PERGAMON

Computers & Graphics 23 (1999) 703–717

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Visibility — techniques and applications

Interactive ray tracing with the visibility complex

Franklin S. Cho*, David Forsyth

University of California at Berkeley, Computer Science Division, S47 Soda Hall, Berkeley, CA, USA

Abstract

We describe a method of producing ray-traced images of 2D environments at interactive rates. The 2D environment consists of a set of disjoint, convex polygons. Our technique is based on the visibility complex [17,19] [Pocchiola M, Vegter G. Proc Int J Comput GEOM Applic 1996;6(3):279–308. Rivière S. Visibility computations in 2D polygonal scenes. PhD thesis, Univ. Joseph Fourier, Grenoble I, France], a data structure in a dual space where a face of the visibility complex corresponds to a contiguous set of rays in the primary space with the same forward and backward views. Sweeping the viewing ray around a viewpoint corresponds to walking along a trajectory on the visibility complex. Producing a ray-traced image is equivalent to walking along and maintaining a set of trajectories. Generating ray-traced images with the visibility complex is very efficient since it uses the coherence among the rays effectively. We have developed a new algorithm for the randomized incremental construction of the visibility complex. The advantage of using an incremental algorithm is that the history of the incremental construction yields an efficient ray-query data structure, which is required for casting secondary rays. The performance of our algorithm is analyzed and a comparison is made with the classical ray-tracing algorithm. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Interactive ray tracing; Visibility complex; Coherence; Line space

1. Introduction

Ray tracing accurately simulates reflection, refraction and lighting effects, thereby rendering appealingly accurate images. The main drawback is that rendering is slow. A casual look at the set of rays reveals that there is an apparent coherence, i.e., many adjacent rays encounter the same set of reflecting/refracting media in the same order (Fig. 1). This coherence is difficult to exploit directly, however, since knowing when this coherence is broken by an obstacle is not a trivial matter.

This coherence can be thought of in terms of "sweeps" of a set of rays. Sweeping a set of rays through a set of obstacles motivates a special data structure that can efficiently report when a ray will encounter an obstacle. Such a data structure is cumbersome in the primary space where the objects reside. However, the representation becomes very clean when we work in a dual space, i.e., a space where each point corresponds to a line in the primary space. Sweeping a ray in the primary space is equivalent to walking along a trajectory in a dual space.

Traditional visibility algorithms have worked almost exclusively in the primary space. By shifting our attention to a dual space, a seemingly difficult problem of ray coherence becomes particularly easy to visualize and represent. In this paper, we will present how the visibility complex data structure can be used to efficiently produce ray-traced images of 2D environments. Visibility computations in 2D environments have many real-world applications, since much of the world around us is essentially 2D or $2\frac{1}{2}$D (e.g., floor plans).

The rest of this paper is organized as follows. We first review previous work on the visibility complex and other relevant topics. Then, we briefly describe the visibility complex data structure and how it can be implemented in a novel way, i.e., as a cylindrical partition of a set of dual objects. We present the analysis of the construction time of the visibility complex, and also the point-location cost associated with the visibility complex. Then, we describe how the visibility complex can be used to perform ray tracing and present empirical results. In the future works

* Corresponding author.
E-mail address: fscho@cs.berkeley.edu (F.S. Cho)

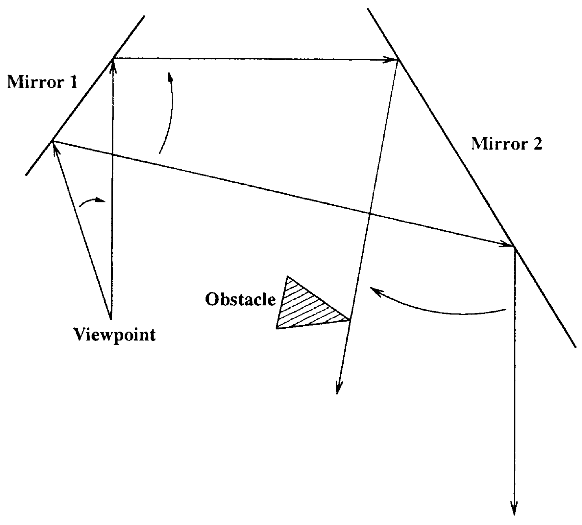


Fig. 1. The set of rays are swept until one of the rays encounters an obstacle.

section, we describe novel techniques to reduce the size of the visibility complex.

2. Previous work

Speeding up ray tracing typically involves either representing a group of objects in a hierarchy of bounding boxes, or decomposing the free space into a quadtree/octree or a *kd*-tree that can be traversed efficiently [1]. Since these data structures reside in the primary space, they do not directly capture the coherence among the rays. For example, we define a contiguous set of eye rays to be equivalent if their ray trees are topologically equivalent. In primary space, it is difficult to represent this equivalence class exactly, and it is almost always represented conservatively.

Arvo and Kirk's ray classification algorithm [2] is closer in spirit to our algorithm. This algorithm associates a 5-tuple with each ray, decomposes the 5D space into hypercubes, and then associates each hypercube with a set of candidate objects. However, decomposing the 5D dual space into hypercubes can be inefficient, since the boundary at which the visible object changes typically does not lie along the hypercube boundary. Ray re-classification technique is reminiscent of the screen (Section 7); both techniques reduce memory requirement at the expense of potentially performing multiple queries per ray.

Beam tracing [3] takes advantage of ray coherence by grouping a set of equivalent rays into a "beam". The main drawback of this algorithm is that it has to perform object-space polygon clipping algorithm every time a beam encounters an obstacle. These beams quickly break into many small beams, and each beam may have a complicated shape.

A popular method of generating mirrored images is to reflect the viewpoint across the mirror, render the scene from the virtual viewpoint and map the view onto the mirror [4–6]. Refraction can be approximated in a similar way. One drawback of this method is that the algorithm must keep track of which pixels correspond to which mirror, which is potentially costly. Also, these algorithms typically over-render the reflected/refracted scene. This technique achieves a substantial speed up over classical ray tracing, but its rendering speed has not yet reached interactive rates. Another popular method of rendering mirrors is to create a reflected copy of the environment across the mirror and render the scene, appropriately clipped [6]. This method is surprisingly tricky to implement (e.g., objects behind the mirror must not appear in the mirror's reflection and reflected object must not appear in front or to the side of the mirror, all of which must be handled by clipping in the rendering engine), and handles inter-reflecting mirrors poorly.

In [7], Ofek and Rappoport discuss how to generate scenes containing reflections on curved objects at interactive rates. Potentially reflected scene objects are tessellated and a virtual vertex is computed for each resulting scene vertex. Ofek and Rappoport report rendering time per frame to be less than 1 second per frame, although the number of reflectors is typically very small (less than 5).

In [8], Teller and Alex attempt to perform ray casting in real time through frustum casting. This algorithm is a combination of the beam tracing technique [3] and Warnock's algorithm [9] where the frustum is recursively subdivided until either no element intersects the frustum, one element obscures all other elements in the frustum, or the frustum covers one pixel. This technique produces maximum frame rate of 5 frames/s at 129×129 maximum resolution, with only primary and shadow rays cast. The scenes are very simple, consisting of a building floorplan where each wall meets the floor and the ceiling.

In [10], Bala et al. describe an interactive ray-tracing algorithm in 3D. In this algorithm, radiance values are lazily collected into linetrees, and these collected values are quadrilinearly interpolated to approximate the radiance value with a guaranteed error bound. These linetrees are reprojected as the user's viewpoint changes in order to accelerate the visibility determination at a pixel. This algorithm renders scenes at minimum 20–30 s per frame, and uses a lot of memory in order to cache the radiance values (around 100 Mbytes).

A survey of ray-shooting algorithms in 2D, 3D and higher dimensions can be found in [11,12]. In [13], Pocchiola and Vegter present a 2D ray-shooting algorithm which runs in $O(\log m)$ time using $O(m + k)$ space, where the environment consists of disjoint convex objects with m simples arcs in total. Here, k is the size of the visibility complex. This algorithm involves performing point-location queries on planar sub-complexes of the visibility complex, where the (directed) free bitangents of the pseudo-triangulation act as one-sided obstacles. In [14], Szirmay-Kalos and Márton prove that given n objects in 3D, any ray-shooting algorithm has worst-case running time of $\Omega(\log n)$ and any logarithmic algorithm must use at least $\Omega(n^4)$ storage and pre-processing time. One way of answering ray-shooting queries in 3D is to perform point-location on a data structure in Plücker dual space. Examples of these algorithms can be found in [15,16]

The original paper on the visibility complex by Pocchiola and Vegter describes how the visibility complex of curved convex objects can be constructed in $O(n \log n + k)$ optimal time and $O(k)$ working set, where n is the number of objects and k is the number of free bitangents among the objects, which is proportional to the size of the visibility complex [17]. In [18], Pocchiola and Vegter present a practical construction algorithm for the visibility complex which runs in $O(n \log n + k)$ time and $O(n)$ space, using greedy pseudo-triangulation. (The environment consists of n disjoint convex obstacles of constant complexity.)

