A Visibility Algorithm for Hybrid Geometry- and Image-Based Modeling and Rendering

Thomas A. Funkhouser
Princeton University

Abstract

Hybrid geometry- and image-based modeling and rendering systems use photographs taken of a real-world environment and mapped onto the surfaces of a 3D model to achieve photorealism and visual complexity in synthetic images rendered from arbitrary viewpoints. A primary challenge in these systems is to develop algorithms that map the pixels of each photograph efficiently onto the appropriate surfaces of a 3D model, a classical visible surface determination problem. This paper describes an object-space algorithm for computing a visibility map for a set of polygons for a given camera viewpoint. The algorithm traces pyramidal beams from each camera viewpoint through a spatial data structure representing a polyhedral convex decomposition of space containing cell, face, edge, and vertex adjacencies. Beam intersections are computed only for the polygonal faces on the boundary of each traversed cell, and thus the algorithm is output-sensitive. The algorithm also supports efficient determination of silhouette edges, which allows an image-based modeling and rendering system to avoid mapping pixels along edges whose colors are the result of averaging over several disjoint surfaces. Results reported for several 3D models indicate the method is well-suited for large, densely-occluded virtual environments, such as building interiors.

1 Introduction

Hybrid geometry- and image-based rendering (GIBR) methods are useful for synthesizing photorealistic images of real-life environments (e.g., buildings and cities). Rather than modeling geometric details and simulating global illumination effects, as in traditional computer graphics, only a coarsely detailed 3D model is constructed, and photographs are taken from a discrete set of viewpoints within the environment. The calibrated photographic images are mapped onto the surfaces of the 3D model to construct a representation of the visual appearance of geometric details and complex illumination effects on each surface. Then, novel images can be rendered for arbitrary viewpoints during an interactive visualization session by reconstruction from these hybrid geometry- and image-based representations. This method allows photorealistic images to be generated of visually rich and large environments over a wide range of viewpoints, while avoiding the difficulties of modeling detailed geometry and simulating complex illumination effects.

Applications for GIBR systems include education, commerce, training, telepresence, and entertainment. For example, grammar school students can use such a system to “visit” historical buildings, temples, and museums. Real estate agents can show a potential buyer the interior of a home for sale interactively via the internet [20]. Distributed interactive simulation systems can train teams of soldiers, fire fighters, and other people whose missions are too dangerous or too expensive to re-create in the real world [3, 26]. Entertainment applications can synthesize photo-realistic imagery of real world environments to generate immersive walkthrough experiences for virtual travel and multi-player 3D games [21].

The research challenges in implementing an effective GIBR system are to construct, store, and re-sample a 4D representation for the radiance emanating from each surface of the 3D model. Previous related methods date back to the movie-map system by Lippman [25]. Greene [18] proposed a system based on environment maps [2] in which images captured from a discrete set of viewpoints are projected onto the faces of a cube. Chen and Williams [6] described a method in which reference images are used with pixel correspondence information to interpolate views of a scene. Debevec et al. [9, 10] developed a system in which photographs were used to construct a 3D model made from parameterized building blocks and to construct a “view-dependent texture map” for each surface. Gortler et al. [17] used approximate 3D geometry to facilitate image reconstruction from their 4D Lumigraph representation. Coorg [7] constructed vertical facades from multiple photographs and mapped diffuse imagery onto the surfaces for interactive visualization. Several walkthrough applications and video games apply 2D view-independent photographic textures to coarse 3D geometry [22]. Other related image-based representations are surveyed in [8], including cylindrical panorama [5, 28], Light Fields [24], and layered depth images [33].

An important step in constructing a GIBR representation is to map pixel samples from every photograph onto the surfaces of a 3D model. The challenge is to develop algorithms that determine which parts of which 3D surfaces are visible from the camera viewpoint of every photograph. This is a classic hidden surface removal (HSR) problem, but with a unique combination of requirements motivated by GIBR systems. First, unlike algorithms commonly used in computer graphics, the HSR algorithm must resolve visible surfaces with *object-space precision*. Image-space algorithms may cause small surfaces to be missed or radiance samples to be misaligned on a surface, causing artifacts and blurring in reconstructed images.

Second, the algorithm should compute a complete *visibility map* for each photograph, encoding not only the visible surfaces but also the visible edges and vertices with their connectivities on the view plane. From the visibility map, a GIBR system can detect pixels covering multiple disjoint surfaces (e.g., along silhouette edges) and avoid mapping them onto any single surface, which causes noticeable artifacts in resampled images. As an example, consider the situation shown in Figure 1. The image on the left shows a "photograph" taken with a synthetic camera of a simple 3D model comprising two rooms connected by a door. Using a standard HSR algorithm (e.g., z-buffering), pixels along the silhouette edge on the left side of the doorway might be mapped onto the wall, floor, and ceiling of the smaller room, even though their colors partially represent contributions from the edge of the door frame. The result is a "line" of incorrectly colored pixels in resampled images (shown in the image on the right). The HSR algorithm must detect silhouette edges so that these artifacts can be avoided.

Third, the HSR algorithm must scale to support very large 3D models. Typical models of interesting real-world environments contain many polygons, most of which are occluded for any given camera viewpoint. For example, consider the building shown in Figure 2. The left image shows a building (Soda Hall) from the exterior, while the right one shows the floorplan for one of seven floors. The entire 3D model contains around 10,000 polygons. Yet, for most camera viewpoints, more than 95% of them are hidden from view. To be effective for such large and densely occluded 3D models, a hidden surface removal algorithm must be *output sensitive*. That is, the expected case running time should depend only on the complexity of the visible portion of the model, not on the size of the entire model.

