Portals and Mirrors:

Simple, Fast Evaluation of Potentially Visible Sets

David Luebke and Chris Georges
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

We describe an approach for determining potentially visible sets in dynamic architectural models. Our scheme divides the models into cells and portals, computing a conservative estimate of which cells are visible at render time. The technique is simple to implement and can be easily integrated into existing systems, providing increased interactive performance on large architectural models.

Introduction

Architectural models typically exhibit high depth complexity paired with heavy occlusion. The ratio of objects actually visible to the viewer (not occluded by walls) to objects theoretically visible to the viewer (intersecting the view frustum) will usually be small in a walkthrough situation. A visibility algorithm aimed at reducing the number of primitives rendered can exploit this property. Following prior work [1,2,3], we make use of a subdivision that divides such models along the occluding primitives into "cells" and "portals". A cell is a polyhedral volume of space; a portal is a transparent 2D region upon a cell boundary that connects adjacent cells. Cells can only "see" other cells through the portals. In an architectural model, the cell boundaries should follow the walls and partitions, so that cells roughly correspond to the rooms of the building. The portals likewise correspond to the doors and windows through which neighboring rooms can view each other.

Given such a spatial partitioning of the model, we can determine each frame what cells may be visible to the viewer. By traversing only the cells in this potentially visible set (PVS), we can avoid submitting occluded portions of the model to the graphics pipeline. What cells comprise the PVS? Certainly the cell containing the viewpoint is potentially visible. Those neighboring cells which share a portal with the initial cell must also be counted as potentially visible, since the viewer could see those cells through the portal. To this we add those cells visible through the portals of these neighbors, and so on. In this manner the problem of determining what cells are potentially visible to the viewer reduces to the problem of determining what portals are visible through the portals of the viewer's cell.

luebke@cs.unc.edu (919) 962-1825
georges@cs.unc.edu (919) 962-1789
CB# 3175 Sitterson Hall; UNC, Chapel Hill, NC 27599-3175

Our system makes this determination dynamically at render time. Rather than finding the exact PVS for each cell as a preprocess, we postpone the visibility computation as long as possible. This type of dynamic evaluation of portal-portal visibility is not new. Earlier efforts have centered on precisely determining sightlines through portals; our method offers a less exact but much simpler alternative. The algorithm has been implemented on the Pixel-Planes 5 graphics computer at the University of North Carolina and provides a substantial speedup on a sample model of 50,000 polygons.

Previous Work

Jones [1] explored the subdivision of geometry into cells and portals as a technique for hidden line removal. In his algorithm, models are manually subdivided into convex polyhedral cells and convex polygonal portals. The subdivision is complete in the sense that every polygon in the dataset is embedded in the face of one or more cells. Rendering begins by drawing the walls and portals of the cell containing the viewer. As each portal is drawn, the cell on the opposite side of the portal is recursively rendered. In this way the cell adjacency graph defined by the partitioning is traversed in depth-first fashion. The portal sequence through which the current cell is being rendered comprises a convex "mask" to which the contents of the cell are clipped. If the intersection of a portal with the current mask is empty, the portal is invisible and the attached cell need not be traversed.

More recent work has abandoned the attempt to compute exact visibility information, focusing instead on computing a conservative PVS of objects that *may* be visible from the viewer's cell. The graphics pipeline then uses standard Z-buffer techniques to resolve exact visibility. Airey [2] was the first to use a portal-based approach effective in architectural environments. He described multiple ways to approach the problem of determining cell-to-cell visibility, including ray-casting and shadow volumes. Teller [3] has taken the concept further and found a closed-form, analytic solution to the portal-portal visibility problem. Using 2D linear programming to test portal sequences against arbitrary visibility beams, Teller computes a complete set of cell-to-cell and cell-to-object visibilities in a preprocess. At render time this PVS is further restricted according to which portals are actually visible. Teller's approach is mathematically and computationally complex, requiring hours of preprocess time for large models [3].

Motivation

Such a large preprocessing cost may be inappropriate for interactive applications. For example, architectural walkthroughs are often used for revision purposes. A visualization of a building under design is more valuable to an architect if inquiries of the type "What if I move this wall out ten feet?" can be answered immediately. Adding portals, moving portals, and redistributing

cells boundaries will all be common operations in an interactive architectural design application. To take full advantage of the static visibility schemes mentioned above, each of these would require a potentially lengthy PVS recalculation best done off-line.