In [19,20], Rivière presents a practical construction algorithm for the visibility complex of convex polygonal objects using topological sweep. This algorithm runs in $O(n \log n + k)$ time. In [21], Rivière describes how to maintain the view along a trajectory, given an initial view at the beginning of the trajectory. In [19,21], Rivière hardly mentions anything about performing point-location in the visibility complex. As we will see in this paper, point-location plays a crucial role in handling reflected/refracted rays.

In [22], Durand et al. present the first attempt at building a visibility complex of 3D objects. This algorithm utilizes a dualization scheme where the region boundaries in the dual space are curved surfaces, which complicates the implementation. To our knowledge, no implementation of this algorithm exists. In [23], Durand et al. describes the visibility skeleton, which comprises the 0-faces and the 1-faces of the 3D visibility complex. This data structure can be used to solve various problems involving visibility computations, including discontinuity meshing. Since this data structure is restricted to only zero and one dimensional elements, we cannot compute the view from an arbitrary view point using this data structure.

3. The visibility complex

In this section, we briefly describe the visibility complex data structure and how we represent the visibility complex in a novel way, i.e., as the cylindrical partition of a set of "sheets".

We represent lines as points in the projective dual of the projective plane, i.e., the line $Ax + By + C = 0$ in the primary space maps to the point (A, B, C) in the dual space in homogeneous coordinates. This dualization has a geometric interpretation shown in Fig. 2. As this geometric interpretation illustrates, as a line approaches the origin, its dual point approaches the point at infinity. (In [19], Rivière uses a slightly different dualization scheme where vertical lines map to the points at infinity.) This dualization scheme has many useful properties. For example, this mapping works backwards as well; the family of lines that pivot around the point $(A/C, B/C)$ in the primary space maps to the line $Ax + By + C = 0$ in the dual space. This representation has the advantage that the family of lines passing through plane polygons becomes a polygon and the family of lines through a point becomes a line segment, so the geometry in the dual space is easy to represent and use.

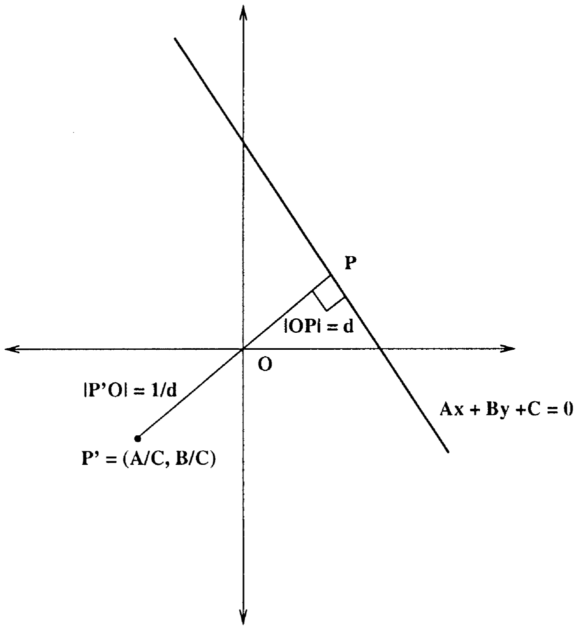


Fig. 2. This figure illustrates the geometric interpretation of the projective dualization. Let P be the closest point to the origin that lies on the line $Ax + By + C = 0$. Let $|OP| = d$. The point dual to this line is $1/d$ units away from the origin, and lies on the line $\overline{OP}$ on the opposite side of the origin.

Let L_O be the set of lines that intersect the object O. If we dualize every line $l \in L_O$, the set of points $\{D(l) | l \in L_O\}$ forms a "sheet" S_O in the dual space (Fig. 3).

Now, the set of rays leaving O may see a number of different objects. We divide the sheet S_O into different "regions" depending on which object the ray sees. For example, in Fig. 4, the set of rays seeing O_2 is collected into the region labeled O_2, and the rest of the rays that do not see anything map to a region labeled "BS", which stands for the "blue sky".

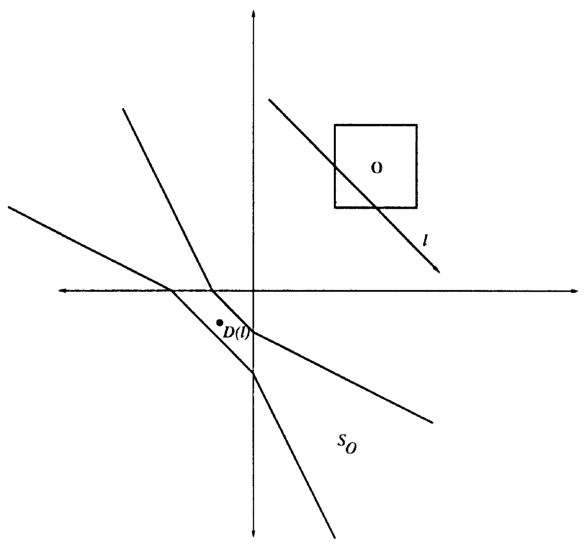


Fig. 3. "Sheet" S_O in the dual space corresponding to the set of lines interesting the object O in the primary space.

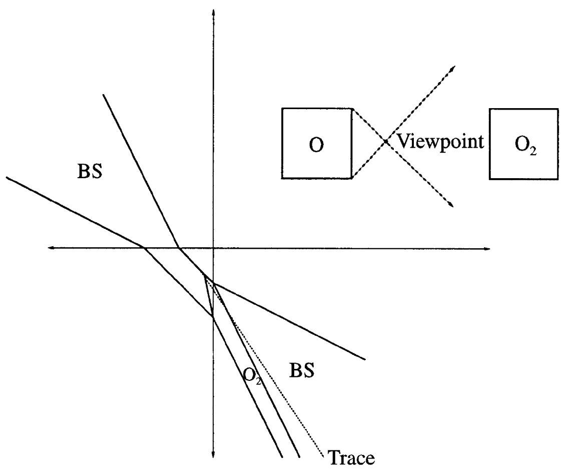


Fig. 4. The set of points on the sheet S_O are separated into different "regions" according to the object visible along the corresponding ray. When viewing rays are swept around a viewpoint, the dualization of the viewing rays form a line segment in the dual space. The set of regions encountered by these dual points correspond to the view around the viewpoint.

Fig. 4 also shows a viewpoint, and the set of viewing rays that pivot around the viewpoint. The dualization of these rays correspond to a line segment in the dual space (marked "trace" in the figure). We can therefore easily compute the view around a view-point by "walking" along the trace on a sheet, and noting which regions are encountered.

Suppose ray r_1 leaves an object O along the line $Ax + By + C = 0$, and r_2 leaves O in the opposite direction along the same line. Notice that r_1 and r_2 dualize to the same coordinates, $(A/C, B/C)$. To represent the dual point of r_1 separately from the dual point of r_2, we associate r_1 and r_2 with different sides of the sheet. In our representation, the rays that are directed clockwise

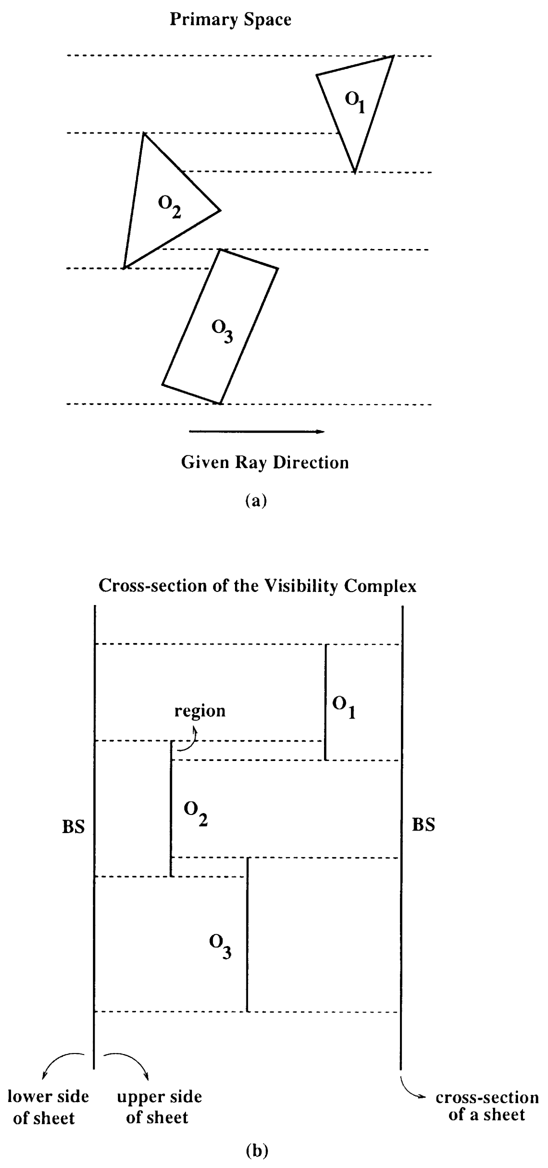


Fig. 5. (a) Given a fixed ray direction, the free space can be decomposed into areas of constant forward and backward views. (b) The corresponding cross-section of the visibililty complex. The solid line segments correspond to the cross-section of a sheet. Each side of a sheet is decomposed into separate regions. Each area in (a) corresponds to a unique region in (b).