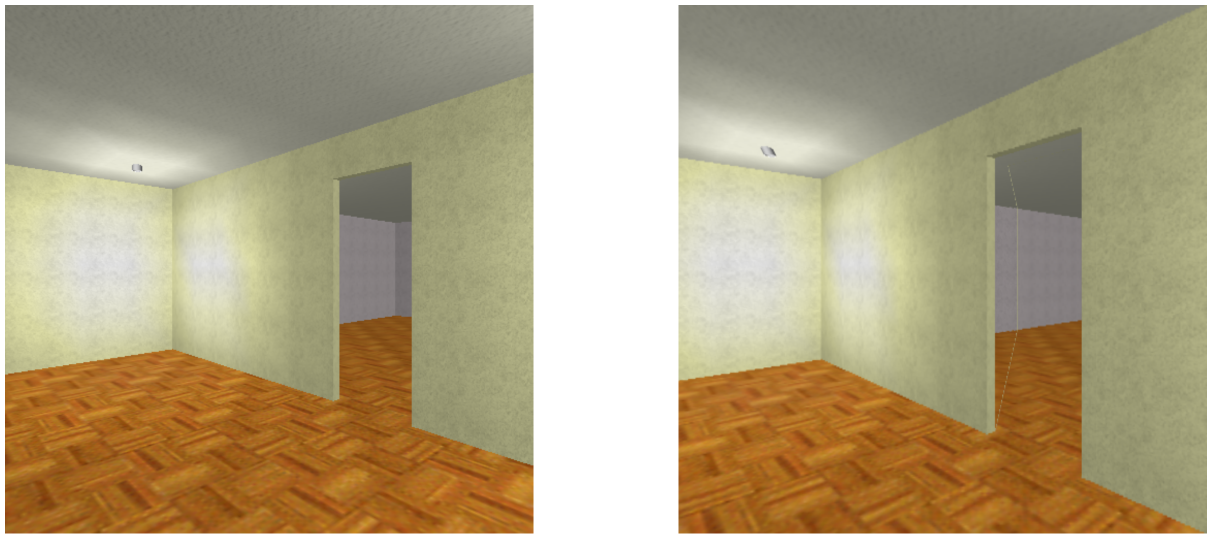


Figure 1: Image (on left) mapped onto surfaces of simple 3D model without detection of silhouette edges leads to artifacts appearing as a "line" of partially yellow pixels on wall, floor, and ceiling in a reconstructed image (on right).

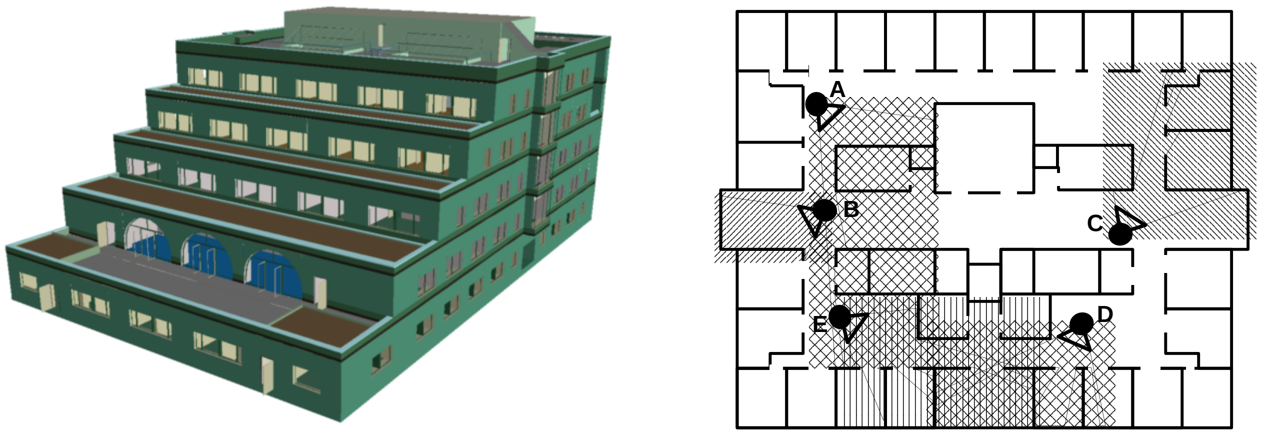


Figure 2: A building (left) and a floorplan (right). The visibility for several camera viewpoints are shown in cross-hatch patterns in the floorplan.

Finally, the algorithm should be tuned to accelerate visible surface determination for multiple camera viewpoints in a single execution. Since GIBR systems use many photographs for constructing a GIBR representation, the cost of precomputing a spatial data structure to accelerate hidden surface removal can generally be amortized over many computations for different viewpoints.

Despite the long history of prior work in hidden surface and hidden line removal in computer graphics [34] and computational geometry [13], there are not currently algorithms that meet all the requirements of a GIBR system. Research in computer graphics has focused mostly on HSR algorithms for image synthesis at screen resolution. Example methods include priority ordering [15, 31], recursive subdivision [36, 38], and depth buffering [4]. Meanwhile, research in computational geometry has focused mostly on proving asymptotic complexities of object-space algorithms. Lower and upper complexity bounds for the hidden surface and hidden line removal problems have been proven to be quadratic in the number of polygon boundaries, and algorithms have been described with optimal performance [11, 27, 32]. Yet, there is a dearth of practical object-space algorithms with attractive expected-case performance.

