Global Visibility

Marek Trefný *(trefny@sgi.felk.cvut.cz)*
JiříBittner *(xbittner@sgi.felk.cvut.cz)*

Czech Technical University, Faculty of Electrical Engineering
Department of Computer Science and Engineering
Karlovo náměstí13, 121 35 Praha 2, Czech Republic

Abstract

Virtual reality applications such as architectural walkthroughs require a powerful graphics hardware to achieve interactive frame rates of rendering. Usual rendering algorithms like z-buffer or BSP-tree often spend significant time processing portions of model that are currently invisible to an observer. To describe restrictions in visibility, the visibility information can be precomputed and used to dramatically reduce amount of data to be processed during rendering. The model of a virtual environment is divided into cells using an algorithm of spatial subdivision. The spatial subdivision is represented by a graph structure. For each cell the cell-to-cell and cell-to-frustum visibility information is determined and stored in a tree data structure within that cell. However, widely used languages for description of model geometry, such as VRML, do not offer resources to describe the scene subdivision and to store the explicit visibility information along with the geometry. This article discusses some solutions that were chosen during the global visibility research project at the Department of Computer Science and Engineering at FEE CTU in Prague.

Keywords: global visibility, VRML, conservative visibility, spatial subdivision, visibility computations.

1 Introduction

The main aim of visibility precomputation is to reduce the amount of data to be processed during rendering. A *model of virtual environment* (MVE) is subdivided into *cells*. The cell is a convex polyhedral volume of space. After the spatial subdivision is completed, a *portal* enumeration finds transparent polygonal regions upon a cell boundary that connects adjacent cells. Obviously, the observer can see from one cell to another through the portals on their shared boundary and to the more distant cells through *portal sequences*. In densely occluded environments, such as architectural models, the cell boundaries often follow walls and partitions, so the cells roughly correspond to rooms of virtual environment. The portals likewise correspond to doors and windows through which neighboring rooms are connected. The entities of spatial subdivision, these are cells, boundaries and portals, together with their topological connections are represented by the *adjacency graph*. For each cell in the adjacency graph we determine a set of potentially visible regions. A hierarchical data structure – the *visibility tree* is used to store the visibility information. The root node of the tree corresponds to the *source cell*, which is the cell the tree is built for. The tree stores *cell-to-cell* visibility relation, which also defines the order in which the cells are visited along sightline from the root cell to any destination cell (see figure 1).

The above mentioned approach of visibility preprocessing is based on research described in [1, 3, 4]. Since no extensions to existing geometry languages considering the visibility restrictions are known to the authors, it was of a great importance to create a data format to handle the visibility information [2].

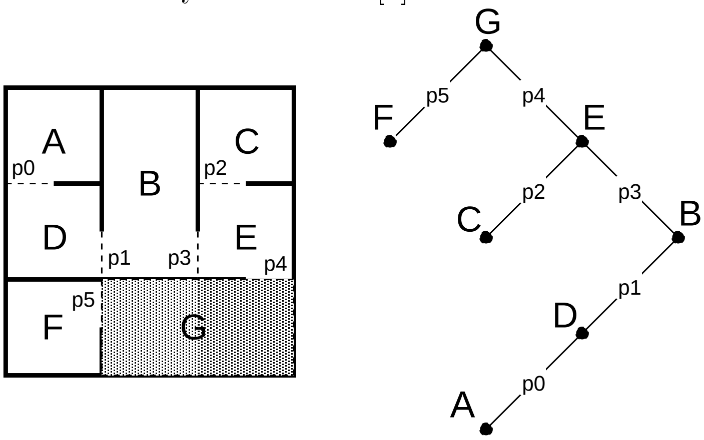

Figure 1: Portal enumeration and cell-to-cell visibility relations embedded in the visibility tree built for the cell G

2 VRML extension

Among existing languages for scene geometry description, VRML plays its role as a simple and efficient tool for representation of a geometry of general static models. It is widely used in the computer graphics industry. However, neither VRML 1.0 nor VRML 2.0 offers means for a description of constructs such as spatial subdivision of a MVE, cell-to-cell visibility relations and cell-adjacency graphs. Our goal was to introduce a language extension to VRML that would fill in this 'gap'. In the following paragraph there's a summary of the extension as it was implemented in the project.