(counter-clockwise) with respect to the origin map to the upper (lower) side of the sheet. Therefore, the upper and the lower side of a sheet are divided into separate sets of regions. These sets of regions (organized according to the sheet to which they belong) and their adjacency relationships define the visibility complex.

Let us examine a cross-section of the visibility complex by examining a set of parallel rays (Fig. 5). This set of rays dualizes to points along a line perpendicular to the rays that contains the origin (Fig. 2). We define the forward (backward) view of the ray as the object visible from the ray origin along (opposite) the ray direction. Let us visualize the cross-section by examining these viewing rays and see how many distinct forward and backward views these rays can have. Each contiguous set of rays which share the same forward and backward views map to a region in the visibility complex. (A region is equivalent to a face of the visibility complex, as described in [17]). Fig. 5 illustrates this concept.

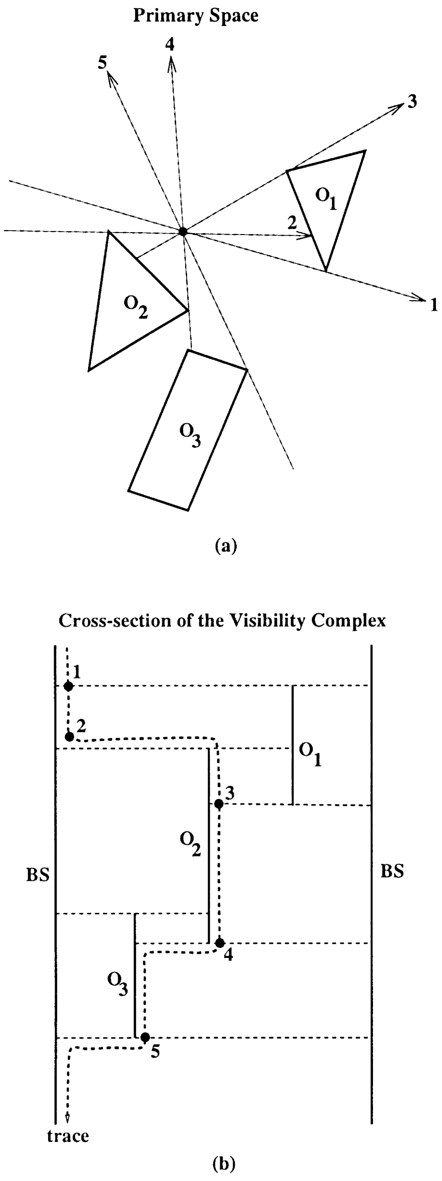


Fig. 6. (a) A viewing ray is swept around a viewpoint. At the marked rays 1, …, 5, the forward or the backward view changes. (b) In the dual space, we can maintain the forward view of the viewing ray by "walking" along the marked trajectory. At the marked point 2, we must decide whether to stay on the same sheet, or to hop onto a different sheet.

As we sweep a viewing ray around a veiwpoint, the forward and backward views change at discrete ray positions (Fig. 6). We can maintain the forward view of the viewing ray by "walking" the visibility complex. Sweeping a viewing ray around the viewpoint corresponds to the dual point translating along a line. Fig. 6 shows the cross-section of the visibililty complex along this line. When the dual point encounters a region boundary, it is sometimes necessary to decide whether to stay on the same sheet, or to hop onto a different sheet (e.g., at point 2 in Fig. 6).

If one examines the cross-sections shown in Figs. 5 and 6, one discovers that it corresponds to a trapezoidal decomposition of a set of line segments. Therefore, the visibility complex can be visualized as a set of "prisms" delimited by a set of sheets, and the cross-section of theses prisms look like Figs. 5(b) and 6(b). If we subdivide each region into trapezoids, the prisms are decomposed into cylinders, and the visibility complex corresponds to a cylindrical partition [24] of the sheets (described in the next section), which is precisely the representation we use in our implementation.

4. Construction algorithm and analysis

This section describes our construction algorithm for the visibility complex. The visibililty complex is constructed in the same way as cylindrical partition. The construction algorithm of the cylindrical partition is loosely based on the construction algorithm given in [24], and the analysis of the running time of the algorithm and the size of the data structure uses many ideas found in [24].

Fig. 7 illustrates cylindrical partition. A set of polygons are defined in Fig. 7(a). To construct a cylindrical partition of these polygons along the z-direction (coming out of the page), we first construct the *primary walls* as follows. From every point that lies on a polygon edge, we vertically extend a line segment in both positive and negative z-directions until the line segment encounters the first polygon. If no polygon stops the line segment, it is vertically extended forever. The union of these vertical segments defines the primary walls. Notice that the primary walls divide the upper (lower) side of each polygon into seperate regions based on which polygon is directly

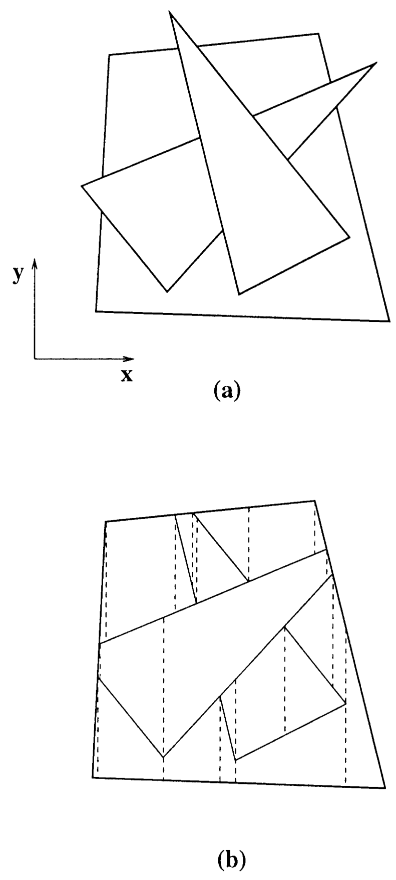


Fig. 7. Cylindrical partition: (a) a set of polygons are defined. After we define a cylindrical partition of these polygons along the z-direction (coming out of the page), the upper side of the quadrilateral is shown in (b).

above (below) each point. Each of these regions uniquely belongs to a parallelepiped extruded along the z-axis. We further refine each parallelepiped by performing trapezoidal decomposition on the region along the y-direction. These walls parallel to the yz-plane are called *secondary walls*. Fig. 7(b) shows the upper side of the quadrilateral after the primary and secondary walls have been constructed. The primary walls are shown in solid lines, and the secondary walls are shown in dashed lines. Now, each parallelepiped has been subdivided into a set of "cylinders". These cylinders and the adjacency relationships among them define the cylindrical partition. For further details, the reader is referred to [24].

4.1. Constructing the visibility complex

To construct the visibility complex, the visibility complex is initialized with a cylinder H_{Initial} delimited by two (infinite) sheets corresponding to the blue sky. In practice, we clip every sheet to a bounding box. If we restrict the viewpoint (or the virtual viewpoint, as explained in the next section) to lie within a bounding box in the primary space, then we can calculate the minimum size of the dual bounding box with the following property: whatever lies outside the dual bounding box will map to less than a pixel in the final rendered image.

Once the visibility complex is initialized, we incrementally add the sheets in a random order. Let N^i denote the set of objects we have added so far, and let $H(N^i)$ denote the cylindrical partition of N^i. After adding the sheet N, the cylindrical partition will be updated to $H(N^{i+1})$ where $N^{i+1} = \{N^i \cup N\}$. The following pseudo-code explains how $H(N^i)$ is updated to $H(N^{i+1})$. ∂N represents the sheet boundary of N:

```
Update Cylindrical Partition ($H(N^i)$, $N$){

  Find the cylinder containing the first vertex of $\partial N$
     "Travel and split" along $\partial N$
     Mark the cylinders lying inside $\partial N$
     Split the marked cylinders into upper and lower
     cylinder
     \\ Process the upper cylinders
     Mark the region boundaries to be deleted
     Delete the marked region boundaries in a random
     order
     \\Process the lower cylinders
     Mark the region boundaries to be deleted
     Delete the marked region boundaries in a random
     order
     Return ($H(N^{i+1})$)
  }
```

When the sheet N is added, we first perform point-location (shown later) to find the cylinder containing the first vertex of the sheet boundary ∂N, and "travel and split" along ∂N while maintaining the cylindrical partition. After "travel and split", we mark the cylinders lying inside ∂N, and split them into two (one adjacent to the upper side of N and one adjacent to the lower side of N). Let $H'(N^i)$ denote the cylindrical partition existing at this point. If we look at the upper side of N (the lower side is handled in exactly the same way), some of the region boundaries must be deleted, and the cylinders adjacent to deleted edges must be updated accordingly. In order to update these cylinders, we mark the region boundaries to be deleted and delete them in a random order. Deleting a region boundary is analogous to deleting a line segment from trapezoidal decomposition, as described in [24]. When all the marked region boundaries are deleted, we obtain $H(N^{i+1})$.