Debevec et al. [10] recently described a visibility algorithm for a GIBR system using both image-space and object-space methods. First, for each camera viewpoint, polygon IDs are rendered with a z-buffer into an item buffer [37] forming an image-space representation of the visible surfaces. Then, for every front-facing polygon P in the 3D model, a uniform sampling of 3D points is projected onto the image plane and checked against the corresponding entries in the item buffer to form a list of polygons occluding P. For each such occluder, P is clipped in object-space against planes formed by the camera viewpoint and the occluder's boundary. The problems with this method are that it is not object-space precise, potentially missing occluders not found by a discrete set of samples; it is not output-sensitive, clipping every front-facing polygon against all its occluders; and, it does not detect silhouette edges, potentially leading to visibility artifacts in reconstructed images.

The contribution of this paper is an algorithm that computes a visibility map for an arbitrary camera viewpoint. The basic idea is to trace pyramidal beams [19] recursively from a camera viewpoint through a precomputed spatial subdivision of cells (convex polyhedra) connected by "portals" (transparent boundaries between cells). Jones [23] has used this method to solve a hidden line removal problem for image generation, Teller [35] has used it for occlusion culling in an interactive walkthrough system, and Funkhouser et al. [16] have used it for acoustic modeling. A similar method has recently been developed by Fortune [14] for simulation of radio frequency wave propagation. We extend the recursive beam tracing method to compute a complete visibility map for a set of camera viewpoints and apply it to mapping photographs onto surfaces in a GIBR system. The algorithm executes at object-space precision, it is able to find silhouette edges efficiently, and its execution time is output-sensitive for each camera viewpoint after an initial precomputation.

2 System Organization

The organization of our complete GIBR system is shown in Figure 3. The input to the system is: 1) a 3D polygonal model of the environment, and 2) a set of photographic images calibrated with 3D camera viewpoints. The output is a sequence of synthetic images generated in real-time as a simulated observer moves through the environment.

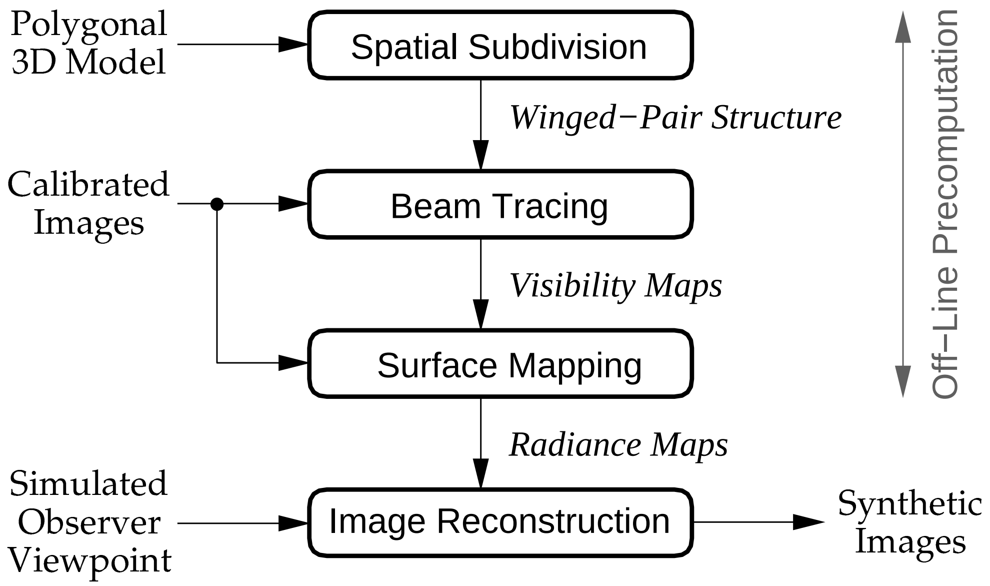


Figure 3: Organization of GIBR system.

The system is divided into four phases, three of which perform off-line computations to preprocess the input data. During the first preprocessing phase, a spatial subdivision data structure is constructed to represent the topology and geometry of the 3D model. Next, beams are traced through the spatial subdivision to compute a visibility map for each camera viewpoint. Then, during the third and final preprocessing phase, the visibility maps are used to map regions of the calibrated images onto 3D polygons to create a set of radiance maps representing the 4D radiance emanating from each surface of the 3D model. Finally, during the fourth phase, synthetic images are generated from the radiance maps at interactive rates for an arbitrary viewpoint moving through the 3D model under user control.

In this paper, we focus on the first two phases of the system: spatial subdivision and beam tracing. The goal of these two phases is to compute a visibility map for each camera viewpoint so that photographic radiance samples can be mapped onto the surfaces of the 3D model. Detailed descriptions of the last two phases are purposely omitted in this discussion. Although we currently use a view-dependent texture map to store the radiance map for each surface (as in [10]), the reader can imagine use of any GIBR representation of the 4D radiance emanating from each surface of a 3D model that can be constructed from a set of photographs augmented with corresponding visibility maps, e.g., Light Fields [24] or Lumigraphs [17].

The remainder of this paper provides detailed descriptions of our spatial subdivision and beam tracing data structures and algorithms, followed by results of experiments with several 3D virtual environments and a brief conclusion.