Envisioning such an application as our final goal, we decided to focus on improving the dynamic visibility determination. Jones' algorithm finds the exact intersection of 2D convex regions, requiring O(n lg n) time for portal sequences with n edges. Teller's linear programming approach computes only the *existence* of an intersection, and runs in time linear in the number of edges. We sought a dynamic solution that would also run in linear time and would integrate easily into existing systems.

Faster Dynamic PVS Evaluation

We use a variation of Jones' approach that employs bounding boxes instead of general convex regions. Our scheme first projects the vertices of each portal into screen-space and takes the axial 2D bounding box of the resulting points. This 2D box, called the *cull box*, represents a conservative bound for the portal; that is, objects whose screenspace projection falls entirely outside the cull box are guaranteed not to be visible through the portal and may be safely culled away. As each successive portal is traversed, its box is intersected with the aggregate cull box using only a few comparisons.

During traversal the contents of each cell are tested for visibility through the current portal sequence by comparing the screenspace projection of each object's bounding box against the intersected cull box of all portals in the sequence. If the projected bounding box intersects the aggregate cull box, the object is potentially visible through the portals and must be rendered. Since a single object may be visible through multiple portal sequences, we tag each object as we render it. This *object-level culling* lets us avoid rendering objects more than once per frame.

Alternatively, we can render each object once for every portal sequence which admits a view of the object, but clip the actual primitives to the aggregate cull box of each sequence. Under this *primitive-level clipping* scheme objects may be visited more than once, but since the portal boundaries do not overlap, no portion of any primitive will be rendered twice. Typically object-level culling will prove more efficient, but for objects whose per-primitive rendering cost far exceeds their clipping cost, primitive-level clipping provides a viable option.

Implementation

We have implemented this approach on Pixel-Planes 5, the custom graphics multicomputer developed at the University of North Carolina. The traversal mechanism treats portals as primitives to be rendered. Each portal consists of a polygonal boundary and a pointer to the adjacent cell; when a portal is encountered during traversal we test its axial screenspace bounding box against the current aggregate cull box. If the intersection is non-empty, we use it as the new aggregate cull box and recursively traverse the connected cell.

We feel that modeler integration is crucial to this problem of interactive model revision. If an architect wishes to move a wall or broaden a doorway, the modeling system should be able to make the change quickly and broadcast that change to the graphics system. In our system the spatial partitioning of the model into cells and portals is directly embedded in the modeler's representation. Portals are treated as augmented polygons, each tagged with the name of the attached cell. Cells are simply logical groupings in the modeler's hierarchy and need not necessarily be convex. We have found this quite convenient when constructing models; each room typically corresponds to a cell and it takes only seconds to add and move a portal, or to reshape a cell. We have already adapted two commercial modelers to our system, which speaks to the simplicity of the integration process.

Results

We have tested our system on a subset of the UNC Walkthrough project's model of Professor Fred Brooks' house, comprised of 367,000 radiositized triangles. The speedup obtained by this visibility algorithm, like the speedup obtained by similar schemes, is extremely view-and model-dependent. Over a 500-frame test path through the model, the frame rate using PVS evaluation ranged from just over 1 to almost 10 times the frame rate of the entire unculled model. For typical views the dynamic PVS evaluation culled away 20% to 50% of the model. It should be emphasized again that these numbers are specific to the model and view path, but they certainly indicate the promise of the algorithm as a simple, effective acceleration technique.

Ongoing and Future Work

Efficiency could be further increased by applying *obscuration culling* to portals [4]. This scheme tests potentially visible items against an "almost complete" Z-buffer before rendering. This would allow the 'detail' objects in each cell as well as the occluding cell walls to block portals, potentially reducing the PVS. The Pixel-Planes architecture makes obscuration culling of portals feasible, and we are currently exploring this possibility.

Teller mentions that the concept of portals may be extended to mirrors [3]. Under this scheme mirrors are treated as portals which transform the attached cell about the plane of the mirror; this has the advantage of automatically restricting the PVS seen through the mirror. Though conceptually simple, mirrors introduce many practical difficulties which require additional clipping by the rendering engine to resolve. For example, geometry behind the mirror must not appear in its reflected "world," and reflected geometry must not appear in front or to the side of the mirror.