4.2. Point location

One important by-product of the randomized incremental construction is that we obtain an efficient point-location data structure by recording the history of construction. (An efficient point-location query is crucial to ray tracing application, as shown in the next section). Let n be the number of sheets to be inserted and let us assume that each sheet has a bounded number of edges (i.e., the corresponding polygon in the primary space has a bounded number of edges). For every cylinder that has ever existed during the incremental construction sequence, we create a node in the history. If a cylinder is "split" by an inserted sheet edge into $O(1)$ child cylinders during the incremental construction, we create links from the node corresponding to the split cylinder to the nodes corresponding to the child cylinders. Also, if one of the region boundaries adjacent to a cylinder is deleted, the cylinder is merged into $O(n)$ child cylinders, and we create links from the corresponding node to the child nodes.

The following pseudo-code outlines how the point location is performed. To find the cyllinder containing the query point p, we call Point-Location (InitialNodeIn-History, p)

```
Point-Location(Node, $p$) {
    if (Node has no child node) {
        Return(Node)
    } else {
        ChildNode = child node containing $p$
        Return(Point-Location(ChildNode, $p$))
    }
}
```

To determine the cylinder containing the query point, we start from the initial node in history, and follow the child links (i.e., determine the child cylinder that contains the query point) until a node corresponding to an undestroyed cylinder is reached. If the current node is linked to $O(n)$ child nodes, we can perform a binary search among the list of child nodes in $O(\log n)$ time since the child cylinders can be uniquely ordered from left-to-right. This point-location process takes $O(\log^3(n))$ average time if we apply the standard analysis found in [24,25]. (We can also prove that this average bound holds with high probability [24]).

4.3. Analysis

We will now present the bound on the average construction time. Let us define the size of the structural change from $H(N^i)$ to $H(N^{i+1})$ as the sum of the face-lengths of the destroyed cylinders $f \in H(N^i)$ and the face-lengths of the newly created cylinders $g \in H(N^{i+1})$. Here, face-length denotes the number of faces in all dimensions. The total structural change denotes the sum of the structural changes over the entire insertion sequence.

Given a fixed insertion sequence, the above construction algorithm runs in

$$O(n \log^3(n) + (\text{total structural change}) \cdot \log(n))$$

average time. Following the derivation found in [24], the expected size of total structural change is

$$E[\text{size of total structural change}] = O\left(\sum_{j=1}^{n} \frac{E[|H(N^j)|]}{j}\right)$$

where N^j is a random subset of j sheets and $|H(N^j)| = \sum_{g \in H(N^j)} \text{face-length}(g)$. Note that $|H(N^j)| = \Theta(j^2)$ in the worst case, in which case the expected size of total structural change is bounded by $\sum_j \Theta(j) = \Theta(n^2)$. In practice, $|H(N^j)|$ may be sub-quadratic and the average time complexity of the construction algorithm may be much better than the worst case.

Similarly, we can derive the expected size of history (i.e., total number of nodes and links) as

$$O\left(\sum_{j=1}^{n} \frac{E[\text{number of cylinders in } H(N^j)]}{j}\right)$$

times $\log(n)$, with N^j defined as above. This average size is no worse than $O(n^2 \log n)$.

4.4. Search path compaction

As we noted previously, point-location is performed in $O(\log^3(n))$ average time. We have developed a technique to "compact" the search path so that point-location can be performed in $O(\log^2(n))$ time. This technique works as follows. If we examine the construction algorithm and its history, we can see that we can create links directly from the node corresponding to the (destroyed) cylinder in $H'(N^i)$ to the nodes corresponding to its child cylinders in $H(N^{i+1})$. A cylinder $C_{\text{child}} \in H(N^{i+1})$ is a child of a cylinder $C_{\text{parent}} \in H'(N^i)$ if they overlap in space. Note that these child cylinders can be uniquely ordered from left-to-right. We call this technique "search path compaction", which decreases the point-location cost to $O(\log^2(n))$ average time. This average query time holds with high probability. The cost we pay for smaller query time is that the number of links is no longer linearly proportional to the number of nodes. Therefore, the $O(n^2 \log n)$ upper bound on the expected size of history no longer holds when search path compaction is applied. As we will demonstrate in Section 7, in practice the size of the history typically decreases by a small percentage by performing search path compaction.

4.5. Degeneracy

Our current implementation does not work well with degenerate configurations: we require that the input polygons be disjoint and no three vertices of the input polygons may be collinear. This does not mean that the visibility complex inherently cannot handle these degeneracies, however. There are techniques to treat the degeneracies arising from inaccuracies of numerical calculations [19]. Also, it is possible to handle input polygons that overlap. Suppose polygons P_1 and P_2 overlap over the region $P_1 \cap P_2$. In the dual space, this configuration can be handled by fusing the sheets $D(P_1)$ and $D(P_2)$ along the dual region $D(P_1 \cap P_2)$ which must be handled specially. Incorporating these techniques into our implementation remains a future work.

5. Ray tracing with the visibility complex

When the visibility complex is walked to produce the current view, three major classes of rays must be handled separately, namely, primary rays, reflected secondary rays and refracted secondary rays. Concurrently sweeping the primary ray and the secondary rays (as shown in Fig. 1) can be achieved by maintaining a sorted array where each element corresponds to a (primary or secondary) ray. The array is maintained in a sorted order according to the generation of the ray, i.e., the primary ray corresponds to the first element of the array, the second-generation ray corresponds to the second element, and so on. Each element of the array maintains its current position in the visibility complex, as well as its trace (the direction along which the dual point is walked).

5.1. Primary rays and reflected rays

As shown in Section 3, sweeping the primary ray around the viewpoint $(A_{vp}/C_{vp}, B_{vp}/C_{vp})$ corresponds to walking along the trace

$$A_{vp}x + B_{vp}y + C_{vp} = 0$$

in the dual space. To sweep a reflected secondary ray, we reflect the view point across the mirror to generate a virtual viewpoint at $(A_{vvp}/C_{vvp}, B_{vvp}/C_{vvp})$ (Fig. 8(a)). The trace corresponding to this reflected secondary ray is along

$$A_{vvp}x + B_{vvp}y + C_{vvp} = 0.$$

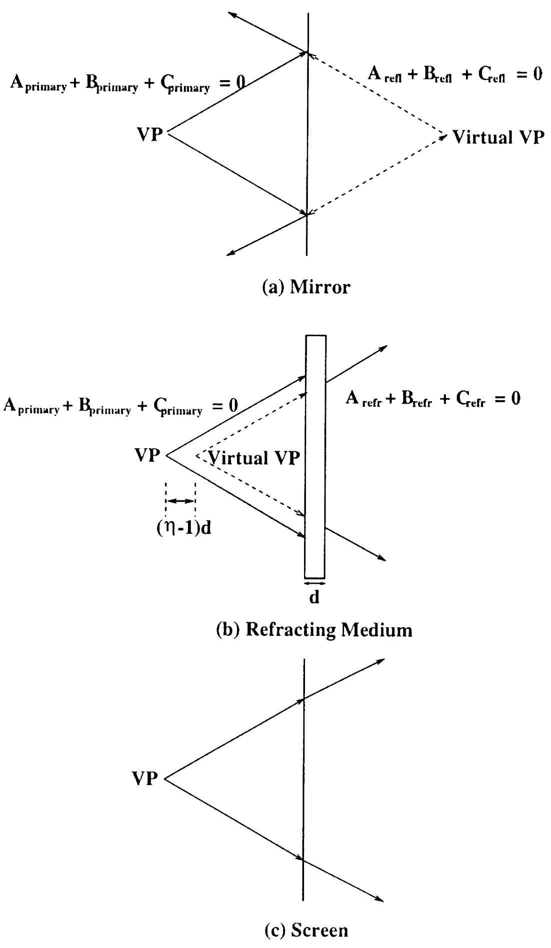
Suppose a primary ray is swept around the viewpoint and encounters a reflecting medium when its orientation in the primary space is

$$A_{primary}x + B_{primary}y + C_{primary} = 0$$

(Fig. 8(a)). Then, we generate the reflected secondary ray at

$$A_{refl}x + B_{refl}y + C_{refl} = 0,$$

determine which cylinder contains the dual point $(A_{refl}/C_{refl}, B_{refl}/C_{refl})$ in the visibility complex, and start walking the new trace from this point. In order to determine which cylinder contains $(A_{refl}/C_{refl}, B_{refl}/C_{refl})$, we use the point-location data structure outlined in the previous section.



$A_{primary} + B_{primary} + C_{primary} = 0$
$A_{refl} + B_{refl} + C_{refl} = 0$
VP
Virtual VP
(a) Mirror
$A_{primary} + B_{primary} + C_{primary} = 0$
$A_{refr} + B_{refr} + C_{refr} = 0$
VP
Virtual VP
$(\eta-1)d$
d
(b) Refracting Medium
VP
(c) Screen

Fig. 8. Different types of transmitting media and the corresponding virtual viewpoints are illustrated. In (a), the line $A_{primary}x + B_{primary}y + C_{primary} = 0$, which is the initial line of the sweep, is reflected to form $A_{refl}x + B_{refl}y + C_{refl} = 0$. In order to sweep the secondary reflected ray, we perform point-location on the dual point $(A_{refl}/C_{refl}, B_{refl}/C_{refl})$ to find the cylinder which contains this point, and walk the sheets starting from that cylinder. In (b), the initial line $A_{primary}x + B_{primary}y + C_{primary} = 0$ is refracted to form $A_{refr}x + B_{refr}y + C_{refr} = 0$. We again perform point-location on the dual point $(A_{refr}/C_{refr}, B_{refr}/C_{refl})$ to find the cylinder which contains the point, and walk the sheets starting from that cylinder. In (c), a screen is shown, which directly passes the light without altering it. A screen, although visually uninteresting, plays an important role in reducing the size of the visibility complex.