3 Spatial Subdivision

During the first preprocessing phase, our system builds a spatial subdivision representing a decomposition of 3D space, which we call a *winged-pair* data structure. The goal of this phase is to partition space into convex polyhedral cells whose boundaries are aligned with polygons of the 3D input model and encode the topological adjacencies of the cells in a data structure enabling output-sensitive traversals of sightlines through 3D space.

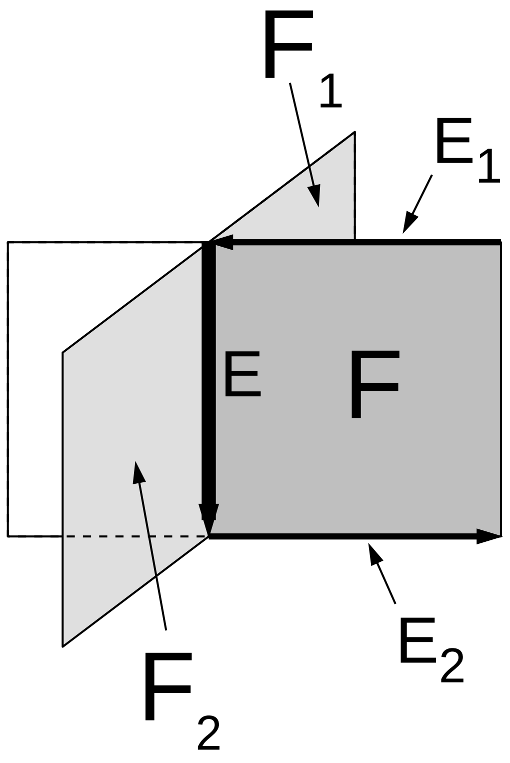
The winged-pair data structure is motivated by the well-known winged-edge data structure described by Baumgart [1]. The difference is that the winged-pair describes topological structures one dimension higher than the winged-edge. While the winged-edge represents a 2-manifold, the winged-pair represents a 3-manifold. The reader can think of the winged-pair data structure as a set of “glued together” winged-edge data structures, each representing the 2-manifold boundary of a single cell.

The winged-pair representation is also similar to the facet-edge representation described by Dobkin and Laszlo [12]. Both encode face-edge, face-face, and edge-edge adjacency relationships in a set of structures corresponding to face-edge pairs. The difference is that Dobkin and Laszlo store a separate structure for each unique pair of face and edge orientations. For most practical purposes, the difference is insignificant, similar to the one between 2D quad-edge and winged-edge structures. Although storing separate structures for different orientations can eliminate a few conditional checks during traversal operations, it requires extra storage – a straight-forward trade-off between time and space. We choose to store exactly one structure for each face-edge pair in our winged-pair representation to simplify code extensions and debugging.

Pseudocode declarations for the winged-pair structure are shown in Figure 4. Topological adjacencies are encoded in fixed-size records associated with vertices, edges, faces, cells, and face-edge pairs. Every vertex stores its 3D location and a reference to one attached edge, every edge stores references to its two vertices and one attached face-edge pair, each face stores references to its two cells and one attached face-edge pair, and every cell stores a reference to one attached face. The face-edge pairs store references to one edge E and to one face F along with adjacency relationships required for topological traversals. Specifically, they store references (*spin*) to the two face-edge pairs reached by spinning F around E clockwise and counter-clockwise (see Figure 5) and to the two face-edge pairs (*clock*) reached by moving around F in clockwise and counter-clockwise directions from E (see Figure 5). The face-edge pair also stores a bit (*direction*) indicating whether the orientation of the vertices on the edge is clockwise or counter-clockwise with respect to the face within the pair.

```
class Vertex {
   Point position;
   Edge *edge;
};

class Edge {
   Vertex *vertex[2];
   Pair *pair;
};

class Face {
   Cell *cell[2];
   Pair *pair;
};

class Cell {
   Face *face;
};

class Pair {
   Face *face;
   Edge *edge;
   Pair *clock[2];
   Pair *spin[2];
   Bit direction;
};
```

Figure 4: Winged-pair declarations.



Clock(Pair(F,E),CW) = F,E_1
Clock(Pair(F,E),CCW) = F,E_2
Spin(Pair(F,E),CW) = F_1,E
Spin(Pair(F,E),CCW) = F_2,E

Figure 5: Pair-pair references.

These simple, fixed-size structures make it possible to execute output-sensitive topological traversals through cell, face, edge, and vertex adjacency relationships. For instance, finding all faces on the boundary of a given cell requires $O(C_f + C_e)$ time, where C_f and C_e are the numbers of faces and edges attached to the cell, respectively. As in winged-edge traversals in 2D, simple conditional statements are often required to check the orientation of each structure before moving to the next. For instance, to find the cell adjacent to another across a given face, the C++ code looks like this:

```
Cell *CellAcrossFace(Face *face, Cell *cell) {
   return (cell == face→cell[0]) ? face→cell[1] : face→cell[0];
}
```

We build the winged-pair data structure for any 3D model using a Binary Space Partition (BSP) [15], a recursive binary split of 3D space into convex polyhedral regions (*cells*) separated by planes. To construct the BSP, we recursively split cells by candidate planes selected by the method described in [29]. As BSP cells are split by a polygon P, the corresponding winged-pair cells are split along the plane supporting P, and the faces and edges on the boundary of each split cell are updated to maintain a 3-manifold in which every face is convex and entirely inside or outside every input polygon. As faces are created, they are labeled according to whether they are opaque (coincide with an input polygon) or transparent (split free space). The binary splitting process continues until no input polygon intersects the interior of any BSP cell, leading to a set of convex polyhedral cells whose faces are all convex and cumulatively contain all the input polygons. The resulting winged-pair is converted to an ASCII representation and written to a file for use by later phases of the GIBR system.

