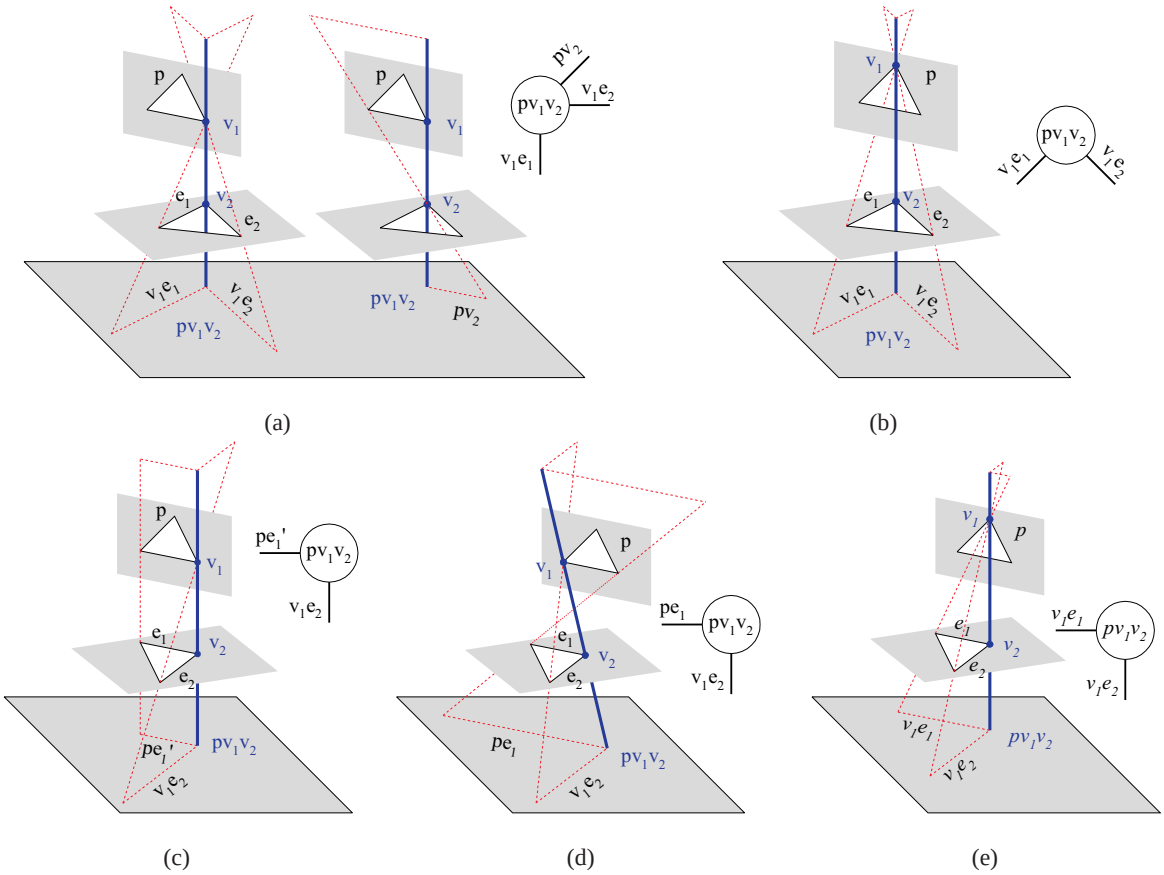


Figure 2: *A PVV node pv_1v_2. In (a) and (b) vertex v_2 is on the same plane as portal p. Unlike the case depicted in (b), in (a) all vertices of p are on the same closed sub-plane defined by line pv_1v_2. In (c), (d) and (e) edge e_1 is on the same plane as portal p. In (c) and (d) but not in (e) all vertices of p are on the same closed sub-plane defined by pv_1v_2. Finally, in (c) edge e_1 and p are on the same closed sub-plane defined by pv_1v_2, and in (d) they are on different sub-planes.*

Skeleton by considering only planar swaths, since no stabbing lines embedding three edges will ever occur between two portals.

In Figure 3, we see the three possible arc types: an EV line swath (a), a PV line swath relating a portal p to one vertex of another portal (b), and a PE line swath relating a portal p to an edge of another portal (c). It should be clear that PV lines occur when the portal p is on the same plane as a vertex of the other portal, and PE lines occur when the portal p is on the same plane as some edge of the other portal. In the upper part of each figure we show three views (with changes in visibility), as seen from a viewpoint located above the scene and, in front of, on, or behind the line swath.