A special case that avoids these problems can be constructed by embedding the mirror in an opaque cell boundary (for example, a wall-mounted mirror in a bathroom), and we have implemented such mirrors (Plate 1). The concept of an immovable mirror fits poorly with our goal of interactive, dynamic environments, however, so we have focused on the more general case. Clipping is complicated further by mirrors that overlap in screenspace, and further still by mirrors which recursively reflect other mirrors. At present our system allows static mirrors, which can reflect each other to arbitrary levels of recursion, or more general "hand-held" mirrors, (an example of free-moving portals), which permit one-bounce reflections. We are currently working on the dynamic, fully recursive case.

Acknowledgments

The authors would like to extend their sincere thanks to Mike Goslin, Hans Weber, Power P. Ponamgi, Peggy Wetzel, and Stump Brady. This work was supported by ARPA Contract DABT63-93-C-C048.

References

[1] Jones, C.B. A New Approach to the 'Hidden Line' Problem. The Computer Journal, vol. 14 no. 3 (August 1971), 232..

[2] Airey, John. Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. Ph.D. thesis, UNC-CH CS Department TR #90-027 (July 1990).

[3] Teller, Seth. Visibility Computation in Densely Occluded Polyhedral Environments. Ph.D. thesis, UC Berkeley CS Department, TR #92/708 (1992).

[4] Greene, Ned, Kass, Michael, and Miller, Gavin. Hierarchical Z-Buffer Visibility. Proceedings of SIGGRAPH '93 (Anaheim, California 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York 1993, pp. 59-66.

Plate 1. View from the master bedroom of the Brooks House showing cull boxes for portals (white) and mirrors (red).

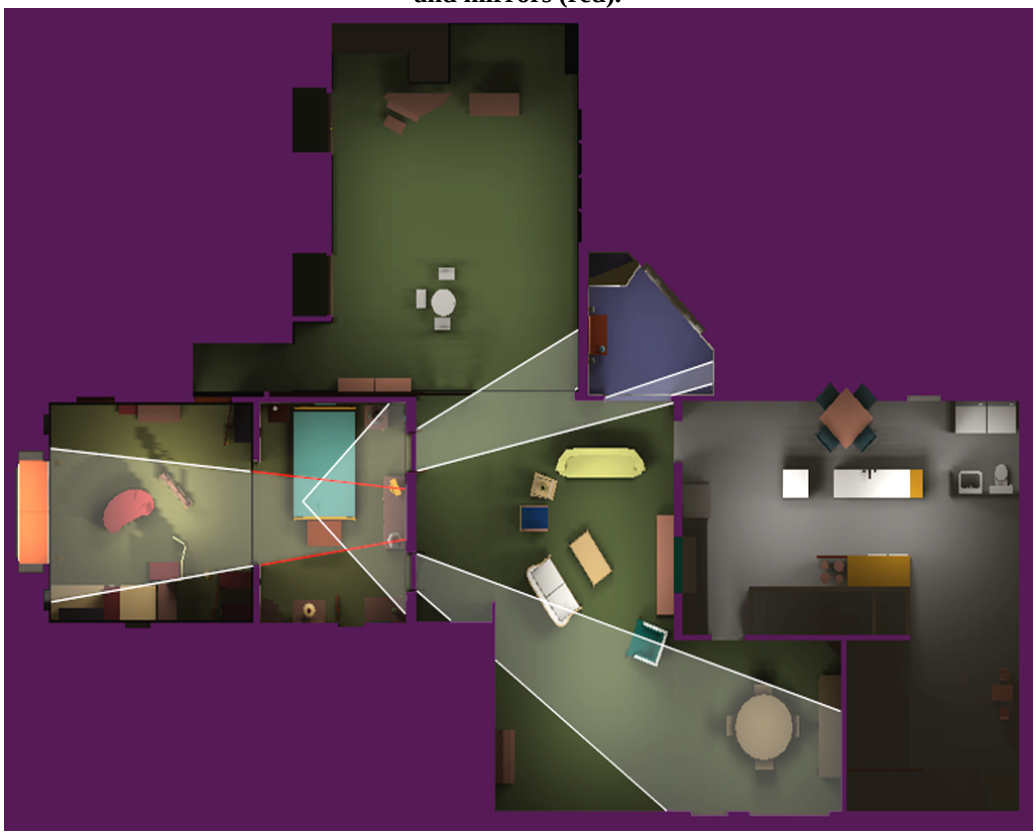

Plate 2. Overhead view of the Brooks House, showing portal culling frustums active in Plate 1 (mirror frustum shown in red).