4 Beam Tracing

In the second phase of our GIBR system, we use a beam tracing algorithm to compute a visibility map for every photograph. Beams containing feasible sightlines from each camera viewpoint are traced via a depth-first traveral through the winged-pair structure, while corresponding convex regions of the visibility map are partitioned recursively. The algorithm is based on recursive beam tracing methods [23, 16, 35], and it is related to recursive convex decompositions [30]. The key feature is that topological information stored in the winged-pair data structure (edge-face adjacencies) is used to construct a visibility map with topological information and explicit silhouette edges.

Psuedocode for the beam tracing algorithm is shown in Figure 6. During the traversal for each camera viewpoint, the algorithm maintains: a *visibility map* M (a 2-manifold representing the geometry and topology of faces and edges visible to the camera), a *current region* R (a convex 2D region of the visibility map), a *winged-pair* W (as described in Section 3), a *current cell* C (a reference to a cell in the winged-pair structure) and a *current beam* B (an infinite convex 3D pyramidal beam emanating from the camera viewpoint containing all sightlines passing through R). Initially, M are R are initialized to one rectangular region enclosing the visible portion of the view plane, W is constructed from the 3D model as described in Section 3, C is set to be the cell of W containing the camera, and B is set to the four-sided pyramid corresponding to the view frustum of the camera.

During each recursive step, the function called *TraceBeams* partitions the current region of the visibility map into multiple convex subregions corresponding to intersections of the current beam with faces on the boundary of the current cell. For each face F_i on the current cell and intersecting the current beam, a convex region R_i is inserted into the visibility map. If F_i is transparent, R_i is recursively refined with a call to *TraceBeams* in which the current region of the visibility map is set to R_i, the new current cell C_i is set to be the cell adjacent to C across face F_i, and the new current beam B_i is formed by trimming B to include only rays intersecting F_i. Otherwise, F_i is opaque, the recursive search along this path terminates, and R_i is marked as a final region of the visibility map associated with face F_i. Contiguous regions of the visibility map associated with the same opaque winged-pair face are marked as one face during the process.

A nice feature of this algorithm is that it constructs a representation of the visibility map with

```
void TraceBeams()
begin
    // Initialization
    $V$ = Viewpoint of camera;
    $M$ = Visibility map;
    $R$ = Region of $M$ initially enclosing viewable area;
    $W$ = Winged-pair structure;
    $C$ = Cell of $W$ containing camera;
    $B$ = Beam enclosing camera view frustum;
    TraceBeams($V, M, R, W, C, B$);
end

void TraceBeams($V, M, R, W, C, B$)
begin
    // Consider each face on cell boundary
    foreach face $F_i$ on boundary of $C$ do
        // Check if beam intersects face
        if (Intersects($B$, $F_i$) then
            // Create new region in visibility map
            $R_i$ = InsertRegion($M$, $F_i$);

            // Recurse along path through transparent face
            if (Transparent($F_i$)) then
                $C_i$ = NeighborCell($W, C, F_i$);
                $B_i$ = NewBeam($B$, $F_i$);
                TraceBeams($V, M, R_i, W, C_i, B_i$);
            endif
        endif
    endfor
end
```

Figure 6: Pseudocode for the beam tracing algorithm.

both topological and geometric information. With the exception of silhouette edges, the topology of the visibility graph matches the topology of corresponding vertices, edges, and faces in the winged-pair structure exactly. Silhouette edges can be found explicitly by checking the orientations of the faces attached to visible edges, which are readily available by traversing *spin* references stored in the winged-pair data structure.

5 Experimental Results

We have implemented the algorithms described in the previous sections, and they run on SGI/Irix and PC/Windows computers. To test the effectiveness of our methods, we executed a series of tests with several 3D models (shown in Figure 7). For each one, we computed a winged-pair data structure and a visibility map for at least 100 camera viewpoints along a typical walkthrough path within the model. All tests were run on a SGI Onyx2 Workstation with a 195MHz R10000 processor.