5.2. Refracted rays

To sweep a refracted secondary ray, we generate a virtual veiwpoint according to the tangent law [3] which closely approximates Snell's law when the incidence angle is small. The tangent law states that

$$\eta_1 \tan \theta_1 = \eta_2 \tan \theta_2,$$

where θ_1 is the angle of incidence, θ_2 is the angle of refraction, and η_1 and η_2 are the indices of refraction. Using the tangent law, the refracted rays meet at a virtual focus. When the viewpoint is at a distance D_1 units away from the medium, the virtual focus is $D_2 = 1/\eta D_1$ units away from the medium, where $\eta = \eta_1/\eta_2$ (relative index of refraction), and the line joining the viewpoint and the virtual focus is perpendicular to the plane of the medium.

Using the tangent law, we can calculate the position of the virtual veiwpoint. If the refracting medium is of thickness d, then the virtual viewpoint is translated $(\eta - 1)d$ units closer to the plane of the refracting medium, along the direction perpendicular to this plane. To simplify the implementation, we have restricted the viewpoint to lie outside the refracting medium. When the angle of incidence becomes large, the tangent law no longer closely approximates Snell's law; we currently use the tangent law for all angles of incidence. Correctly handling (or more closely approximating) refraction in this case remains a future work.

Refracting media are handled in the same way as reflecting media (Fig. 8(b)). When a primary ray encounters a refracing medium at

$$A_{\text{primary}}x + B_{\text{primary}}y + C_{\text{primary}} = 0$$

in the primary space, we generate the refracted secondary ray at

$$A_{\text{refr}}x + B_{\text{refr}}y + C_{\text{refr}} = 0,$$

determine which cylinder contains the dual point $(A_{\text{refr}}/C_{\text{refr}}, B_{\text{refr}}/C_{\text{refr}})$ in the visibility complex, and start walking the new trace from that point. Again, the point-location data structure is used to determine which cylinder contains $(A_{\text{refr}}/C_{\text{refr}}, B_{\text{refr}}/C_{\text{refr}})$.

Once the virtual viewpoint $(A_{\text{vvp}}/C_{\text{vvp}}, B_{\text{vvp}}/C_{\text{vvp}})$ is created for the refracted ray, the trace lies along

$$A_{\text{vvp}} \cdot x + B_{\text{vvp}} \cdot y + C_{\text{vvp}} = 0$$

in the same way as a reflected secondary ray (Fig. 8(b)).

Although it is theoretically possible to use Snell's law in order to handle refraction, using Snell's law has several drawbacks. First of all, the trace of the refracted ray in dual space lies along a curve rather than a line, which complicates the implementation. Refracting a refracted ray is much more difficult than handling just one refraction. Due to these drawbacks, we decided to use the tangent law in our implementation.

5.3. Screens

Fig. 8(c) illustrates a "screen" which passes light rays without altering them. We can imagine screens as refracting media with zero depth. The significance of screen lies in its ability to reduce the size of the visibility complex and its history, as will be explained in Section 7.

5.4. Walking the visibility complex

Once the array of traces and the dual points are initialized, the dual points are walked in lock step, i.e., at each time step, the dual point with the shortest walk is determined and every dual point walks this shortest distance. A subtle point to notice is that we cannot directly compare these distances in the dual space to find the shortest distance; these distances must be translated into angles before they can be compared. When a dual point encounters a region corresponding to a reflecting/refracting medium, we spawn a new trace and replace the next element in the array with the new trace. Now, we must update the rest of the array by recursively applying the same procedure to the new trace. Once the array is initialized, the dual points are walked in lock step again.

Although our implementation currently does not handle shadow rays, they can be handled in exactly the same way as the reflected/refracted rays: When an eye ray first hits an object, we initialize a shadow ray from the light source pointing toward the intersection. When the primary and secondary eye rays are swept, these shadow rays are swept concurrently. The trace of a shadow ray lies along a line. We can also build a bidirectional ray tracer by sweeping light rays around each light source and recording the radiance values at each intersection. Building a bidirectional ray tracer using the visibility complex is particularly attractive, since the light rays are swept in object-precision and thus obviates the need to deal with sampling issues.

6. Experimental results

In this section, we empirically measure the running time and the memory usage of our algorithm, and present the analyses. For Figs. 9–11 ,the test case used was a set of triangles scattered around the plane (Fig. 20). The plane is divided into N by N imaginary grids, and a triangle is randomly placed within each grid. With this configuration, the size of the visibility complex can potentially grow at $\Theta(n^2)$, since each object may see the maximum number of objects. As Fig. 9 strongly suggests, the growth rate of the size of the visibility complex in this configuration is actually linear. Fig. 9 also shows the size of the history. As we can see, search path compaction actually decreases the size of history by a small percentage in practice. The size of history grows substantially faster than the size of the visibility complex. To combat this high growth in memory usage, we propose techniques to cut down the size of the data structures in Section 7.

Fig. 10 shows the construction time of the visibility complex, plotted against the number of objects. The construction time remains fairly modest at less than 3 min for up to 256 objects. As expected, search path

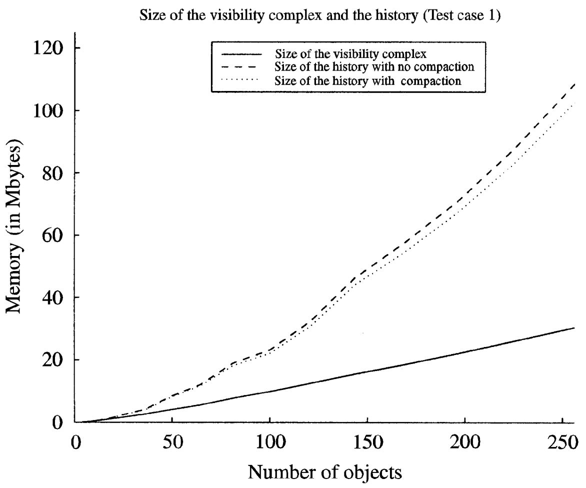


Fig. 9. This graph shows how the size of the visibility complex and its history grows as the number of objects increases. The size of the visibility complex is shown in solid line, the size of the history with no search path compaction is shown in dashed line, and the size of the history with search path compaction is shown in dotted line. As can be seen in this graph, the size of the history dominates the memory usage.

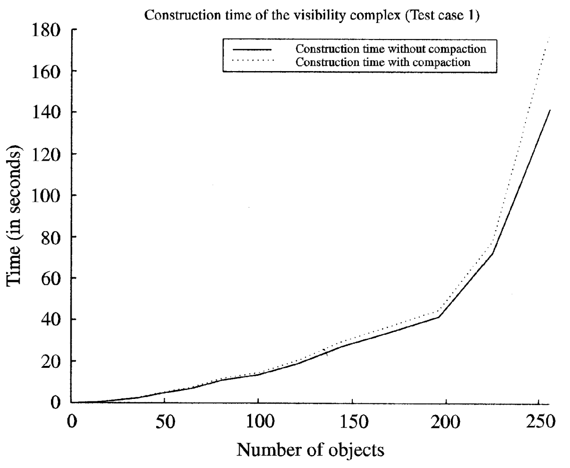


Fig. 10. This graph shows how construction time behaves as the number of objects increases. The construction time with no search path compaction is shown in solid line, and the construction time with search path compaction is shown in dotted line. The time requirement is fairly small, at less than 3 min for 256 objects. The sharp increase in the growth rate around 200 objects can be attributed to thrashing in virtual memory.

compaction adds to the construction time. The amount of overhead is fairly small, however. At around 200 objects, the construction time suddenly increases as a sharper rate. This can be attributed to the size of the data structures exceeding the size of RAM. To generate Figs. 9 and 10, we used a 200 MHz Pentium machine with 128 Mbyte of RAM running Linux.

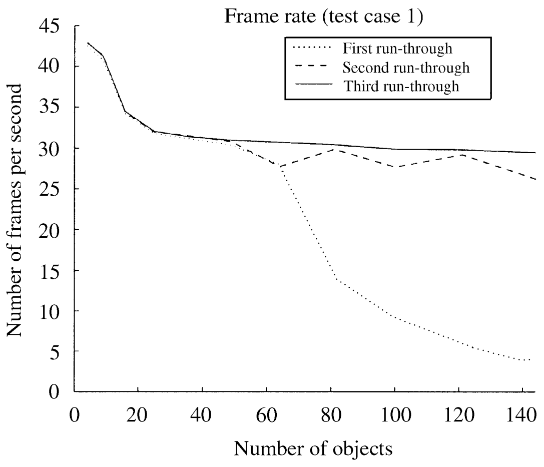


Fig. 11. This graph shows how the frame rate varies as the number of objects increases. For this graph, we rendered scenes along a trajectory 3 times in a row. The frame rate from the first trial is shown in dotted line, the second trial in dashed line, and the third trial in solid line. This increase in frame rate as the trials progress can be attributed to the working set being fetched from the disk into RAM. As can be seen from the graph, the working set is paged out of RAM at around 50 objects. This graph also illustrates that the working set in all cases are fairly small; in all cases, the implementation maintains a frame rate close to 30 frames/s on the third trial.