3. Computing the Conservative Viewing Information

We make the following assumptions: the input is a list of m oriented convex polygons (portals), P_1 ... P_m. The first polygon P_1 is the light source, and all others are holes. The polygons are disjoint, and ordered in the sense that the negative halfspace determined by the plane of P_i contains all polygons P_j, $i < j \leq m$. Thus, an observer looking along a stabbing line from the light source would see the vertices of each polygon arranged in counterclockwise order.

We assume that each portal P_i is placed between cells C_i and C_{i+1}. Thus, it is clear that we may initialize the cell-to-cell visibility information of cell C_i with links to C_{i-1} and C_{i+1}, for all $1 < i < m$. The cell-to-cell information of cells C_1 and C_m is initialized with links to C_2 and C_{m-1}, respectively.

Given two consecutive portals P_i and P_{i+1}, as we are only considering convex cells, then cell C_{i+2} is directly visible from source cell C_i. Thus, we can add cells C_{i+2} and C_{i-2} to the set of cells that form the cell-to-cell visibility information of cell C_i, for all $2 < i < m-1$. Also, we can add cells C_3 and C_{m-2} to the cell-to-cell information of cells C_1 and C_m, respectively.

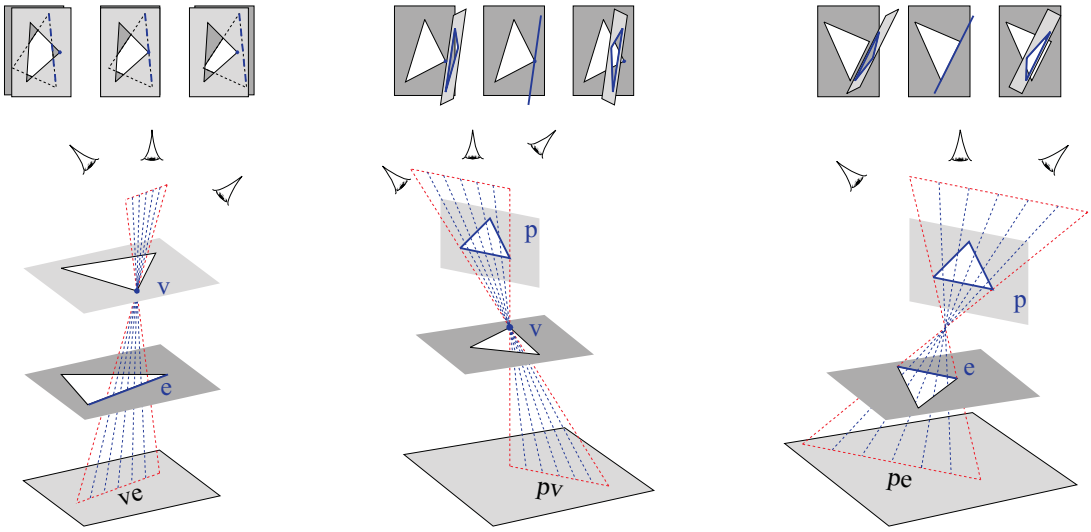


Figure 3: *(a) While the eye traverses line swath ve, vertex v crosses the edge e. (b) In front of swath pv we see the front side of p, on the swath we see a line, and behind we see the other side of p. (c) The swath pe is similar to the pv case.*

In general, for cells C_i and C_{i+k} to be visible to each other, for $k > 2$, there must be at least one stabbing line through portals P_i, P_{i+1} ... P_{i+k-1}. In other words, mutual visibility between cells C_i and C_{i+k} requires the viewing volume for the portal sequence P_i, P_{i+1} ... P_{i+k-1} to be non-null.

Cell-to-object visibility information can be easily computed by first determining in which cell each object lies and, if that cell is visible from the source cell, then the object is visible only if it falls within the viewing volume for the sequence of portals connecting the two cells.

All it remains now is to be able to compute the viewing volume (antipenumbra) for portal sequences of arbitrary length.

There is a key observation shown by Teller[10] which is important for our algorithm: if we consider any pair of portals P_i, P_j, $i < j \leq m$ and any edge e on one of them (Figure 4), then edge e uniquely defines a *separating plane* such that P_i and P_j lie in different halfspaces. This separating plane is spanned by edge e, and some vertex v on the other portal. Clearly, one halfspace of this plane (the halfspace containing the portal P_j) must contain the antipenumbra of any portal sequence containing this portal pair. As shown in Section 2.2, edge e and vertex v define a swath, which will be called a *separating swath*. Similarly, its corresponding arc in the graph structure will be called a *separating arc*.

First we present an efficient algorithm to compute the viewing volume for two portals, next the algorithm to compute the conservative approximation of the viewing volume for three portals, and finally, the algorithm to compute a conservative approximation of the viewing volume for a sequence of four or more portals.

3.1. The viewing volume for two portals

In order to complete the visibility information for this case, we need only to compute the cell-to-region viewing volume for the portal sequence P_1, P_2. It should be noted that when only two portals are involved, the viewing volume will be bounded by separating planes only, and thus the procedure described below will actually compute the *exact* viewing volume.

We begin with the simplest case, that is, when no vertex of one portal is on the same plane as the other portal. Here, all extremal stabbing lines are Vertex-Vertex lines and, in the graph structure, each node has four adjacent arcs representing VE swaths, as shown in Figure 5a.

Consider the edges of every portal as directed segments, following the order of its vertices, that is, a counterclockwise order as shown in Figure 5b. Then we can impose this direction on every swath defined by the edges and on their corresponding arcs in the graph structure.

In Figure 5b the directed separating swaths and their corresponding directed separating arcs in the graph structure are drawn in solid lines. These separating swaths form the border of the viewing volume, and the nodes in the graph structure on which they are incident represent the Vertex-Vertex extremal stabbing lines. We call these nodes the *bounding nodes* of the viewing volume.

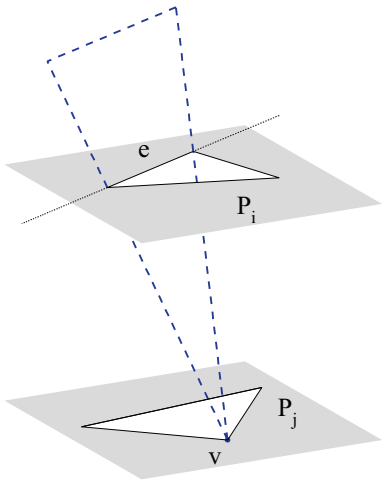


Figure 4: *The plane through edge e, separating portals P_1 and P_2.*

Computing the viewing volume for a sequence of two portals means to find, in the graph structure, all the bounding nodes and all the separating arcs. We can do that by finding one bounding node and then following the directed separating arcs to reach the other bounding nodes. Note that each node has two incoming and two outgoing arcs. This means that when a given node is reached, we need to decide which of the two outgoing arcs is a separating arc. If both are separating arcs, then we can choose either.

Our algorithm has two parts. The first part consists of choosing the first separating arc. For this, we pick any edge of one portal and find a separating swath defined by it and some vertex on the other portal. This separating swath corresponds to the first separating arc to be traversed by the algorithm and is stored in a list. The second part of the algorithm consists of repeatedly finding another separating arc which leaves the bounding node reached by the last arc in the list. When the first bounding node is reached again, the algorithm stops, and the list contains the computed antipenumbra.

Let n_1 and n_2 be the number of edges of the two portals. Since any edge defines a separating arc, then the first part of the algorithm entails only searching for an appropriate vertex. Given an edge e of the first portal and a vertex v_i of the second portal, then e and v_i define a separating plane if and only if v_{i-1} and v_{i+1} both lie on the same side of the plane but all remaining vertices of the first portal (which are not endpoints of e) lie on the opposite side. This can be determined merely by testing v_{i-1} and v_{i+1}, since the relative positions of the vertices of the first portal with respect to e are known in advance. Thus, even if all vertices of the second portal need to be tested, the first part of the algorithm can be performed in $O(n_2)$ time. Of course, we could have chosen an edge of the second portal, in which case locating the appropriate vertex of the first portal would take $O(n_1)$ time.

Since each edge of both portals uniquely defines a separating swath with some vertex on the other portal and is part of the viewing volume, then the total number of separating swaths that form the viewing volume is $O(n_1+n_2)$. In the second part of the algorithm, each time a bounding node is reached, we need to compute two swaths and check which one is the separating swath. Thus, the total time to identify the separating swaths is $O(2(n_1+n_2))$, which is also the total worst-case time complexity of the algorithm. Note that this is considerably better than the $O((n_1+n_2)^2)$ algorithm proposed by Teller[10].