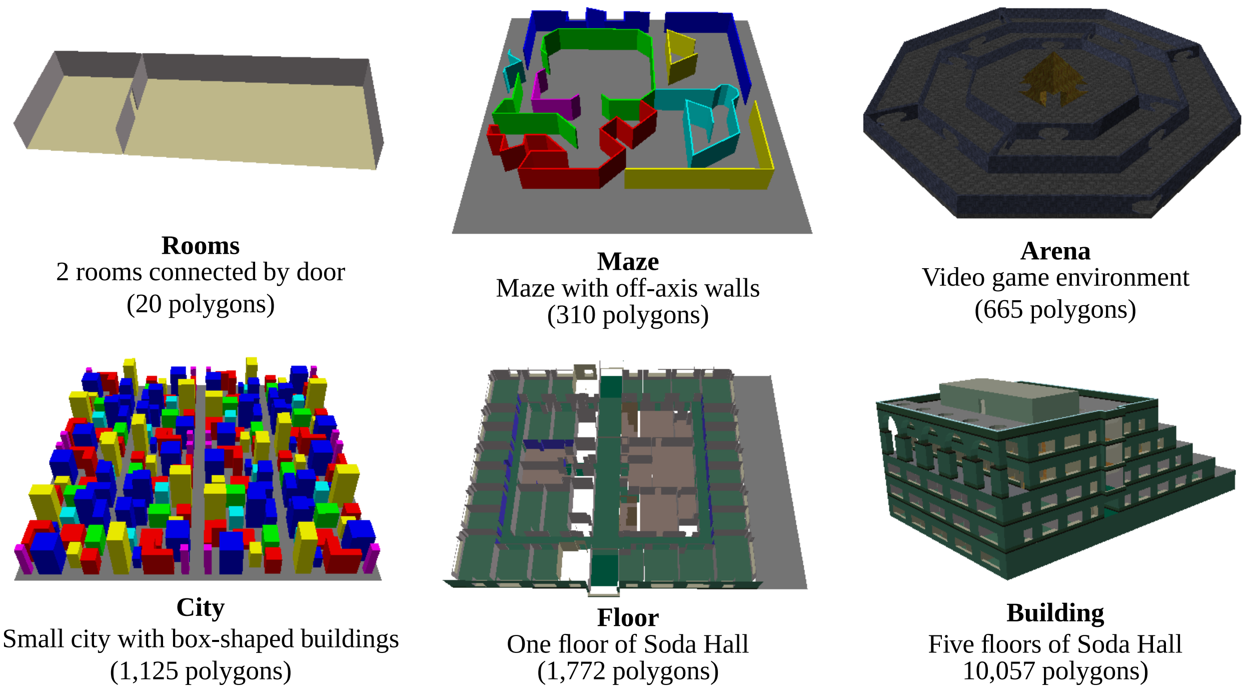


Figure 7: Test models.

Results of the spatial subdivision phase are shown in Table 1. For each test model, we list how many input polygons it has (# Polys), along with the numbers of cells, faces, edges, and vertices in the resulting winged-pair structure and the wall-clock time (in seconds) taken by the spatial subdivision algorithm. We note that the complexity of the winged-pair grows linearly with the numbers of polygons for these models. Also, the spatial subdivision processing times are reasonably quick (less than one minute), even for complex 3D models such as a small city or a large building.

Test Model	# Polys	# Cells	# Faces	# Edges	# Verts	Time (s)
Rooms	20	6	38	66	35	0.26
Maze	310	315	1,485	2,113	944	0.84
Arena	665	294	2,181	3,625	1,739	3.96
City	1,125	829	4,746	7,568	3,652	3.25
Floor	1,772	808	5,773	9,623	4,659	11.36
Bldg	10,057	4,504	33,066	54,459	25,898	50.76

Table 1: Spatial subdivision statistics.

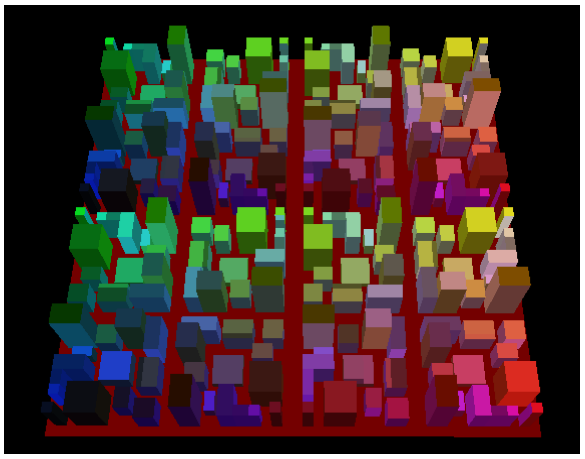
Table 2 shows results of the beam tracing phase. The first two columns list the test model and how many input polygons it has. The remaining columns list the minimum, average, and maximum statistics measured over all tested camera viewpoints in each model. Specifically, groups of three columns list the numbers of beams traced by our algorithm, the numbers of faces and edges in the resulting visibility maps, and the wall-clock times (in milliseconds) required by our algorithm for each viewpoint. From these results, we observe that the time required by our algorithm correlates closely with the number of beams traced, and not necessarily with the number of polygons in the 3D model. For example, although the ‘Floor’ model contains polygons for only one of five floors in the ‘Building’ model, the statistics gathered during our tests with both models are similar because the same viewpoint path was used and the complexity of the computed visibility maps were similar. Similarly, although the ‘Building’ model contains around nine times more polygons than the ‘City’ model (10,057 versus 1,125), the numbers of beams traced, the computation times, and the complexities of the visibility maps measured in tests are far less for the ‘Building’ due to the dense occlusions of walls. Finally, we note that the maximum time required to compute the visibility map in all our tests was a little more than one second.

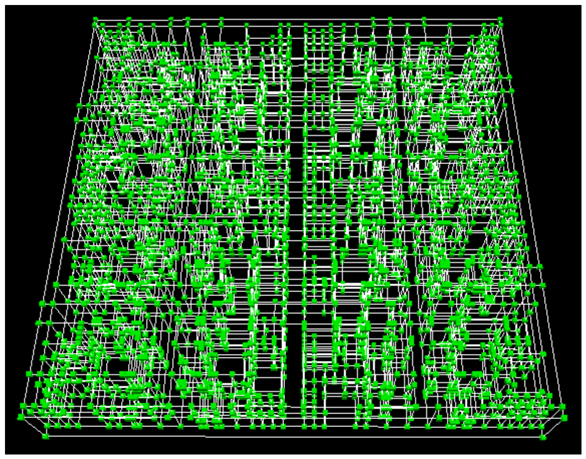
Test	#	# Beams			# Faces			# Edges			Time (ms)		
Model	Polys	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
Rooms	20	1	2	3	3	10	18	6	20	36	1	3	6
Maze	310	2	37	100	4	32	82	8	64	163	2	25	60
Arena	665	10	57	136	33	81	234	66	160	456	23	108	216
City	1,125	15	642	1,680	4	166	382	8	324	748	15	471	1,092
Floor	1,772	6	23	54	13	67	183	26	132	361	9	31	79
Bldg	10,057	6	25	67	13	73	183	26	144	361	10	36	97

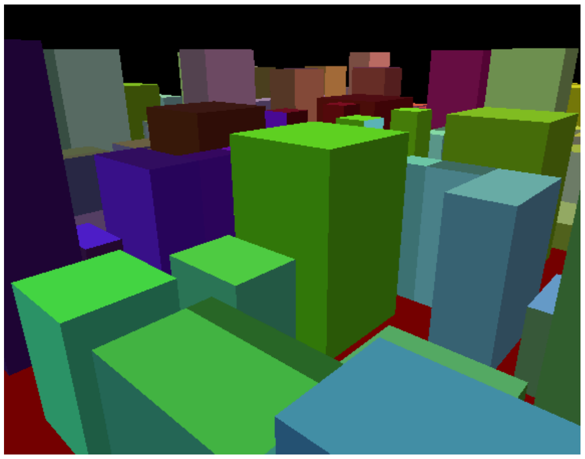
Table 2: Beam tracing statistics.

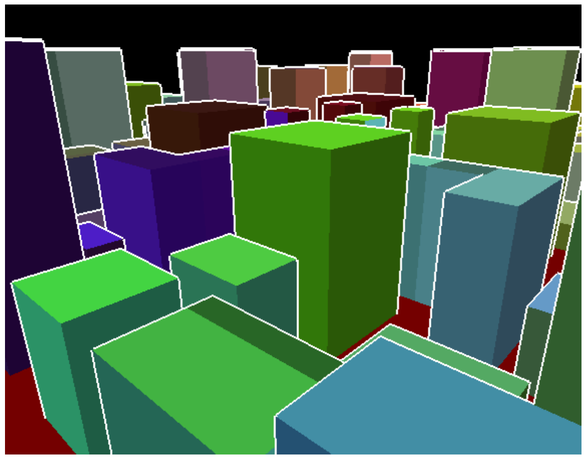
Figure 8 shows visualizations of our algorithms captured from an interactive program computing the visibility map for viewpoints in the ‘City’ model. The top two images ((a) and (b)) show the winged-pair structure constructed by our system. In these images, every face is drawn with a unique color, edges are drawn as solid white lines, and vertices are drawn as green dots. Note how few input polygons are split by binary space partitioning planes. The next three images ((c), (d), and (e)) show views from one camera flying over the city. As before, every face is drawn with a unique color, but computed silhouette edges are also shown as wide white lines in image (d), and intersections between beams and winged-pair faces are shown as yellow lines in image (e). The bottom-right image (f) shows a birds-eye view of the set of surfaces (blue polygons) visible to the viewpoint (looking from the bottom-left corner of the image towards the top-right corner) overlaid with edges of the winged-pair structure (white lines).

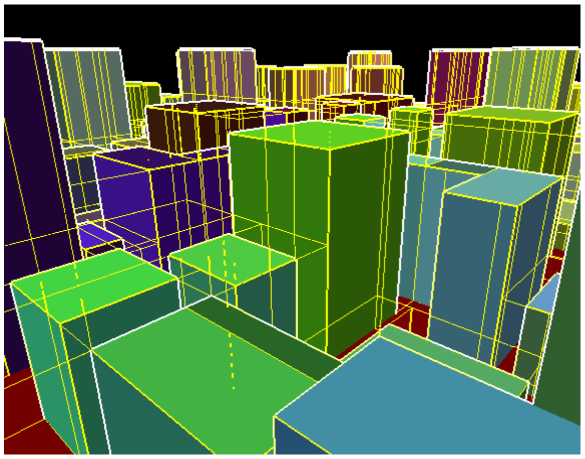
The images in the bottom row of Figure 8 illustrate the most significant problem with the recursive beam tracing approach: beams get fragmented by cell boundaries as they are traced through free space [14, 35]. In theory, the number of beams can be exponential in the number of winged-pair faces traversed. In practice, the number of beams traced depends on the complexity of the visible region of the model. As a result, these methods are best-suited for use in densely occluded environments, such as building interiors. In future work, we plan to pursue topological beam tracing methods in which beams are split only at silhouette edges (as in [14]).

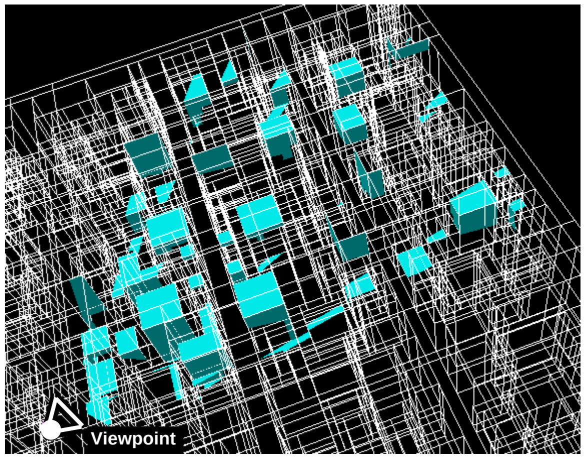

(a) Winged-pair faces (birds-eye view).
(every face is drawn with a unique color)


(b) Winged-pair edges and vertices (birds-eye view).
(edges are white, and vertices are green)


(c) Winged-pair faces (interior view).
(every face is drawn with a unique color)


(d) Silhouette edges (interior view).
(silhouette edges are thick white lines)


(e) Visibility map computation (interior view).
(yellow lines show beam-face intersections)



(f) Visible surfaces (birds-eye view).
(winged-pair edges are white, visible surfaces are blue)

Figure 8: Visualization of winged-pair structure and visibility map computed for ‘City’ model.

6 Conclusion

This paper presents data structures and algorithms useful for mapping images onto surfaces in a hybrid geometry- and image-based rendering system. Our method uses a preprocessing phase to construct a winged-pair data structure encoding the topology and geometry of the 3D input model. A second phase traces beams through the winged-pair structure to find visible surfaces. The beam tracing algorithm computes visible surfaces with object-space precision, it is able to find silhouette edges efficiently, and its execution time depends only on the complexity of the visible region for each camera viewpoint. Topics for future work include investigation of how topological relationships can be used to construct, store, and sample radiance maps more efficiently.

Acknowledgements

The author would like to acknowledge Steve Fortune and David Dobkin for their helpful discussions. Thanks also go to Seth Teller for use of the ‘Maze’ test model, and Bruce Naylor for the ‘Arena’ test model. Finally, I would like to thank Wilmot Li and Wagner Correa for their contributions to the beam tracing implementation.

References

[1] Bruce G. Baumgart. Geometric modeling for computer vision. AIM-249, STA -CS-74-463, CS Dept, Stanford U., October 1974.

[2] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19:542–546, 1976.

[3] James Calvin, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller, and Dan Owen. The simnet virtual world architecture. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 450–455, September 1993.

[4] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974.

[5] Shenchang Eric Chen. Quicktime VR - an image-based approach to virtual environment navigation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 29–38. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[6] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH ’93 Proceedings)*, volume 27, pages 279–288, August 1993.

[7] Satyan Coorg. *Pose Imagery and Automated 3-D Modeling of Urban Environments*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 1998.

[8] Paul Debevec and Steven Gortler. Image-based modeling and rendering. In *SIGGRAPH 98 Course Notes*. ACM SIGGRAPH, Addison Wesley, July 1998.

[9] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[10] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Rendering Workshop*, pages 105–116, June 1998.

[11] F. Dévai. Quadratic bounds for hidden line elimination. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 269–275, 1986.

[12] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[13] S. E. Dorward. A survey of object-space hidden surface removal. *Internat. J. Comput. Geom. Appl.*, 4:325–362, 1994.

[14] Steven Fortune. Topological beam tracing. In *to appear in Proc. ACM Symposium on Computational Geometry*, 1999.

[15] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. volume 14, pages 124–133, July 1980.

[16] Thomas A. Funkhouser, Ingrid Carlbom, Gary Elko, Gopal Pingali, Mohan Sondhi, and Jim West. A beam tracing approach to acoustic modeling for interactive virtual environments. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 21–32. ACM SIGGRAPH, Addison Wesley, July 1998.

[17] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[18] Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11), November 1986.

[19] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH ’84 Proceedings)*, volume 18, pages 119–127, July 1984.

[20] Modern Homes. www.modernhomes.com/demo.html, 1999.

[21] id Software. Quake, 1996.

[22] William Jepson, Robin Liggett, and Scott Friedman. An environment for real-time urban simulation. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 165–166. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[23] C. B. Jones. A new approach to the 'hidden line' problem. *Computer Journal*, 14(3):232–237, August 1971.

[24] Marc Levoy and Pat Hanrahan. Light field rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[25] A. Lippman. Movie-maps: an application of the optical videodisc to computer graphics. *Computer Graphics*, 14(3):32–42, July 1980.

[26] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Donald P. Brutzman, and Paul T. Barham. Exploiting reality with multicast groups. *IEEE Computer Graphics and Applications*, 15(5):38–45, September 1995.

[27] M. McKenna. Worst-case optimal hidden-surface removal. *ACM Trans. Graph.*, 6:19–28, 1987.

[28] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[29] Bruce Naylor. Constructing good partition trees. In *Proceedings of Graphics Interface '93*, pages 181–191, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[30] Bruce F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, May 1992.

[31] Martin E. Newell, R. G. Newell, and T. L. Sancha. A new approach to the shaded picture problem. In *Proc. ACM Nat. Conf.*, page 443. 1972.

[32] A. Schmitt. On the time and space complexity of certain exact hidden line algorithms. Report 24/81, Fakultät Inform., Univ. Karlsruhe, Karlsruhe, West Germany, 1981.

[33] Jonathan W. Shade, Steven J. Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 231–242. ACM SIGGRAPH, Addison Wesley, July 1998.

[34] I.E. Sutherland, R.F. Sproull, and R.A. Shumacker. A characterization of ten hidden surface algorithms. *ACM Computing Surveys*, 6(1):1–55, March 1974.

[35] Seth Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, (Also TR UCB/CSD 92/708) CS Dept., UC Berkeley, 1992.

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

[37] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Trans. Graph.*, 3(1):52–69, 1984.

[38] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics (SIGGRAPH '77 Proceedings)*, 11(2):214–222, July 1977.