Fig. 11 shows the rendering rate of our implementation. All of our examples are rendered at 800×800 resolution. The triangles are rendered as extruded columns, as shown in Fig. 20. The triangles are rendered without reflections or refractions. As Fig. 11 shows, the rendering rate depends heavily on whether the working set is currently residing in the main memory. This observation motivates an intelligent pre-fetching strategy for walking the visibility complex, which is left as future work. To generate the rendering rates, an SGI O_2 workstation was used. Fig. 11 suggests that even when the visibility complex and its history take a lot of memory, the working set to render scenes along a trajectory is actually small, since our implementation maintains a frame rate close to 30 frames/s on the third trial for all case.

Fig. 12 describes rendering rate in the hall of mirrors environment (Fig. 21). The rendering time per frame is plotted against the maximum depth of the ray tree. To compare the performance of our implementation against a conventional ray tracer, we have implemented a 2D ray tracer which encodes the set of objects in a quadtree. The quadtree is subdivided until each leaf node contains at most one object. A quadtree leaf node maintains pointers to its four neighbors, and each neighbor is at the same level or higher in the quadtree hierarchy. To walk the quadtree, we determine through which of the four sides we exit the current quadtree leaf node, and examine the

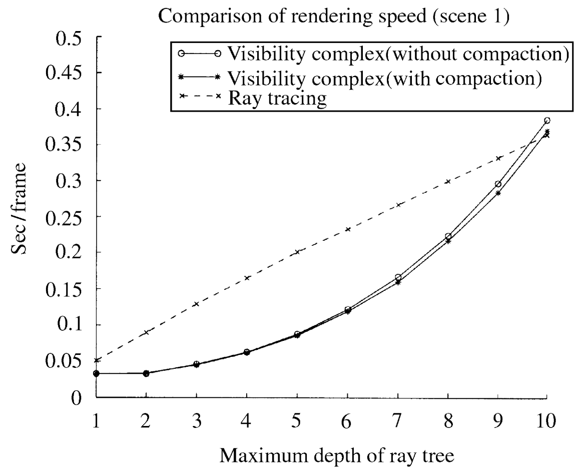


Fig. 12. This graph was obtained by rendering scenes of the "Hall of Mirrors" environment while traveling along a trajectory. The solid curves show the rendering time per frame for the visibility complex, as the depth into the ray tree grows. The upper solid curve was obtained without search path compaction, and the lower solid curve was obtained with search path compaction. Note that the rendering time increases with scene complexity, unlike the case of the conventional ray tracer (dashed curve).

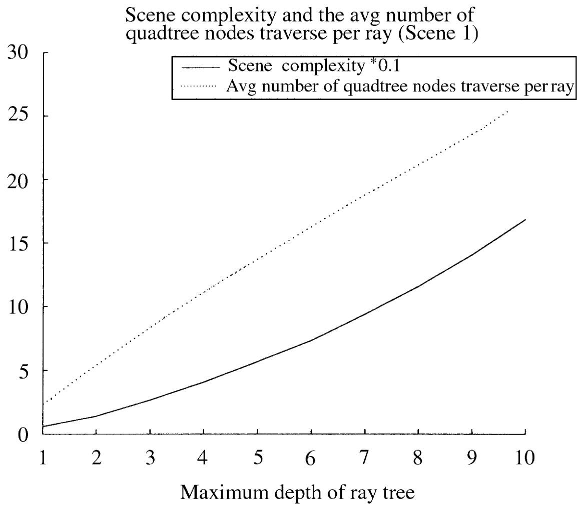


Fig. 13. The average scene complexity is shown in solid line. The average number of quadtree nodes traversed by a ray per frame is shown in solid line.

corresponding neighbor node. If this neighbor node is not a leaf, then we recursively follow the child link until we reach a leaf node. Our aim was to empirically compare the performance of our visibility complex implementation with a standard 2D ray tracer which uses a generic spatial decomposition technique. Thus, our comparison between the performance of our implementation and a standard 2D ray tracer is only a rough one; performance of either implementation can be improved by code optimization and other techniques. Both our implementation and the 2D ray tracer using quadtree rendered frames at 800×800 resolution. Since the conventional ray tracer works in 2D, 800 rays were cast to render each scene. Our implementation using the visibility complex produces images in object precision, which are then rendered at 800×800 resolution.

As shown in Fig. 12, the rendering time of the conventional ray tracer increases linearly with the maximum depth of the ray tree, while the rendering time of the visibility complex grows with the scene complexity. In all but one case, the visibility complex outperforms the conventional ray tracer, by effectively capturing the coherence among the rays.

In Fig. 13, the average scene complexity of the hall of mirrors environment is shown in solid line. The scene complexity of an image is defined as the number of x-coordinates where a visual event happens, i.e., either a primary ray or one of the secondary rays encounters a new object. The average number of quadtree nodes traversed by a ray per frame is shown in dotted line. From this graph, we can see that the running time of our implementation roughly corresponds to the scene complexity, and the running time of the quadtree-based ray tracer roughly corresponds to the average number of quadtree nodes traversed by a ray in the hall of mirrors environment.

Fig. 14 is obtained by walking an enviornment containing screens, mirrors and refracting media (Fig. 22), again at 800×800 resolution. The frame rates are plotted as the number of objects are increased. The maximum depth of the ray tree is set at 5. In this scene, our algorithm runs up to 3.5 times as fast as the classical ray tracer. The visibility complex achieves 10.5 to 14.9 frames/s on an SGI O_2 workstation.

Fig. 15 shows the average scene complexity of the environment containing screens, mirrors and refracting media in solid line. Again, the running time of our implementation roughly corresponds to the scene complexity. The average number of quadtree nodes traversed by a ray per frame is shown in dotted line. In this enviornment, the average number of quadtree nodes shows a weak correlation with the frame rate. Part of the reason is that we must also take into account the processing that happens when a ray intersects an object, i.e., when it intersects a mirror, the orientation of the reflected ray must be calculated, etc.

Fig. 16 describes how the size of the active set changes as the number of objects grow for our ray tracer. The active set is defined to be the set of cylinders that are touched in order to render a frame. This figure illustrates that the size of the active set remains roughly constant as the number of objects are increased.

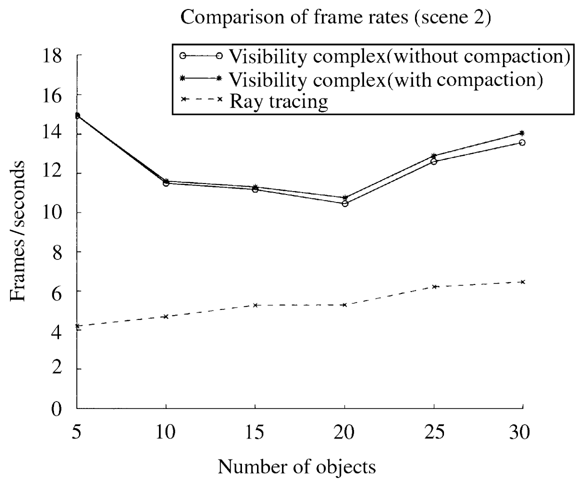


Fig. 14. This graph was obtained by rendering scenes along a trajectory, where the environment contains mirrors, screens and refracting media. As the number of objects grows, the number of frames per second for the visibility complex and the conventional ray tracing remain roughly stable. The visibility complex maintains a frame rate up to 3.5 times that of the conventional ray tracer.

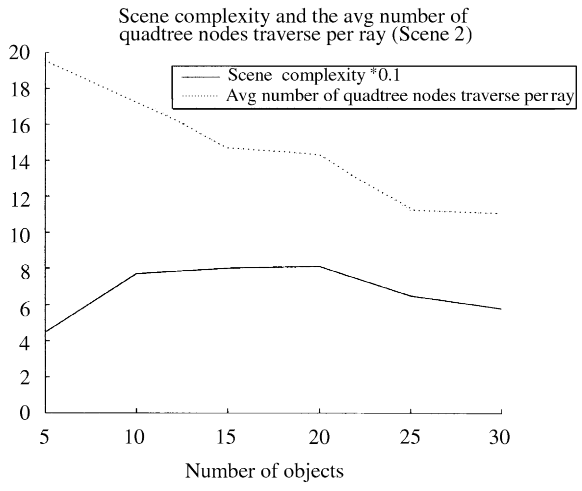


Fig. 15. The average scene complexity is shown in solid line. The average number of quadtree nodes traversed by a ray per frame is shown in solid line.

7. Current and future work

As Fig. 9 shows, the size of the visibility complex and its history is substantial. Therefore, the usefulness of the visibility complex depends on how to effectively cut down the memory usage. We have developed two techniques to decrease the memory usage, namely, placing screens and instancing the visibility complex.