Finally, we have to consider the cases where a vertex (Figure 6a) or an edge (Figure 6b) of one portal lies on the same plane as the second portal. In such cases, the traversal of the graph is similar, but when a PVV node is reached through an VE arc, we must choose the PV or PE arc leaving from it in order to reach the next PVV node.

3.2. The conservative viewing volume for three portals

The rationale used in the algorithm to compute the conservative viewing volume for three portals can be better understood by referring to the two-dimensional example shown in Figure 7. In our discussion, we denote by VP_iP_j (or VP_jP_i) the exact viewing volume formed by the lines stabbing portals P_i and P_j. Thus, in Figure 7a, P_1 and P_2 defines a exact viewing volume called VP_1P_2 (or VP_2P_1).

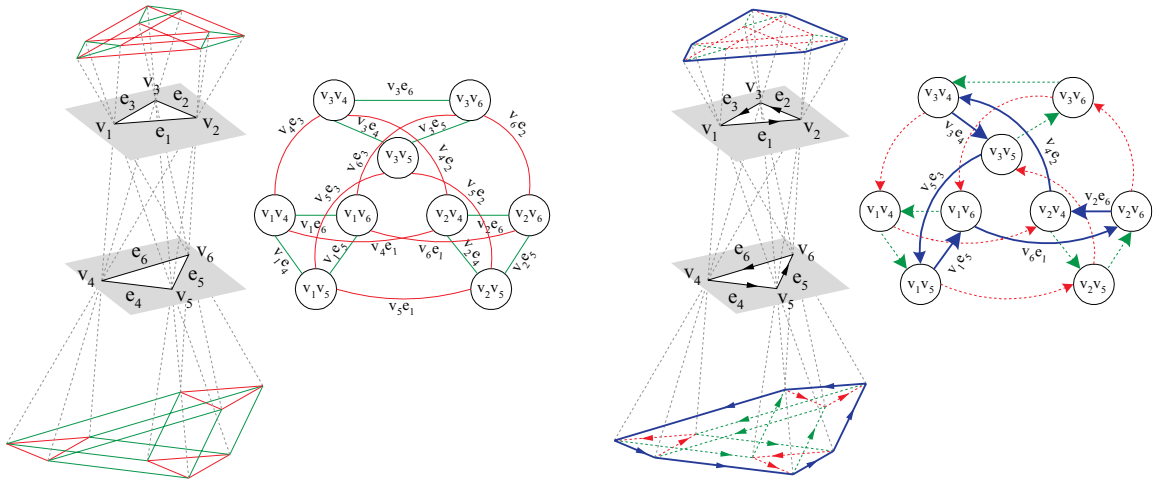
The exact viewing volume VP_1P_2 is formed by all lines stabbing portals P_1 and P_2 and, if some of these lines stab portal P_3 as well, then they form the conservative viewing volume for portals P_1, P_2 and P_3, which we will term $VP_1P_2P_3$. All lines that form $VP_1P_2P_3$ stab portals P_2 and P_3 and are a subset of VP_2P_3. Similarly, all lines that form $VP_1P_2P_3$ stab portals P_1 and P_3 and are a subset of VP_1P_3. Thus we can say that $VP_1P_2P_3$ is formed by the intersection of VP_1P_2, VP_2P_3 and VP_1P_3, (Figure 7b).

If part of P_3 falls outside VP_1P_2 then the lines of $VP_1P_2P_3$ will stab only the part of P_3 that falls within VP_1P_2 (Figure 7b). We denote this part of P_3 as portal P_3^1. Thus, $VP_1P_2P_3$ and $VP_1P_2P_3^1$ are the same volume (Figure 7c). Similarly, if part of P_1 falls outside $VP_2P_3^1$ then the lines of $VP_1P_2P_3^1$ will stab only the part of P_1 that falls within $VP_2P_3^1$ (Figure 7c). This part of P_1 is a portal denoted by P_1^1. Thus, $VP_1P_2P_3^1$ and $VP_1^1P_2P_3^1$ are the same volume (Figure 7d). We also notice that $VP_1P_2P_3$ and $VP_1^1P_2P_3^1$ are the same volume.

Finally, we see that $VP_1^1P_2P_3^1$ and $VP_1^1P_3^1$ are the same volume (Figure 7d), that is,

$$VP_1^1P_2P_3^1 = VP_1^1P_2 \cap VP_1^1P_3^1 \cap VP_2P_3^1 = VP_1^1P_3^1$$

In order to understand this, notice that since P_3^1 is totally



(a) (b)

Figure 5: *(a) Extremal stabbing lines, swaths and graph structure for two portals. (b) The directed swaths and directed graph structure.*

included in $VP_1^1P_2$, then $VP_1^1P_3^1$ is a subset of $VP_1^1P_2$. Similarly, since P_1^1 is totally included in $VP_2P_3^1$, then $VP_1^1P_3^1$ is a subset of $VP_2P_3^1$.

This way, we may compute the conservative approximation of the viewing volume for the sequence of three portals P_1, P_2 and P_3 as though we had only two portals: P_1^1 and P_3^1. The algorithm is described schematically below. We use the following notation:

- `Volume` *ViewVolume*3(P_i, P_j, P_k) is the function for computing the conservative viewing volume for portals P_i, P_j and P_k.
- `Volume` *ViewVolume*2(P_i, P_j) is the function for computing the exact viewing volume for portals P_i and P_j, as described in Section 3.1.
- `Portal` *IntersectVolumePortal*(V, P) denotes a function to compute portal P clipped by viewing volume V, that is, $P \cap V$.

The algorithm can be best understood by referring to a simple example in two dimensions as shown in Figure 7. It consists of the following steps:

```
Volume ViewVolume3(P1, P2, P3)
  begin
    VP1P2 ← ViewVolume2(P1, P2)
    P3^1 ← IntersectVolumePortal(VP1P2, P3)
    VP2P3^1 ← ViewVolume2(P2, P3^1)
    P1^1 ← IntersectVolumePortal(VP2P3^1, P1)
    VP1^1P3^1 ← ViewVolume2(P1^1, P3^1)
    return VP1^1P3^1
  end
```

It should be noted that, in practice, the first step of the algorithm is seldom necessary since the value of VP_1P_2 will be available from previous computations.

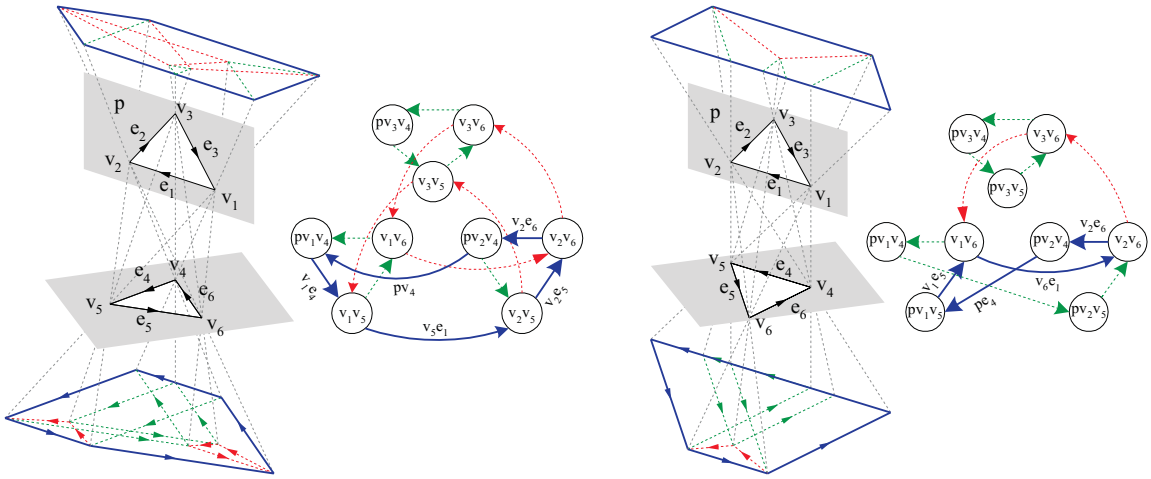
The time complexity of the algorithm is $O(n)$, where n is the total number of edges in P_1, P_2 and P_3. This can be inferred by noticing that:

i. The time complexity of function *ViewVolume*2 is linear on the number of edges of its arguments.
ii. Function *IntersectVolumePortal* can be implemented in a manner that is analogous to any algorithm for computing the intersection of convex polygons. The complexity of this problem was shown to be[7] $O(n_1 + n_2)$, where n_1 and n_2 are the number of edges of the operand polygons.
iii. The complexity of conservative viewing volumes, i.e., the number of separating swaths that form them, was shown to be not greater than twice the total number of edges in the portal sequence[10]. This is due to the fact that each edge can contribute with no more than 2 separating swaths.
iv. Since the complexity of the final viewing volume $VP_1^1P_3^1$ is no greater than $2n$, then portals P_1^1 and P_3^1 will sum no more than $2n$ edges.

Since all functions used in the algorithm have time complexity linearly dependent on the number of edges/swaths of their arguments, and given that all arguments have complexity no greater than $2n$, then we may safely assume that the algorithm is linear on n.

3.3. The conservative viewing volume for sequences of four or more portals

Now we compute the conservative approximation of the viewing volume for a sequence of m oriented portals P_1 ...



(a) (b)

Figure 6: *Directed swaths and directed graph structure when (a) vertex v_1 is on the same plane as portal p, and (b) edge e_4 is on the same plane as portal p.*

P_m, $m > 3$. We do this in a progressive way, that is, once we know the conservative viewing volume for the sequence P_1 ... P_i , the conservative viewing volume for the sequence P_1 ... P_{i+1} may be computed with one more call to function *ViewVolume*3.

Given the first two portals P_1 and P_2, we can compute the viewing volume VP_1P_2 as described in Section 3.1. Next, when portal P_3 is added, we use the process described in Section 3.2 to compute the conservative approximation of the viewing volume for this sequence of three portals, which is the volume $VP_1^1P_3^1$.

When the fourth portal P_4 is added, we compute the conservative approximation of the viewing volume for this sequence of four portals as though we had only three portals: P_1^1, P_3^1 and P_4 , as follows:

$$VP_1^2P_4^1 \leftarrow ViewVolume3(P_1^1, P_3^1, P_4)$$

The remaining portals can be processed in the same way. Thus, in general, we may write:

$$VP_1^{(m-2)}P_m^1 \leftarrow ViewVolume3(P_1^{(m-3)}, P_{(m-1)}^1, P_m)$$

The algorithm is presented schematically below:

```
Volume ViewVolume(P1,P2,...,Pm)
  begin
    for i ← 2 to m do
      if i = 2 then
        VP1P2 ← ViewVolume2(P1,P2)
      else if i = 3 then
        VP1_1P3^1 ← ViewVolume3(P1,P2,P3)
      else
        VP1^(i-2)Pi^1 ← ViewVolume3(P1^(i-3),P(i-1)^1,Pi)
      end if
    end for
    return VP1^(m-2)Pm^1
  end
```

Notice that the resulting viewing volume $VP_1^{(m-2)}P_m^1$ will contain no more than $2n$ separating swaths[10], where n is the total number of edges in portals P_1 ... P_m. It is safe to say that portals $P_1^{(m-3)}$ and $P_{(m-1)}^1$ may have not more than $2n$ edges each. Thus, the last call to function *ViewVolume*3 will spend $O(n)$ time. We may safely assume that all other calls to functions *ViewVolume*2 and *ViewVolume*3 are also bounded by $O(n)$. Overall, $m-1$ such calls are performed and thus we infer that the worst case time complexity of the algorithm is bounded by $O(m \cdot n)$. Observe that this is, in most cases, better than the algorithm proposed by Teller which runs in $O(n^2)$ time.

4. Implementation and construction statistics

The algorithms described above has been implemented in C++ and applied to a synthetic test scene comprising 72 square cells arranged in a 8 by 9 rectangular array (see Fig-

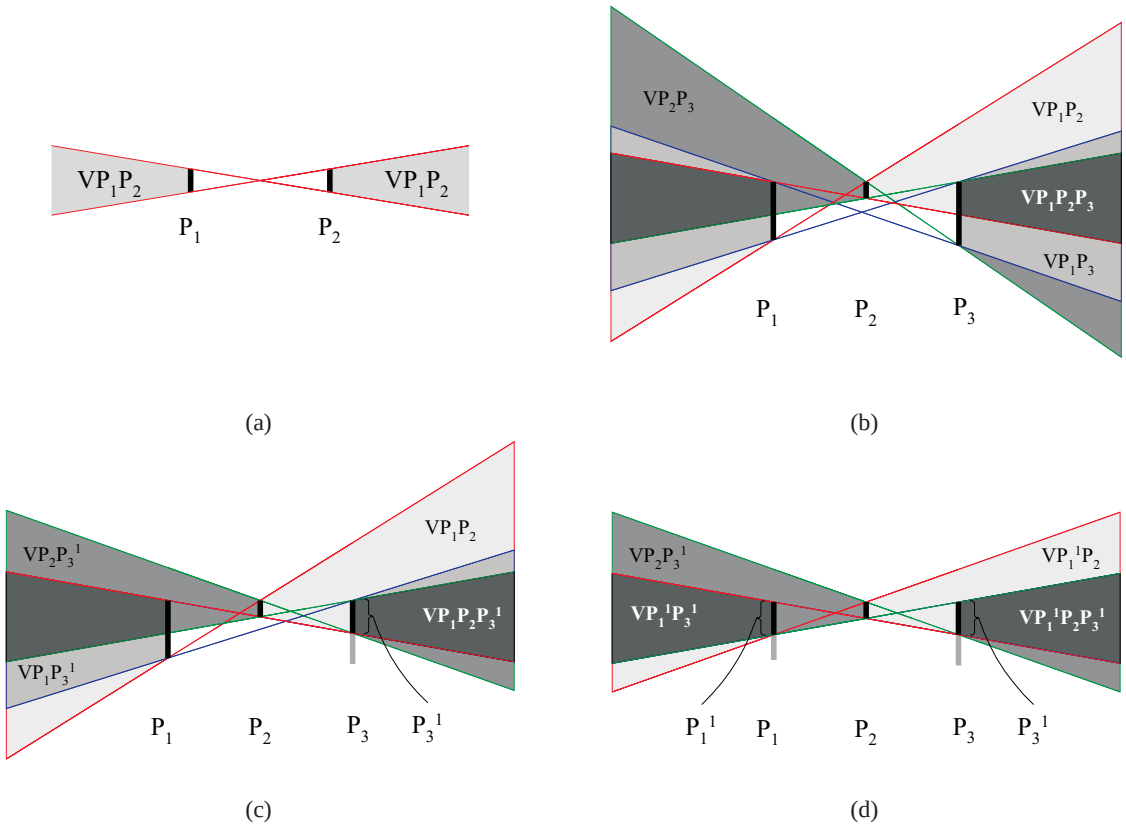


Figure 7: *(a) P_1 and P_2 define a viewing volume called VP_1P_2 (or VP_2P_1). (b) $VP_1P_2P_3$ is formed by the intersection of VP_1P_2, VP_2P_3 and VP_1P_3. (c) $VP_1P_2P_3^1$ and $VP_1P_2P_3$ are the same volume, (d) $VP_1^1P_3^1$ and $VP_1^1P_2P_3^1$ are the same volume.*

ure 8). All adjacent cells in the scene are connected by convex polygonal portals ranging from 3 to 10 edges. The application computed cell-to-cell and cell-to-region visibility information for all cells and all portal sequences of the test scene in roughly 28 seconds on a Pentium-II-based computer running at 300MHz. A total of 5772 viewing volumes were found, i.e., the conservative cell-to-cell visibility graph for the scene contains that number of edges. Detailed statistics are shown in Table 1.

Although this simple test scene cannot be regarded as a typical architectural model, the obtained results seem to confirm our estimation that the proposed algorithm computes the conservative viewing volume for sequences of m portals in time which varies linearly with m. In fact, since the average complexity of the viewing volumes does not vary overmuch in the scene, the average time required to process another portal in a sequence (i.e., calls to *ViewVolume*3) also remains fairly constant. We believe that this situation is not uncommon in most typical scenes, that is, as a sequence of portals grows longer, the complexity of the resulting conservative viewing volume tends to stabilize. As a consequence, for such scenes, the total time required to compute the conservative antipenumbra for a sequence of portals will vary linearly with the length of the sequence. This is evidenced graphically in the chart presented in Figure 9.

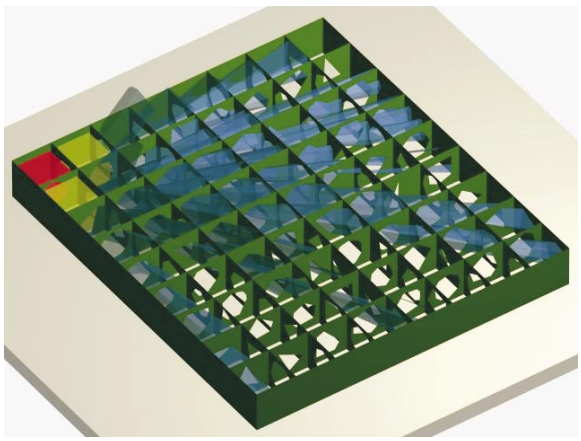


Figure 8: *A simple test scene. The viewing volumes of all portal sequences starting at the red cell are shown.*

Sequence length (portals)	Number of sequences	Avg. total number of portal edges in sequence	Avg. complexity of antipenumbra (number of swaths)	Avg. time to add one portal to sequence (s)	Avg. total time for computing sequences' antipenumbrae (s)
2	334	12.4371	12.4371	0.000849	0.000849
3	659	18.7602	12.2800	0.002696	0.003560
4	930	25.2891	11.7378	0.002785	0.006421
5	1001	32.1263	11.4363	0.002539	0.009184
6	901	38.9373	11.4309	0.002509	0.011869
7	703	45.8478	11.3250	0.002471	0.014598
8	478	52.5862	11.3835	0.002455	0.017208
9	304	59.3810	11.5205	0.002495	0.019978
10	175	65.6286	11.4371	0.002443	0.022407
11	89	71.8883	11.5642	0.002443	0.024820
12	43	77.5632	11.4828	0.002461	0.027101
13	18	83.1351	12.3243	0.002729	0.030716
14	8	88.7500	12.0625	0.002676	0.033174
15	2	95.5000	11.0000	0.002417	0.034375

Table 1: *Results obtained by our sample implementation when applied to the test scene shown in Figure 8.*

5. Conclusions

We have presented several techniques and algorithms that make possible the computation of conservative visibility information for scenes composed of convex cells and portals. We introduced the Portal Visibility Skeleton, an adaptation of the Visibility Skeleton proposed by Durand et al.[3] and used it to solve the problem of computing the antipenumbra for a sequence of two portals. We presented an algorithm for computing conservative antipenumbrae for portal sequences of arbitrary length which has time complexity that is no worse than previously known approaches and, in some cases, it fares considerably better.

We are currently investigating the possibility of computing the conservative visibility volume for sequences of three portals using the PVS. Although the asymptotic complexity of the algorithm should remain the same, we hope to obtain a faster algorithm than the one presented herein. We also plan on using the PVS to compute exact visibility volumes.

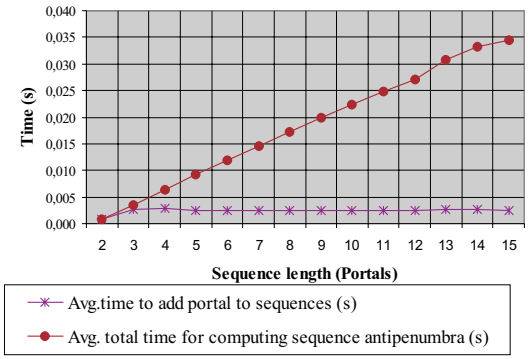


Figure 9: *Average times taken by antipenumbra computations as a function of the sequence length.*

References

1. A.J.Steward and S.Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. *Computer Graphics* : 231-238. Orlando. SIGGRAPH'94 Proc. July 1994. 2

2. C.B.Jones. A New Approach to the 'Hidden Line' Problem. *The Computer Journal* **14**(3): 232. August 1971. 1

3. F.Durand, G.Drettakis, and C.Puech. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. Turner Whitted. *Computer Graphics* : 89-100. Los Angeles, Addison Wesley. An-

nual Conference Series. SIGGRAPH'97 Proc. August 1997. 2, 9

4. G.Drettakis and E.Fiume. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. *Computer Graphics* : 223-230. Orlando. SIGGRAPH'94 Proc. July 1994. 2

5. H.Plantinga and C.R.Dyer. Visibility, Occlusion, and the Aspect Graph. *International Journal of Computer Vision* **5**(2): 137-160. 1990. 2

6. J.Airey. Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. **TR #90-027**. UNC-CH CS Departament. Ph.D Thesis. July 1990. 1

7. J.O'Rourke. Computational Geometry in C. Cambridge University Press. 1994. 6

8. M.Pellegrini. Stabbing and Ray Shooting in 3-Dimensional Space. *Proceedings of the Sixth Annual Symposium on Computational Geometry* : 177-186. June 1990. 2

9. S.J.Teller. Computing the antipenumbra of an area light source. *Computer Graphics* **26**(4): 139-148. Chicago. SIGGRAPH'92 Proc. July 1992a. 2

10. S.J.Teller. Visibility Computation in Densely Occluded Polyhedral Environments. **TR #92/708**. UC Berkeley CS Department. Ph.D Thesis. 1992b. 1, 2, 4, 5, 6, 7

11. Z.Gigus, J.Canny, and R.Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. on Pat. Matching & Mach. Intelligence* **13**(6). June 1991. 2

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