# Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments[1]

John M. Airey, John H. Rohlf[†], and Frederick P. Brooks, Jr.

Department of Computer Science, Sitterson Hall
University of North Carolina
Chapel Hill, NC 27599-3175

[†]Silicon Graphics Computer Systems
Mountain View, CA 94039-7311

## Abstract

Two strategies, pre-computation before display and adaptive refinement during display, are used to combine interactivity with high image quality in a virtual building simulation. Pre-computation is used in two ways. The hidden-surface problem is partially solved by automatically pre-computing potentially visible sets of the model for sets of related viewpoints. Rendering only the potentially visible subset associated with the current viewpoint, rather than the entire model, produces significant speedups on real building models. Solutions for the radiosity lighting model are pre-computed for up to twenty different sets of lights. Linear combinations of these solutions can be manipulated in real time. We use adaptive refinement to trade image realism for interactivity as the situation requires. When the user is stationary we replace a coarse model using few polygons with a more detailed model. Image-level linear interpolation smooths the transition between differing levels of image realism.

**CR Categories and Subject Descriptors:** I.3.3 [Computer Graphics]: Picture/Image Generation - display algorithms, viewing algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - color,shading and shadowing.

**Additional Key Words and Phrases:** model-space subdivision, potentially visible, radiosity, adaptive refinement.

[1]This work was supported by NSF Grant#CCR-8609588 and ONR Grant#N00014-86-K-0680



## 1. Introduction

Our basic goal is a virtual building environment, a system which simulates human experience with a building, without physically constructing the building. Many components of a building simulator, corresponding to many human senses, are important. We concentrate on techniques to enhance visual simulation.

We find natural motion 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 1 frame per second, the system is painful to use. It is necessary to use a two-dimensional floorplan display, or *map view*, to navigate.

• As the frame rate increases to around 20 frames per second interactivity appears to increase rapidly before leveling off. At around 6 frames per second the virtual building illusion begins to work. It is possible to navigate with only the three-dimensional display, or *scene view*.

However, realistic images are also important. We use a lighting model that realistically simulates the complicated diffuse, interreflections within a building interior. Furthermore, studying real buildings demands large, detailed models.

Each of these simulation enhancements greatly increases the number of primitives the display subsystem must process and has an adverse effect on interactivity. To help our display subsystem cope with this increased load and remain interactive, we use two stategies. We pre-compute as much work as possible and use adaptive refinement techniques during display [Bergman 86]. The resulting combination is a system that attempts to keep the update rate above about six frames per second while providing as realistic an image as possible.

Section 2 describes the system level organization in a concrete, detailed manner. This is followed by sections which cover the concepts and algorithms used in the system.

Our system uses pre-computation in two ways. A new process automatically associates sets of viewpoints with potentially visible subsets of the model [Airey 90]. The display subsystem has to process only the potentially visible subset of the model that is associated with the current viewpoint. The update rate increase due to this process proportional to the depth complexity, hence the size of the building modelled. For a 7125-polygon model of Sitterson Hall, we have achieved speedups of not less than 3.25 in the worst case, and 30 in the average case, depending upon the viewpoint. The process is based on model-space subdivision and volume-to-polygon visibility testing. We describe this process in Section 3.

The radiosity lighting model accurately simulates the diffusely reflecting surfaces that dominate building interiors [Goral 84]. Because diffuse reflection is view-independent, the shading can be pre-computed for finite areas or patches. A resampling filter transfers patch radiosity values to color values at patch vertices. The color values are interpolated across the patch during display to get a smooth-shaded image. Display systems with hardware support for color interpolation can display radiosity patches rapidly. Since no lighting-model dot products need to be performed, radiosity patches can be rendered faster than Phong-shaded or Gouraud-shaded polygons. We present our variation of an algorithm to compute radiosity in Section 4 and relate our practical experiences using radiosity as a tool for virtual worlds. Our system allows the user to interactively and independently modulate 20 different lighting circuits in models with tens of thousands of radiosity patches.

Adaptive refinement provides another way to increase system performance. The user often stops moving to examine the current view more closely. We take advantage of the reduced need for interactivity to increase realism. For example, the radiosity process dices large polygons into smaller finite elements or *patches*. We maintain both the original polygons and the derived patches in a form we call *hierarchical polygons*. Image quality dynamically improves by replacing the original model polygons with the derived patches whenever the viewer is stationary. We further describe adaptive refinement techniques in Section 5.

## 2. System Overview

We are iteratively developing a virtual world system that lets architects and their clients explore a proposed building during the design-development phase [Brooks 86]. Although this paper concentrates on the techniques we have used to advance image realism, increase the frame rate and integrate advances in these two areas, here are the larger aims of the research project. A complete system has the following major parts (Figure 1).

- A modelling system 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 lighting circuit settings.

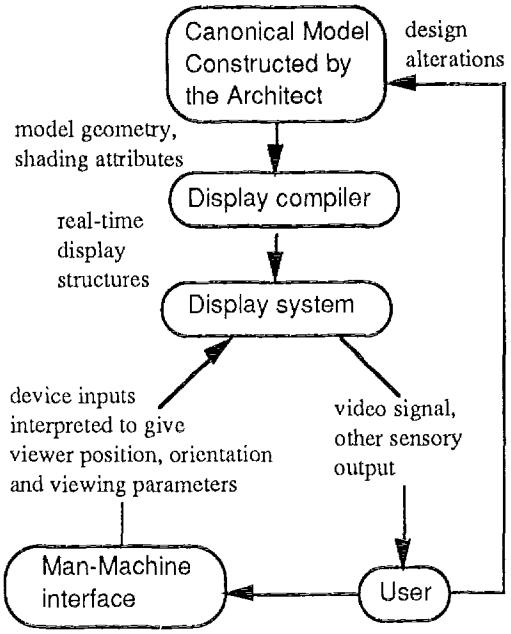



**Figure 1. Overview of a Virtual Building System.**

### 2.1 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). For our modelling capabilities, we have adopted AutoCAD. This helps us establish partnerships with architects for datasets and system evaluation.

### 2.3 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). We expect to report on the research results of the interface subsystem in another publication.

### 2.4 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 2 depicts the 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 filename extensions.

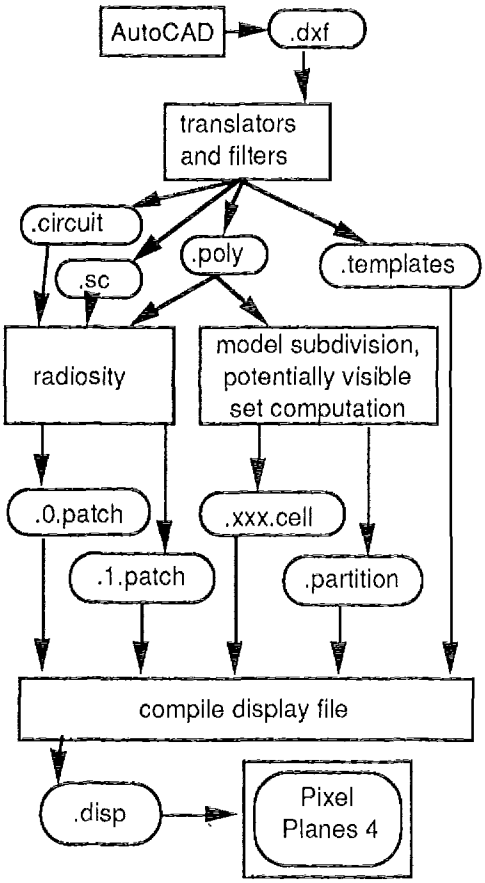



**Figure 2. Display File Compilation in UNC Walkthrough System.**

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. The display compilation forks into the radiosity process and the model subdivision process.

The radiosity system is described in Section 4. It processes .poly files by automatically dicing the polygons into derived patches and computing color values for each patch for each independent light circuit. The original polygons are given color values by averaging the values of their derived patches. The patches retain indices to parent polygons. The shaded polygons are written to a file with a .0.patch extension and the derived patches are written to a file with a .1.patch extension. This information is used by the next program to construct hierarchical polygons. A *hierarchical polygon* is a polygon that has an associated list of polygons that may be used to replace the polygon. We have experimented only with one level of refinement and with refining all polygons to the same level simultaneously, but our data structures and display code allow arbitrary subdivision for each polygon independently.

The model subdivision process is described in Section 3. It generates a recursive subdivision of the model space. 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.

## 3. Display Acceleration by Model-Space Subdivision and Volume-to-Polygon Visibility Testing

Architectural databases possess special characteristics. We list some of these properties here and then describe how we exploit them.

1. The model is changed much less often than the viewpoint, which makes pre-processing desirable.

2. Many buildings have high average depth complexity. Any image computed from an interior viewpoint will have many surfaces covering every pixel. A related observation is that most of the model does not contribute at all to any given image. Furthermore, the depth complexity of a building is basically independent of tesselation due to shading and independent of the amount of detail modelled.

3. Most polygons are axial, that is parallel to two of the coordinate axes. Additionally many polygons are rectangles.

4. The set of polygons that appears in each view changes slowly as the viewpoint moves, except when crossing certain thresholds, e.g., doors, which we call *portals*.

Consider a simplified three room floorplan (Figure 3). The set of polygons visible in a 360 degree field of view from $v_1$ does not change much until we get near $v_2$. Then it stays roughly the same until we get to $v_3$ where it changes radically again.

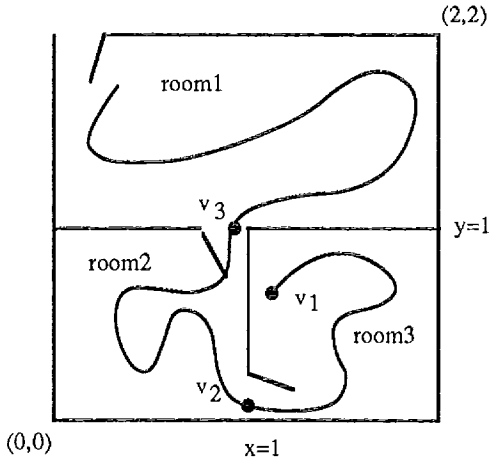



**Figure 3. Simple Three Room Floorplan with Possible Viewer Path.**

5. 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 represent the shading determined by radiosity calculations.

6. Because the viewpoint is usually inside the building, the role of surface interreflections in shading calculations are very important for spatial comprehension.

### 3.1 Model Space Subdivision

We automatically subdivide 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* 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.

For the simple iconic example in figure 3, the labelled rooms approximate what we want in a cell. If the doors were closed (and they had no glass) then we could simply render the polygons in room2 when the viewpoint is in room2 and we would get roughly a three-fold increase in system update time. If the doors are open, we have to 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. We want cells whose potentially visible set is not much larger than the visible set of any one viewpoint in the cell. This ensures that we get the best possible speedup.

**Objective 2.** Minimize the number of cells. We can not afford a cell for every viewpoint, so Objective 1 must be countered with the objective that two adjacent cells should not have similar potentially visible sets.

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 during display, and every cycle it uses increases system latency.

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

Restriction 1 implies we must use some type of data structure suitable for range searching. We chose recursive binary partitioning planes. Furthermore, we found it 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 octrees or k-d trees [Mehlhorn 84]. Any of these data structures allow the cell containing a viewpoint to be found quickly.

To satisfy Objective 1 and Restriction 2, we devised a heuristic function to choose the splitting planes used in the recursive binary subdivision scheme. Since we generally want a splitting plane that is largly opaque, we 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, which we call the *balance* of the split,

- how well the plane hides the two sides from each other. A floor hides much better than a wall with a door in it. We call this the *occlusion* factor of the split,

- 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 we use 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:

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

To satisfy Objective 2, the process terminates when no partitioning plane has a partition priority exceeding a user-defined threshold or when other limits, such as tree-depth, are exceeded. The process generates a tree with interior nodes representing binary separating planes and leaf nodes representing cell volumes.

If we ran this function on the "planes" in our simple example floor plan, the wall that separates room 2 and room 3 from room 1, the plane y=1, would have 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 4 and 5).

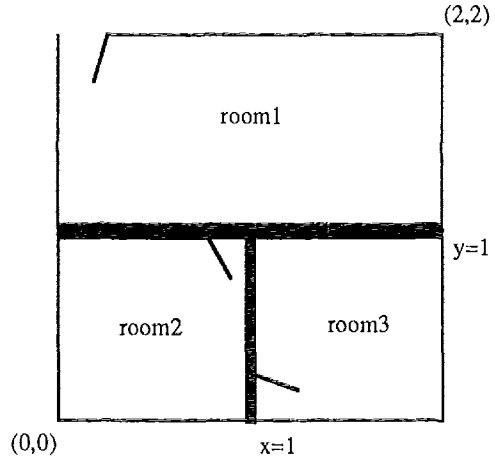



**Figure 4. The Subivided Floor Plan.**

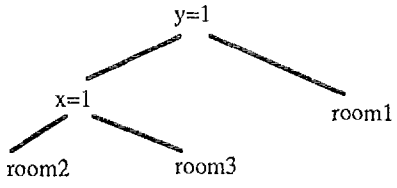



**Figure 5. The Corresponding Tree Data Structure for Figure 4. Interior Nodes Represent Splitting Planes and Leaf Nodes Represent Cell Volumes.**

### 3.2 Volume-to-Polygon Visibility Testing

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 actual doors. In real-life datasets, hallways, stairwells, windows, and oddly shaped rooms give rise to other portals. Algorithms that compare co-planar sets of polygons can compute the actual polygonal definitions of the portals [Ottman 85],[Weiler 81].

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

Unfortunately, this is also a difficult problem. We need to know what is visible from an *area,* an infinite albeit bounded number of viewpoints. We call this problem the view*area* problem.

This is fundamentally equivalent to computing the polygons that receive direct illumination from an area light source [Nishita Nakamae 85]. