Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations

by
John Milligan Airey

A Dissertation submitted to the faculty of the The University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
1990

Approved by

Frederick P. Brooks Advisor

James Coggins Reader

Jan Prins Reader

John Milligan Airey. **Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations** (Under the direction of Frederick P. Brooks, Jr.)

Abstract

Pre-processing some building models can radically reduce the number of polygons processed during interactive building walkthroughs. New model-space subdivision and potentially visible set (PVS) calculation techniques, used in combination, reduce the number of polygons processed in a real building model by an average factor of 30, and a worst case factor of at least 3.25.

A method of recursive model-space subdivision using binary space partitioning is presented. Heuristics are developed to guide the choice of splitting planes. The spatial subdivisions resulting from binary space partitioning are called *cells*. Cells correspond roughly to rooms.

An observer placed in a cell may see features exterior to the cell through transparent portions of the cell boundary called *portals*. Computing the polygonal definitions of the portals is cast as a problem of computing a set difference operation on co-planar polygons. A plane-sweep algorithm to compute the set operations, union, intersection and difference, on co-planar sets of polygons is presented with an emphasis on handling real-world data.

Two different approaches to computing the PVS for a cell are explored. The first uses point sampling and has the advantage that it is easy to trade time for results, but has the disadvantage of under-estimating the PVS. The second approach is to analytically compute a conservative over-estimation of the PVS using techniques similar to analytical shadow computation.

An implementation of the Radiosity lighting model is described along with the issues involved in combining it with the algorithms described in this dissertation.

Acknowledgements

Andries Van Dam and the Brown Computer Graphics Group introduced me to computer graphics. That was a once in a lifetime experience that I won't forget.

I feel fortunate to have been a part of the graphics cluster at UNC. Everyone in it, (and those who claim not to be in it), contribute to a great research environment. Dr. Fred Brooks, in particular, sets an example that has inpired many students, myself included.

Finally, my family and my friends have given endless support and I thank them all, with special thanks to Hali.

Table of Contents

List of Figures vi
List of Tables vii

I.Overview and Results 1
1.1 . UNC Building Walkthrough Summary 2
1.1.1 Modelling 3
1.1.2 User Interface 3
1.1.3 Display Compiler 3
1.2 Algorithm Overview 5
1.2.1 Properties of Architectural Databases 5
1.2.2 Model Space Subdivision 7
1.2.3 Potentially Visible Set calculations 9
1.3 Results 10
1.4 Speedup Methods Used by Other Interactive 3D systems 11
1.4.1 Vehicle Simulations 12
1.4.2 Object Display Applications 12
1.4.3 Environment Simulations 13
1.4.4 Other Pre-Computation-based Speedup Schemes 14
1.5 Guide to the Chapters 15

II. Model-Space Subdivision - searching for PVS coherence 16
2.1 The Partitioning Machinery 16
2.1.1 Partition.c: C code to compute interior node splitting planes and compute exterior node cell volumes 19
2.2 Computing Portals 23
2.2.1 Why Triangulation? 26
2.2.2 A Plane-Sweep Algorithm for Triangulation 27
2.2.3 Generalizing the Simple Triangulation Algorithm 31
2.2.4 Implementation 36
2.2.5 Applications 37

III. Potentially Visible Set (PVS) Calculation 40
3.1 Over-Estimating Visibility vs. Under-Estimating Visibility 42
3.2 Sampling Algorithms 43
3.2.1 Environment-Independent, or Fixed, Sampling 44
3.2.2 Environment-Dependent, or Targeted, Sampling 51
3.3 Over-Estimation Methods 52
3.3.1 Concave Shadow Casters 54
3.3.2 Multiple Shadow Casters (possibly concave) 56
3.3.3 A Naive Algorithm to Analytically Compute 2D Shadow Relationships 58
3.3.4 O'Rourke's worst case n^4 example 60
3.3.5 Extending the Analytic Algorithm to Three Dimensions . . . 61
3.3.6 Other approaches 62
3.3.7 An Over-Estimation Implementation 63
3.4 Impressions and Comparisons of the Implemented Algorithms . . . 66
3.5 PVS computation taking account of viewing direction 71

IV. A Radiosity Implementation 72
4.1 A Ray-Casting Approach 72
4.2 Interactive Light Manipulation 73
4.3 Using a Physically Based Lighting Model on Non-Physical Models . 74
4.4 Adaptive Refinement 74

V. Contributions and Future Work 78

References 81

Appendix A. Primary Source Code 86

List of Figures

Figure 1.1 Walkthrough goals 1
Figure 1.2 Overview of a virtual building system 3
Figure 1.3 Display File Compilation 4
Figure 1.4 Programs to compute model subdivision and PVSs 5
Figure 1.5 Simple three room floorplan 6
Figure 1.6 Subdivided floorplan and corresponding tree data structure . . . 9

Figure 2.1 graphical view of the operation of `partition()` 20
Figure 2.2 Data structure for polygons in axial planes 22
Figure 2.3 Portal Computation 23
Figure 2.4 Maintenance of the second plane-sweep invariant 29
Figure 2.5 Triangulation example 30
Figure 2.6 Catching edge intersections 32
Figure 2.7 A complex transition vertex 34
Figure 2.8 Modification to keep all edges present in the input 37
Figure 2.9 Fixing Cracks 39

Figure 3.1 The PVS problem 40
Figure 3.2 Four types of visibility 41
Figure 3.3 Ray-Line Segment Intersection 43
Figure 3.4 Hemi-cube sample pattern 45
Figure 3.5 Aligned hemi-cube sampling problems in a hallway 46
Figure 3.6 The jittered hemi-cube sampling pattern 47
Figure 3.7 Cosine-weighted (radiosity) hemisphere pattern (off-axis view) . 48
Figure 3.8 Cosine-weighted (radiosity) hemisphere pattern (top view) . . . 49
Figure 3.9 Linear radius-subdivision hemisphere pattern 50
Figure 3.10 Generating random target points on a polygon 52
Figure 3.11 A shadow cone 53
Figure 3.12 A shadow volume 54
Figure 3.13 Shadows cast by a concave polygon 55
Figure 3.14 Computing the umbra cast by a concave polygon 55
Figure 3.15 Multiple shadow casters I 56
Figure 3.16 Multiple shadow casters II 57
Figure 3.17 Linear Separators 57
Figure 3.18 Solution to multiple shadow casters 58
Figure 3.19 Example for 2-D algorithm 59
Figure 3.20 O'Rourke's worst case example 60
Figure 3.21 Linear separators in 3-D 61
Figure 3.22 Counter example for direct visibility segment operations 63
Figure 3.23 Reversing direction of projection 64
Figure 3.24 Occlusion operation 65

Figure 4.1 A hierarchical polygon 75
Figure 4.2 Choosing quadrilateral diagonals 76

List of Tables

Table 1.1 Speedup results 11
Table 3.1 Summary of timing comparisons 70

Chapter I

Overview and Results

This dissertation research was performed as part of the UNC Building Walkthrough project [Brooks86], [Brooks88], [Airey89d], [Airey90b]. The basic goal of the Walkthrough project is to create a virtual building environment, a system which simulates human visual experience with a building, without physically constructing the building. This is intended to aid the part of building design known as design development, when the architect explores, sometimes with the client, different design options.

Part of the goal of the Walkthrough project can be stated simply as, "accurate images, real-time feel, and big models", (figure 1.1). Choosing these objectives and attempting to work on the area of our system that was most deficient forced us to focus on the pre-processing partitioning approach, the pre-processing radiosity approach, and display-time adaptive refinement. The other part of the Walkthrough goal is a natural man-machine interface.

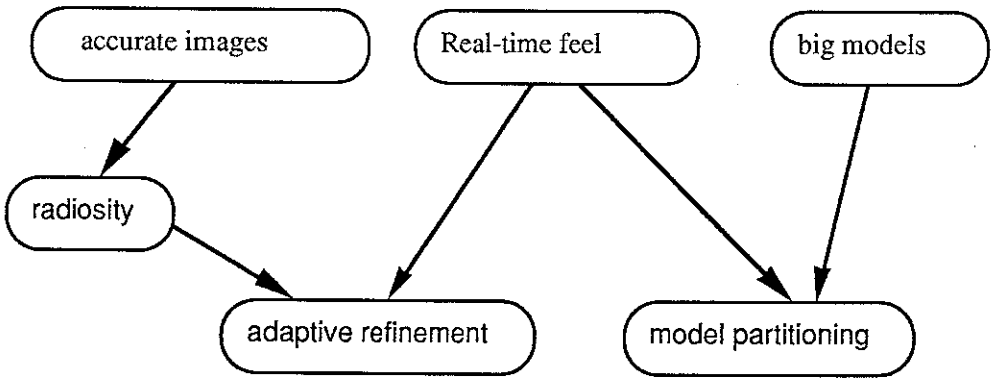

Figure 1.1 The Walkthrough goal, "accurate images, real-time feel and big models", pushed us to develop pre-processing methods to improve update rates, adopt a pre-processing shading model, radiosity, and apply the concept of adaptive refinement during display.

Many components of a building simulator, corresponding to many human senses, are important. My work within the Walkthrough project concentrates primarily on techniques to enhance visual simulation. Visual simulation can be separated into two components, a kinetic component, how the scene moves, and a realism component, how the scene looks. I sought to improve the kinetic visual experience by increasing update rates and to improve the illusion of reality by using the radiosity shading model [Goral84], [Cohen85], [Cohen88], [Wallace89], [Baum89], [Airey89]. For both components, one may use a

strategy of pre-computing as much work as possible prior to display. This dissertation describes techniques developed to pre-compute data structures which are used at display time to significantly increase the update rate. Our use of the radiosity shading model is described in chapter IV.

The Walkthrough research team finds natural motion, the kinetic component, to be a very important component of visual simulation. We observe user behavior to be qualitatively different at six updates per second as compared to behavior at one update per second.

• At one update per second (ups) or less, the system is painful to use. It is necessary to use an auxiliary two-dimensional floorplan display, or *map view,* to navigate.

• As the update rate increases from one ups to around twenty ups, interactivity appears to increase rapidly (superlinearly) before levelling off. At around six ups, the virtual building illusion begins to work. It is possible to navigate with only the three-dimensional display, or *scene view.*

In the fall of 1986, we were able to display a relatively spartan model of the Sitterson Hall Computer Science building, composed of 7125 polygons, at about four ups with the Pixel Planes 4 graphics machine [Fuchs85]. We wanted at least six ups and preferably twenty ups.

We perceived the slow update rate as the greatest system deficiency. I developed a process that automatically associates sets of viewpoints, called *cells,* with *potentially visible subsets* (PVS) of the model. The display subsystem has to process only the PVS that is associated with the cell containing the current viewpoint.

This process allows the Sitterson model to be displayed with at least 14 ups and often many more, depending upon the view, on the same Pixel-Planes 4 machine that in 1986 yielded four ups.

The update rate increase due to this process is dependent upon the data. Roughly speaking, the increase in update rate is proportional to the number of rooms in the building.

1.1 UNC Building Walkthrough Summary

A complete building walkthrough system has the following major parts (Figure 1.2).

• A modelling subsystem for the architect, where the canonical model is maintained. We use AutoCAD.

• An image-generation process for constructing a display file from the canonical model and generating the images. This includes the display-compiler subsystem and the display subsystem.

• An interface subsystem that allows the use of many different man-machine interface devices for controlling viewing parameters and illumination of the model.

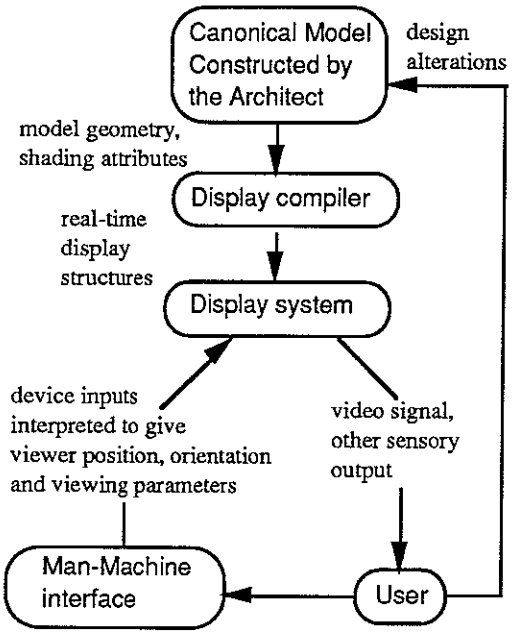

Figure 1.2. Overview of a virtual building system.

1.1.1 Modelling

The modelling subsystem must address ease of construction, (i.e. how many man-hours are required to create the model), model modification, and management of many different versions of the model (a task very similar to source code control [Tichy82]). For our modelling capabilities, we have adopted AutoCAD. This helps us establish partnerships with architects; these partnerships yield us datasets and system evaluation. We have only had limited partnerships so far.

1.1.2 User Interface

The interface subsystem must coordinate input devices with the display system. We have found that a flexible interface to a variety of devices is important. We already use several devices, often concurrently, and need to test new devices often. Frequent users such as architects may want complete freedom of motion (joysticks), while an infrequent user, e.g., a client, may desire a restrictive but more natural interface, (a treadmill with a head-mounted display or big screen display).

1.1.3 Display Compiler

The display compilation task must translate the geometric and surface attribute information from the model into a form suitable for rapid display and interaction with the interface devices. Figure 1.3 depicts our current process of converting an AutoCAD dataset into a form suitable for a virtual world system. The rectangles represent programs. The ovals represent data files. The type of the data file is depicted by the Unix convention of file name extensions.

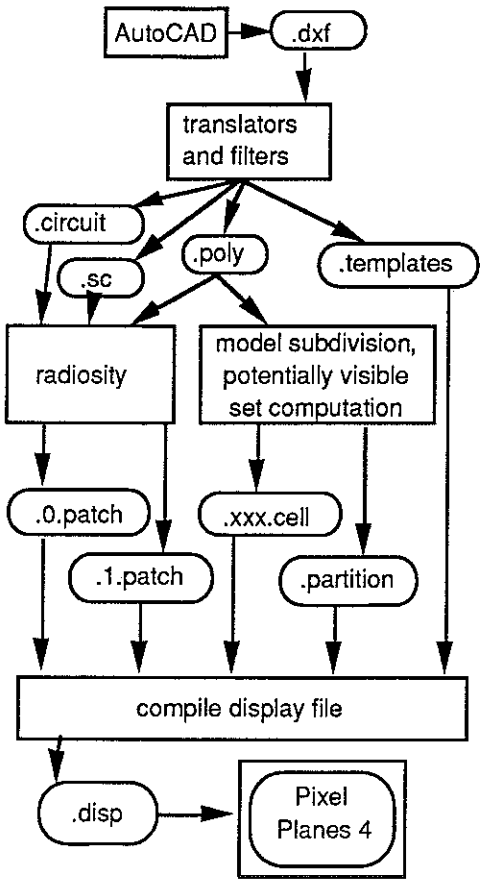

Figure 1.3 Display File Compilation

The AutoCAD external files (.dxf extension) are parsed and converted to a simple format which consists entirely of polygons (.poly extension). Separate files are generated which contain surface attribute information (.sc extension), and the sets of lights for which we want to compute independent radiosity solutions (.circuit extension). Another file (.template extension) allows Phigs+ -like structures to be incorporated into the final display file [Van Dam88]. Instances and templates allow movable objects such as furniture, or temporary diagnostic objects such as cell portals, to be easily added to the display. The display compilation splits into the radiosity process and the model subdivision process.

The radiosity process is briefly described in chapter IV.

The main right fork in Fig 1.1.3.1. is the model subdivision process. It generates a recursive subdivision of the model space [Samat90]. The result of this subdivision process is a tree of splitting planes (.partition extension). The .partition file defines subvolumes, or *cells,*of the model (.cell extension). Each of these cells is processed to determine the polygons that are potentially visible to an observer ranging freely inside the cell. These polygons are then associated with the cell. During display, the cell containing the current viewpoint is found, and only its associated polygons are rendered. Note that a particular polygon may be associated with many cells- i.e., visible from many subvolumes.

The box labelled *model subdivision, potentially visible set computation,* (figure 1.3) contains the programs that are the principal subject of this dissertation. Figure 1.4 is an

expansion of that box in figure 1.3.

The box labelled *compile display file* is a program whose main task is to combine all the information generated by previous programs into one coherent structure for display. Even though it serves primarily as a bookkeeper, it is quite a complicated program.

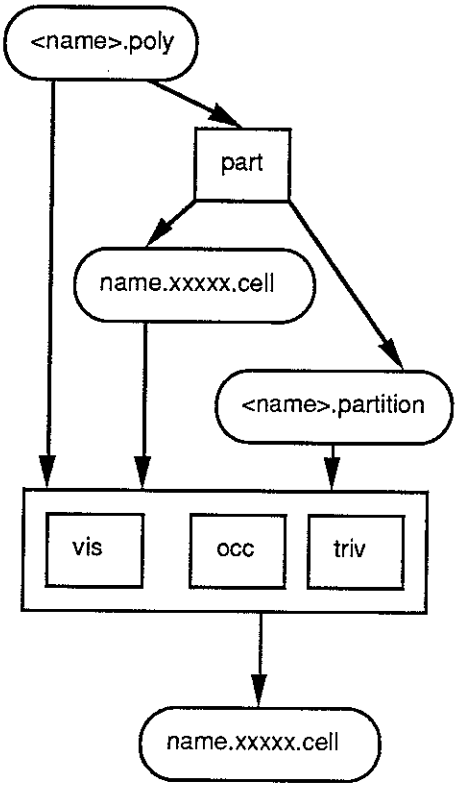

Figure 1.4. The programs used to compute the model subdivision or *part*ition, and identify potentially visible sets. Vis computes the PVS for a cell with a sampling algorithm. Occ computes the PVS for a cell with an analytic algorithm. Triv simply assigns the polygons inside a cell to the PVS and is usually used only to evalute the result of *Part*ition.

1.2 Algorithm Overview

1.2.1 Properties of Architectural Databases

Architectural databases possess special characteristics. Some of these properties follow. Techniques which exploit some of them are described later.

1. Properties of polygon configuration.

1.1. Most polygons are *axial*, i.e., normal to one of the coordinate axes.

1.2 Most polygons are rectangles.

1.3. Large planar surfaces are often structured into multiple, co-planar levels for modelling purposes, shading purposes, and realism detail purposes. For example, the

modeller may represent a ceiling with one polygon, but it may be diced into many smaller co-planar polygons to obtain a more accurately shaded image.

1.4. The set of polygons that appears in each view changes slowly as the viewpoint moves, except when crossing certain thresholds, e.g., doors and windows. Consider a simplified three room floorplan (Figure 1.5). The set of polygons visible in a 360 degree field of view from v_1 does not change much until one nears v_2. Then it stays roughly the same until one gets to v_3 where it changes radically again.

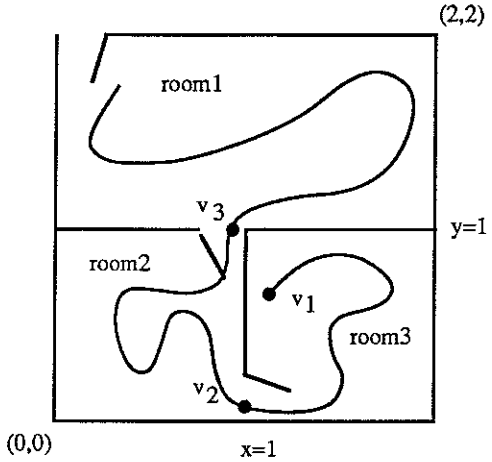

Figure 1.5 Simple three room floorplan with possible viewer path. The set of polygons visible in a 360 degree field of view from v1 does not change much until one nears v2. Then it stays roughly the same until one gets to v3, where it changes radically again.

2. Properties related to the inside-out viewing nature of environment models.

2.1. Many viewpoints have a large portion of the model outside the field of view.

2.2. Surface interreflections in shading calculations are very important for spatial comprehension inside a building.

3. Properties related to depth complexity.

3.1 Any image computed from an interior viewpoint will have many surfaces covering every pixel.

3.2. Many surfaces are completely hidden; they do not contribute to the image at all. Besides the obvious example of back-facing surfaces, forward-facing surfaces in the next room or on the floor above or below do not contribute to the image.

3.3.The fraction of hidden surfaces in a building is basically independent of tesselation required for shading. The number of surfaces hidden is a function of the number of walls, floors and ceilings. Tesselating them uniformly into small patches does not change the fraction of hidden surfaces. Thus the methods outlined in this dissertation should get results independent of shading tesselation.

A simple quantitative analysis comes from [Gharachorloo89], [Sutherland74]. Given P polygons projected onto the image plane, of average area A, a total of $P \cdot A$

pixels must be calculated. Assuming the scene covers a screen of N pixels and that the polygons are overlaid in D layers everywhere, we have: $P \bullet A = N \bullet D$, where D is thedepth complexity. If walls are represented with small individual tiles for shading purposes, then P increases while A decreases and D remains constant.

Additional detail such as fancy window frames and baseboard moldings, etc.has a similar but less easily analyzed effect. The fraction of hidden surfaces increases only slightly.

1.2.2 Model Space Subdivision

The pre-processing algorithm automatically subdivides model space or equivalently, viewpoint space, into *cells*. Define the union of visible polygons for all the viewpoints in a cell as the *potentially visible set* (PVS) for that cell. For any viewpoint in the cell, rendering the potentially visible set for that cell generates an image with no missing polygons. Since the size of the potentially visible set is usually much smaller than the size of the model it came from, it takes less time to render. The rendering process involves at least transforming the polygon to the correct perspective view, and then scan-converting it into the frame buffer if it appears in the viewing frustum and faces the viewer.

For the simple three room floorplan given in figure 1.5, the labelled rooms approximate what one wants in a cell. If the doors were closed (and they had no glass) then one could simply render the polygons in room2 when the viewpoint is in room2 and get roughly a three-fold increase in system update time. If the doors are open, one must add the polygons that can be seen through the doors from room2 to the potentially visible set for that cell.

Even from this simple example it is obvious that certain model subdivisions are better than others. The subdivision process should try to satisfy the following objectives.

Objective 1. Minimize the size of the potentially visible sets. One wants cells whose potentially visible set is not much larger than the visible set of any one viewpoint in the cell. This ensures the best possible speedup.

Objective 2. Minimize the number of cells; split cells only if the resulting child cells have significantly smaller PVSs.

Besides these loosely stated objectives, a subdivision algorithm must satisfy other restrictions.

Restriction 1. It must be easy to find the cell that contains the current viewpoint. This operation is performed for every update during display.

Restriction 2. It must be automatic. No hand-tooling allowed. Architects cannot afford the time and may not have the expertise to hand-weave databases for esoteric display algorithms.

Restriction 1 implies the use some type of data structure suitable for range searching. I chose recursive binary partitioning planes. Furthermore, it is sufficient to restrict their orientation to be normal to one of the coordinate axes. Other options include a regular 3D grid, or adaptive space subdivision techniques such as octtrees or k-d trees [Mehlhorn84], [Samet90]. Other proposed approaches utilize the fact that adjacent cells are typically accessed in a sequential order and usually in the horizontal plane. This would allow a more

adaptive subdivision algorithm [Prins90]. Any of these data structures allow the cell containing a viewpoint to be found quickly.

To satisfy Objective 1 and Restriction 2, I devised a heuristic function to choose the splitting planes used in the recursive binary subdivision scheme. Since one wants a splitting plane that is largely opaque, one may limit the choice of splitting planes to those that contain model polygons. The function evaluates each plane containing a polygon for its suitability as a separating plane. Criteria considered are

• how evenly the plane separates the model, called the *balance* of the split,

• how well the plane hides the two sides from each other, called the *occlusion* factor of the split. For example, a floor hides better than a wall with a door in it.

• how little the plane splits individual polygons, since polygons that are split will have to be put in the potentially visible sets of both partitions. This is called the *split* factor.

The metrics used quantify these criteria between 0 and 1. A linear combination of these values, with the occlusion factor weighted most heavily, has proven to be successful, e.g.,

partition priority = .5*occlusion + .3*balance + .2*split.

To satisfy Objective 2, the recursive process terminates when no partitioning plane has a partition priority exceeding a user-defined threshold or when a maximum tree-depth is exceeded. The process generates a tree with interior nodes representing binary separating planes and leaf nodes representing cell volumes. Several people have noted that feedback from the second phase of the pre-processing algorithm, described in the next section, could be used to adjust the model-subdivision [Prins90].

If one runs this function on the "planes" in our simple example floor plan, the wall that separates rooms 2 and 3 from room 1, the plane y=1, has a higher partition priority than the wall that separates room 2 from room 3, the plane x=1, based on its higher occlusion factor. This yields two cells, room 1 and the combination of room 2 and room 3. Recursively evaluating our heuristic function on these two cells suggests that room 2 and room 3 can be further split into two cells along the plane x=1 (figure 1.6).

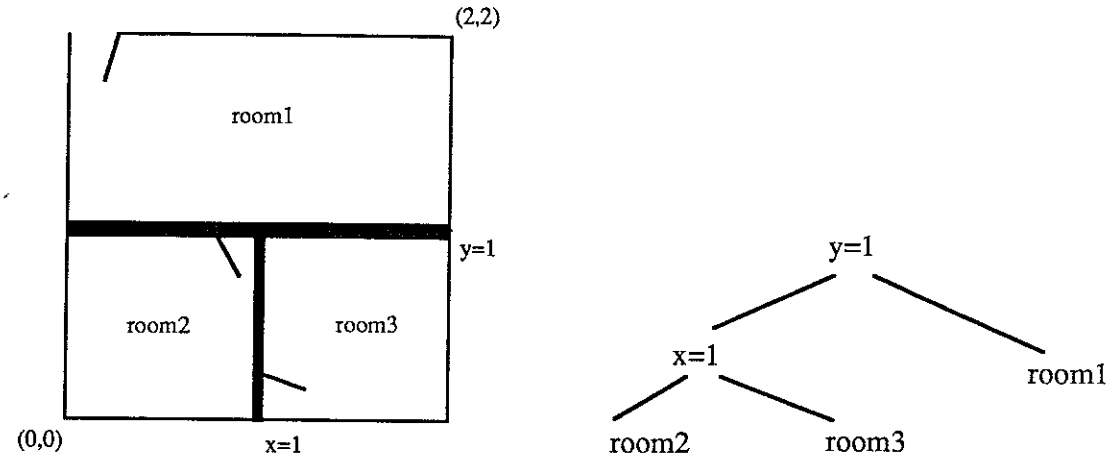

Figure 1.6. Left, the subdivided floor plan. Room1 is separated from room2 and room3 by the plane y=1. Room2 is then separated from room3 by the plane x=1. Right, the corresponding tree data structure. Interior nodes represent splitting planes and leaf nodes represent cell volumes.

1.2.3 Potentially Visible Set calculations

After model-space subdivision, the subset of the model potentially visible to an observer inside each cell is computed and stored with the cell. If the cell is completely sealed, that is, its boundary is composed of opaque surfaces, then this is easy to do. The potentially visible set for the cell is simply the set of polygons that intersect the cell. However, if the cell has holes in its boundary, called *portals,* then the problem is more difficult.

In our simple example, the only portals are doors. In real-life datasets, hallways, stairwells, elevator shafts, windows, and oddly shaped rooms give rise to other portals. Portal geometry is defined only implicitly as the absence of polygons in the boundary of the cell. Explicit polygonal descriptions of the portals are obtained by computing the boolean difference of the cell boundary and polygons lying in the cell boundary planes. Algorithms that perform boolean set operations on co-planar sets of polygons can compute the explicit polygonal definitions of the portals [Ottman85], [Weiler81], [Airey89c]. Section 3.2 describes one such algorithm.

We call the question of what external polygons one should add to the potentially visible set for a cell the *polygon-to-cell* visibility problem. This can be reduced to another problem. One really has to worry only about what can be seen from the cell portals, which can each be represented with polygons. Taking the union of what is visible from all the cell portals of a cell solves the cell-to-polygon visibility problem.

Unfortunately, this is also a difficult problem. One needs to know what is visible from a portal which is an *area,* an infinite albeit bounded set of viewpoints. Call this problem the *polygon-to-portal* visibility problem.

Computing the PVS is equivalent to identifying the polygons that receive direct illumination from an area light source [Nishita85], [Chin89], (the portal acts as an area light source). This is similar to the underlying problem of surface-to-surface reflection that

appears in models of global lighting effects, such as the radiosity lighting model. Other researchers have examined a related problem in two dimensions which deals with visibility from an edge [Avis86], [O'Rourke87].

Since algorithms to compute the *exact* solution for the portal-to-polygon visibility problem are very complex, I have developed two complementary classes of algorithms to compute approximations to the exact solution. These are detailed in Chapter III.

One class of algorithms uses point sampling and may underestimate the set of polygons to add to the cell's potentially visible set. This is analogous to the use of point sampling in radiosity solutions. In fact, the same ray-polygon intersection library is used in the Walkthrough radiosity implementation [Airey90b].

Another class of algorithms establishes occlusion relationships among polygons. This is similar to the computation of shadow volumes [Crow77]. Since exhaustive computation of shadow volumes is expensive, the only alternative is to compute a partial solution. This may overestimate the set of polygons to add to the cell's potentially visible set. Since the exact solution is bracketed by these two algorithms, one hopes they can be combined into a more accurate algorithm in the future.

Currently, these approaches are both expensive. In practice we use mostly the sampling-based methods because they are less expensive than the occlusion-relation based methods, although they do occasionally miss potentially visible polygons.

Just as a radiosity solution can be accelerated with graphics workstation hardware [Baum90], the ability to generate environment samples quickly can be used to accelerate a sampling-based approach. For radiosity, the environment samples are not colored pixels, but polygon identifiers. In the case of radiosity, the polygon identifiers and the pixel location are used to construct form factors. In this case, after rendering the view from a sample point on the portal, any polygon whose identifier appears in the frame buffer is added to the PVS under construction. Note that reading the sample values, the polygon identifiers, back from the frame buffer must be efficient also; low bandwidth here can defeat this idea.

1.3 Results

I have run this algorithm on a few databases and compiled statistics to document the speedup results. The databases include

• A 7125-polygon model of Sitterson Hall, *Sitterson1*. Walls are represented by single polygons with separate colors for the front-facing and back-facing sides. AutoCAD was not used for this model. (Modelled by Dana Smith from plans by Phil Freelon of O'Brien and Atkins)

• A second model of Sitterson Hall was constructed with AutoCAD, *Sitterson2*. This model consists of over 22,000 polygons. It consists mostly of polygons that are designed to be seen from only one side. The walls have thickness and are modelled with a pair of polygons. The lobby portion of this model, *lobby2*, appears in a Siggraph '89 video [Airey89b]. *Lobby2* has 3949 polygons. (Modelled by Penny Rheingans.)

• The Orange United Methodist Church Fellowship Building. An early version with 7812 polygons is called *Church1*. (Model by Penny Rheingans from plans by Wesley McClure and Craig Leonard of McClure NBBJ.)

• A later version of the Church consists of over 12,000 model polygons. Since the radiosity process increases the number of shading patches that must be stored in display memory by about an order of magnitude, a 6037-polygon subset was used because of display memory limitations. This subset, called *Church2,* consists of the main meeting hall and a few adjoining rooms, including a fully furnished kitchen.

• This dataset was later re-enlarged to 6974 polygons by adding part of the basement. This version of the church is called *Church3*. (John Alspaugh helped model and manage later church databases). The *Church3* database is featured as a demo for the SGI Iris 4D VGX series machines.

Table 1.1. summarizes the results of the model-subdivision algorithm on these datasets.

The best results, as one would expect, comes from the two complete models, *Sitterson1* and *Church1*, buildings with many rooms, easily detected pockets of PVS coherence. These have the highest depth complexity. *Sitterson1* was subdivided into 269 cells. The cell with the largest potentially visible set had 2195 polygons to display. The average number of polygons to display was a little more than 230. The speedup was 3.25 in the worst case and 30 in the average case.

Data	polys	cells	polys/cell avg.	polys/cell max.	speedup avg.	speedup min.
Sitterson1	7125	269	230	2195	30.98	3.25
Lobby2	3949	54	466	2550	8.47	1.55
Church1	7812	108	291	2055	26.85	3.80
Church2	6037	16	1887	3477	3.20	1.74
Church3	6974	106	767	3768	9.09	1.85

Table 1.1. Summary of Model Subdivision speedup results.

The additional display memory required to store the data structure generated by the visibility pre-computation is reasonable, about 20%. The main requirement is the need to store potentially visible sets for each cell. Since several cells may see each polygon, there is a potential for large display memory use unless polygon descriptions are shared among cells. The polygons need be represented only once; the PVS for each cell is composed of references to the polygons. From the numbers in Table 1.1 for the Sitterson model, one can see that, on average, about ten cells can see each polygon. This means one needs about 10 more words per polygon to store the pointers. Since the storage required for a color-interpolated quadrilateral is about 200 bytes, the total display file size is increased by about 20%. The storage requirement for the other databases is less.

1.4 Speedup Methods Used by Other Interactive 3D Systems

Most interactive 3D applications can be implemented with some general-purpose technology. All use some form of the well-known image generation pipeline [Molnar89].

In general, however, it is possible and usually desirable to capitalize on certain properties of a given application to increase performance. Specialized hardware may be built to capitalize on application specifics if economics allow it. A less drastic solution is to build specialized software.

When a software approach is used, the strategy of pre-computation is common. Some classes of applications and the methods they use to achieve usable speed follow.

1. *Vehicle* simulations, such as flight simulators, large ship simulators for harbor pilots, or automobile simulations [Schachter83],[Deyo88].

2. *Object* simulation for the visual appearance of mechanical parts or anatomical structures.

3a. Man-made indoor *environment* simulations for buildings or ships, (UNC's Building Walkthrough is in this category).
b. Man-made outdoor *environment* simulations for parks, planned communities, etc.

1.4.1 Vehicle simulations

Vehicle simulations, such as flight simulators, can attain 60 Hz update rates with textured, anti-aliased images including atmospheric effects such as haze and fog. To help achieve such high performance, designers often assume constraints on the models. Grossly-accurate, mostly-static databases composed of relatively few, large, textured polygons are typical of landscapes used in flight simulators [Schachter83],[Abram87].

Because the emphasis is on the simulation experience and not the model, it is reasonable to spend a large amount of time hand-tooling a model that fits the constraints of the system. Since the model will be used over and over again, modeling expense is amortized over many simulations.

Limited depth complexity and large polygons make hard-wired, image-order, or scan-line algorithms, and priority graph approaches very effective[Sutherland74]. The bottleneck in the most general purpose graphics workstations [Akeley89], which typically use the Z-buffer method for hidden surface elimination, is the limited bandwidth to the frame buffer and Z buffer. Image-order algorithms that work on databases with low depth complexity can avoid this bottleneck because visibility is determined before information is written to the frame buffer [Molnar89].

The lighting models used in flight simulators may be relatively simple because direct sunlight is responsible for most of the shading effects seen on the landscape. Surface interreflections, which are responsible for much of the expense in other lighting models, such as ray tracing or radiosity, can be disregarded [Hall88].

1.4.2 Object display applications

Graphics workstations suitable for computer aided engineering (medical and mechanical) have been available for several years. They have taken advantage of the application specifics to achieve the needs of the users with minimal cost. The displayed data is typically not an environment. It is a tie-rod or distributor cap or kidney some other object than can be conveniently viewed through the computer window. An object may be comprised of many small polygons derived from a higher order surface description. Since the object typically is inspected individually and is mostly convex, expensive lighting