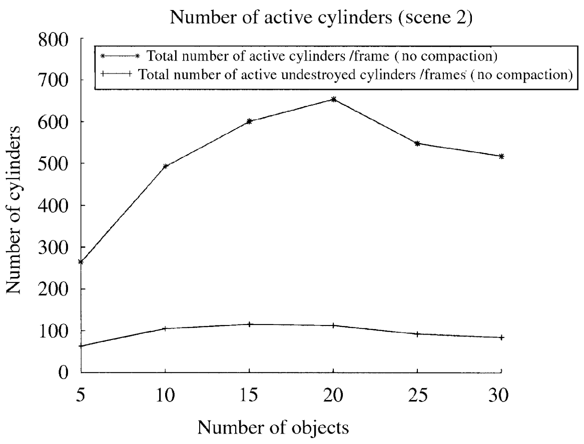


Fig. 16. The asterisks represesnt the average number of cylinders in the history and the visibility complex that are touched to render a frame, plotted against the number of objects. The crosses represent the average number of cylinders in the visibility complex that are touched to render a frame. As can be seen in the graph, the average number of active cylinders is modest and remains stable.

Screens: As we have seen in the previous section, the size of the visibility complex can grow very quickly. As [17] shows, the size of the visibility complex is proportional to the number of free bitangents in the primary space. Given n objects, we can have $\Theta(n^2)$ free bitangents in the worst case. We have also seen that the average size of history can be as big as $O(n^2 \log n)$ (with no search path compaction performed).

However, using a device called a *screen*, we can reduce the size of the visibility complex. Recall that a screen passes light through without altering them, therefore screens are invisible objects that do not change the final rendered image. Since the size of the visibility complex is proportional to the number of free bitangents, placing a screen can reduce the size of the visibility complex by blocking maximum number of free bitangents while introducing minimum number of new free bitangents. Thus, the benefit gained by introducing a screen is

$$(\# \text{ of free bitangents cut}) - |\text{ vertices visible from } v_1| - |\text{ vertices visible from } v_2|,$$

where v_1 and v_2 denote the endpoints of the screen.

In the best case, a screen can block $\Theta(n^2)$ free bitangents while introducing at most $O(n)$ new free bitangents. This happens, for example, when there are two columns of objects each consisting of $n/2$ objects. If each objects in the left column is visible from each object in the right column and vice versa, the number of free bitangents is $\Theta(n^2)$, assuming each object is of bounded size. Placing a screen in between these two columns blocks $\Theta(n^2)$ free bitangents while introducing $\Theta(n)$ new free bitangents.

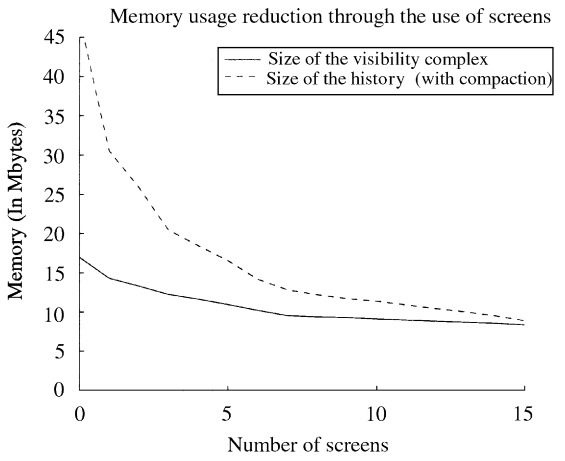


Fig. 17. This figure illustrates how the screens can reduce the size of the visibility complex and its history. As the number of screens increases, the total size of the visibility complex decreases since screens cut more free bitangents than it creates. Screens also reduce the number of free bitangents that happen over the insertion sequence, so the size of the history becomes smaller as well.

A screen can also reduce the size of the history by restricting the number of free bitangents that can arise during the insertion sequence. In order to effectively reduce the size of history, the screens should be placed before the objects. Once the screens are inserted, the set of input objects are inserted in a random order.

Fig. 17 shows the reduction in memory usage through the use of screens. To generate Figs. 17–19, the test case of Fig. 20 was used with 144 triangles arranged in a 12×12 array. The screens are placed along the grid boundaries in a *kd* tree-like fashion. The objects are added after the screens are placed. As the graph shows, introducing screens achieves significant savings in terms of both the size of the visibility complex and the history. Fig. 18 shows that the construction time decreases as well through the introduction of screens. When the screens are introduced, the frame rate typically decreases (Fig. 19).

Currently, the screen locations are chosen by the user. A promising future research direction is develop an algorithm/heuristic to calculate the optimal screen placement. This is currently an active area of research.

Instancing the visibility complex: The second technique involves instancing the visibility complex, i.e., the visibility complex can be defined hierarchically. For example, in order to construct the visibility complex for a floorplan, one needs to only define the visibility complex for one canonical room, and this canonical visibility complex can serve as a representation of every room. The canonical room can undergo arbitrary affine transformation, since the only properties that we require from the transformation is that it preserves incidence and collinearity. We are also actively pursuing this line of research.

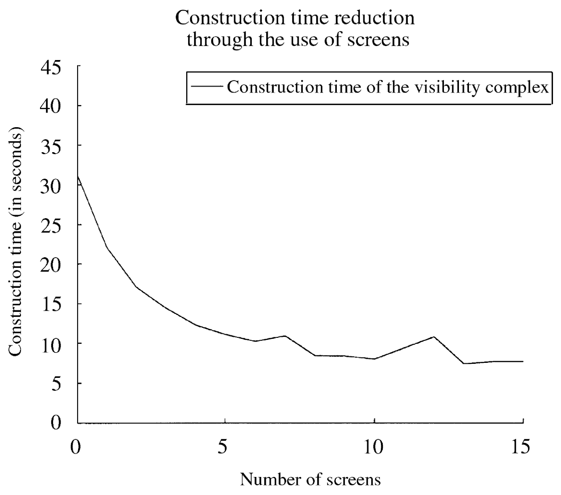


Fig. 18. This figure illustrates that the construction time of the visibility complex decreases as more screens are introduced. Since screens block many bitangents through the insertion sequence, the screens prevent the construction of many vertices over the insertion sequence, and therefore improve the construction time.

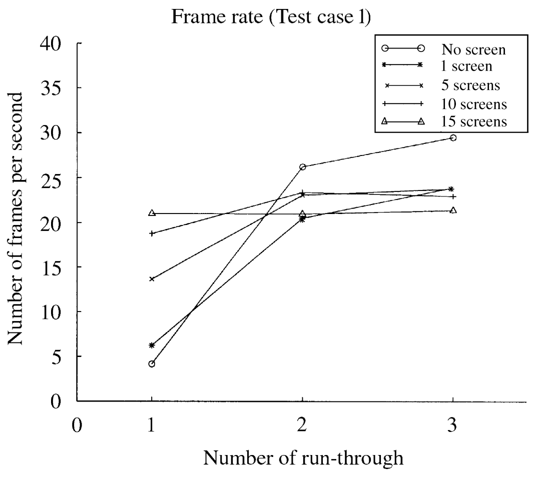


Fig. 19. The frame rate typically decreases as the screens are introduces, since the rendering algorithm must sweep several rays concurrently, rather than just the primary ray. As Fig. 11 also illustrates, the algorithm displays a degree of cache coherence, because the frame rates are higher at the third run-through than the first run-through. When screens are placed, the size of the visibility complex becomes smaller, so the frame rates remain relatively stable among the first, second and the third run-throughs. When no screen is inserted, the size of the visibility complex is much larger, so the frame rate varies a lot among the first, second and the third run-throughs.

Other improvements: Firstly, there is no need to perform point-location at either specular reflections or refractions; instead, the point-location can be performed by

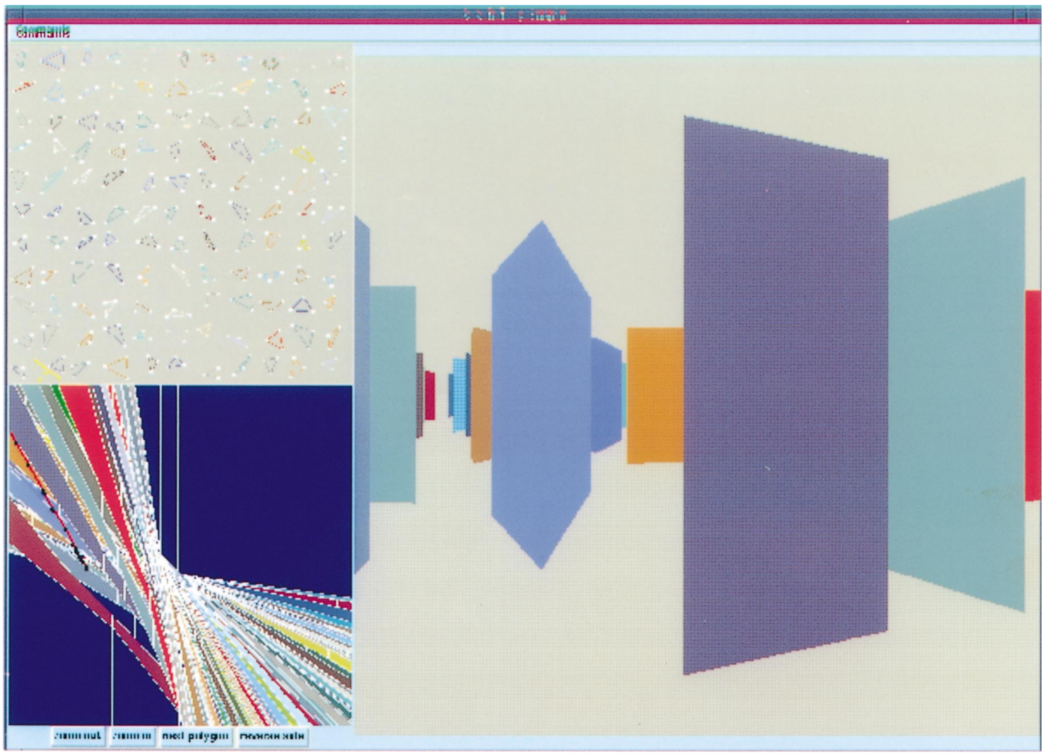


Fig. 20. The user interface of our implementation. The view from the current viewpoint is shown on the right, and the bird's eye view is shown on the upper left corner. Lower left corner shows the visibility complex with the trace highlighted in red.

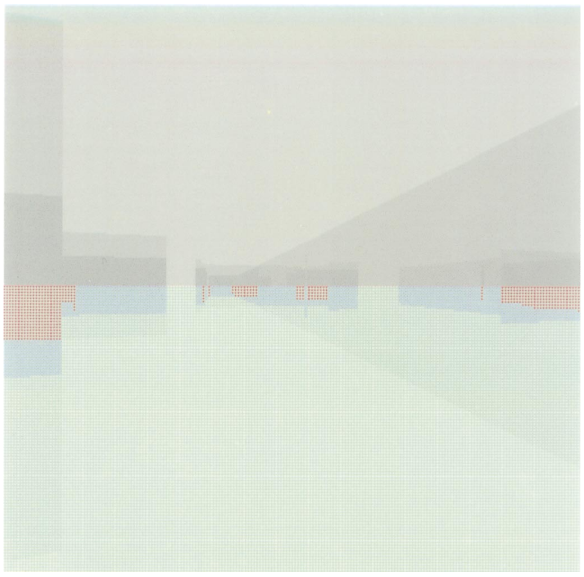


Fig. 21. Snapshot of the "Hall of Mirrors" test environment.

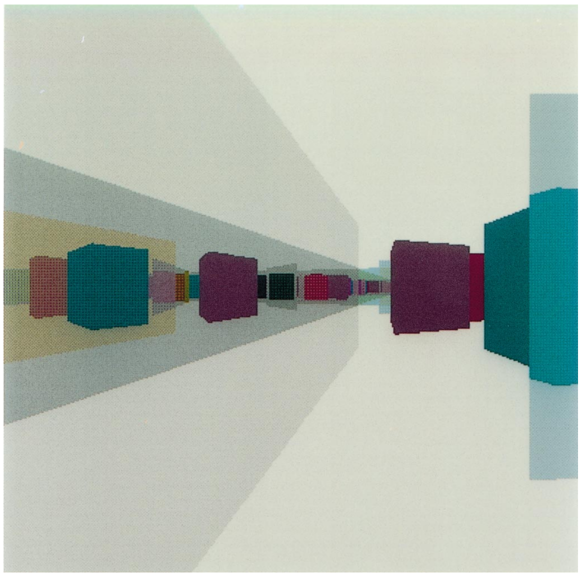


Fig. 22. Snapshot of the test environment containing mirrors, screens and refract.

locally walking the complex. This technique may lead to an improvement in rendering rate. Secondly, all the calculations we have demonstrated apply to illumination effects as well; the visibility complex should yield an efficient forward and bidirectional ray tracer. The bidirectional ray tracing using the visibility complex is attractive since all the shading information is calculated in object precision, thereby avoiding sampling problems. Thirdly, the visibility complex interacts well with line-based representations of objects (like the lumigraph of [26,27]); inserting such an object into our world involves not much more than pointer indirections. Finally, a dynamic radiosity algorithm using the visibility complex is described in [28]. It will be interesting to combine our ray-tracing implementation with radiosity computation to generate more physically accurate renderings.

8. Conclusion

In this paper, we have described a novel construction algorithm of the visibility complex, which yields an efficient point-location data structure as a by-product. Then, an algorithm is described which can generate ray-traced images by concurrently walking the visibility complex along several trajectories. We empirically measured the performance of our implementation, and compared it to a conventional ray tracer with quadtree-based space decomposition scheme. Then, we have presented promising future research directions, including screens and instancing the visibility complex. These techniques can potentially give us large savings in memory usage.

References

[1] Foley JD, van Dam A, Feiner SK, Hughes JF. Computer graphics: principles and practice. Reading, MA: Addison-Wesley, 1996.

[2] Arvo J, Kirk D. Fast ray tracing by ray classification. Proceedings of SIGGRAPH 87. p. 55–64.

[3] Heckbert PS, Hanrahan P. Beam tracing polygonal objects. Proceedings of SIGGRAPH 84. p. 119–127.

[4] Diefenbach P, Badler N. Multi-pass pipeline rendering: realism for dynamic environments. Proceedings of the 1997 Symposium on Interactive 3D Graphics, 1997. p. 59–70.

[5] Diefenbach P, Badler N. Pipeline rendering: interactive refractions, reflections, and shadows. Displays (special issue on interactive computer graphics) 1994;15(3):173–80.

[6] Luebke D, Georges C. Portals and mirrors: simple, fast evaluation of potentially visible sets. Proceedings of the 1995 Symposium on Interactive 3D Graphics, 1995. p. 105–6.

[7] Ofek E, Rappoport A. Interactive reflections on curved objects. Proceedings of SIGGRAPH 98. p. 333–42.

[8] Teller S, Alex J. Frustum casting for progressive, interactive rendering. MIT Laboratory for Computer Science Technical Report 740.

[9] Warnock J. A hidden-surface algorithm for computer generated half-tone pictures. Tech Rep TR 4-15, NTIS AD-733 671, University of Utah, Computer Science Department, 1969.

[10] Bala K, Dorsey J, Teller S. Bounded-error interactive ray tracing. ACM Transactions on Graphics, to appear.

[11] Goodman JE, O'Rourke J. Handbook of discrete and computational geometry. Boca Raton, New York: CRC Press, 1997.

[12] Agarwal PK, Erickson J. Geometric range search and its relatives. Tech report CS-1997-11, Dept. of Computer Science, Duke University, 1997.

[13] Pocchiola M, Vegter G. Pseudo-triangulations: theory and applications. Proceedings of 12th Annual, ACM Symposium on Computational Geometry, 1996. p. 291–300.

[14] Szirmay-Kalos L, Márton G. Analysis and construction of worst-case optimal ray shooting algorithms. Computers and Graphics 1998;22(2–3):167–74.

[15] de Berg M. Efficient algorithms for ray shooting and hidden surface removal. Ph.D. thesis Rijksuniversiteit te Utrecht, The Nederlands, 1992.

[16] Pellegrini M. Ray shooting on triangles in 3-space. Algorithmica 1993;9:471–94.

[17] Pocchiola M, Vegter G. The visibility complex. Proceedings of International Journal of Computational Geometry and Applications 1996;6(3):279–308. Teaneck, NJ: World Scientific.

[18] Pocchiola M, Vegter G. Topologically sweeping visibility complexes via pseudo-triangulation. Discrete and computational geometry 1996;16:419–53.

[19] Rivière S. Visibility computations in 2D polygonal scenes. Ph.D. thesis, Université Joseph Fourier, Grenoble I.

[20] Rivière S. Topologically sweeping the visibility complex of polygonal scenes. Proceedings of 11th Annual ACM Symposium on Computational Geometry, 1995. p. 36–37.

[21] Rivière S. Walking in the visibility complex with applications to visibililty polygons and dynamic visibility. Proceedings of 9th Canadian Conference on Computational Geometry, 1997.

[22] Durand F, Drettakis G, Puech C. The 3D visibility complex: a new approach to the problems of accurate visibility. Proceedings of Eurographics Workshop on Rendering, June 1996.

[23] Durand F, Drettakis G, Puech C. The visibility skeleton: a powerful and efficient multi-purpose global visibility tool. Proceedings of SIGGRAPH 97. p. 89–100.

[24] Mulmuley K. Computational geometry: an introduction through randomized algorithms. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[25] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. Computational geometry: algorithms and applications. Berlin: Springer, 1997.

[26] Levoy M, Hanrahan P. Light field rendering. Proceedings of SIGGRAPH 96. p. 31–42.

[27] Gortler S, Grzeszczuk R, Szeliski R, Cohen M. The lumigraph. Proceedings of SIGGRAPH 96. p. 43–54.

[28] Orti R, Rivière S, Durand F, Puech C. Radiosity for dynamic scenes in flatland with the visibility complex. Computer Graphics Forum, 1996;15(3):237–48. Proceedings of Eurographics '96.