All cells in the MVE are separated by boundaries. Portals reside right upon them. Boundary is in VRML extension defined as a group node. Two input fields *cellA* and *cellB* take the names of neighboring cells as arguments. Portal definitions are nested in boundary definitions. Both portals and cells must have assigned names since they are referenced to by visibility tree nodes and boundaries. The visibility tree definition is nested in the cell definition. The syntax of the visibility tree node allows us to express all the cell-to-cell visibility relations in the MVE. An example presenting all the new nodes is shown in segment 1.

3 Visibility preprocessing

The spatial subdivision of a MVE is constructed incrementally by selection of an *occluder* along which a splitting plane is induced, until no more candidates exist. In the current heuristics all the polygons larger than some threshold are treated as occluders. This choice makes the final cells of the subdivided model roughly correspond to actual rooms of the virtual environment. Thus, the spatial subdivision is based on binary space partitioning and uses data structure known as k-D tree with $k = 3$. The cell *boundaries* are created explicitly at each step of the spatial subdivision algorithm.

After the spatial subdivision is finished the portals on each boundary are enumerated by subtracting union of incident polygonal occluders from the boundary itself. The cell-to-cell visibility relation is established by involving a stabbing line algorithm for sequences of portals. The problem is solved by transforming portal edges to Plücker coordinates [5] which suitably parameterize lines in 3D and involving six dimensional linear programming task. Furthermore the *cell-to-frustum* visibility is determined. The computation of the potentially visible frustum induced by a portal sequence is based on conservative approximation of the anti-penumbra, when treating the first portal of the sequence as a light-source. The algorithm uses mapping to Plücker coordinates again and invokes the five dimensional convex polyhedra enumeration [4].

```
Boundary {
    cellA  frontCell
    cellB  backCell
    # the boundary geometry comes here
    DEF portalA Portal {
     # the portal's location upon the boundary comes here
    }
}
DEF frontCell Cell {
    VisibilityTree {
        VisibilityTree {
            through    portalA
            canSeeCell backCell
        }
        # more nested VisibilityTree nodes may come here
    }
    # the geometry of objects located in the cell comes here
}
DEF backCell Cell {
    ..
}
```

Segment 1: An example of a simple scene.

4 Implementation

VRML parser was built using the YACC and FLEX tools. Since the browser uses data structures different from the VRML tree ones, an additional pass had to be involved. This pass traverses the VRML tree, resolves any references, and builds the browser data structures. The browser as well as the preprocessor were both written in C++. The browser primarily uses Open GL for rendering the simulated scene walkthroughs, and optionally (on SGI workstations) it uses services of Open Inventor library to display the map of the simulated scene and the actual relations between its portions. A binary format for the language was invented to reduce the size of generated data files.

Scene	room	bez3	bez4
Vertices	60	5216	68328
Polygons	29	1304	17082
Subdivision computation time [s]	0.2	0.5	0.16
Cells	24	156	1956
Boundary polygons	73	360	4500
Portals	37	174	2346
Static visibility computation time [s]	0.5	5.0	850.0
Average number of statically visible cells	6	7	9
Speedup due to static visibility	4	22	217

Table 1: Test data

5 Conclusions and Future Work

The table 1 shows some test data characteristics after the visibility computations were made. The preprocessor was run on Silicon Graphics *Indigo*^{2} R4400/200 MHz workstation with 128 MB of RAM. All the test scenes were models of one-floor buildings. For an outdoor type of scenes there is a large increase in the time of static visibility computation (even several hours) since much greater number of portal sequence combinations needs to be considered. The table depicts that for the scene named "bez4" the average number of visible cells is 200 times lower than the number of cells that would have to be taken into account if the global visibility approach weren't exploited.

In the future, we would like to move on to the **Moving Worlds** (VRML 2.0) and introduce the same extensions to this language as we did to its previous version. Finally, we would like to build a server application that will answer to requests from browser-running workstations by sending them only the relevant portions of the MVE.

References

[1] Teller, S.: *Visibility Computations in Densely Occluded Polyhedral Environments.* Dissertation thesis – University of California at Berkeley, Berkeley, California, USA, 1992.

[2] Trefný, M.: *Formálnípopis 3D scén.* Master thesis – Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 1996.

[3] Teller, S.J., Sèquin, C.H.: *Visibility preprocessing for interactive walkthroughs.* Computer Graphics *(Proc. SIGGRAPH '91)*, Vol. 25, 4, (1991), 138-148.

[4] Teller, S.J.: *Computing the anti-penumbra cast by an area light source.* Computer Graphics *(Proc. Annual conference Series 1993)*, 236-246.

[5] Pellegrini M., Shor P.: *Finding stabbing lines in 3-dimensional space*, Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms, 1991, pp. 24–31, 1991.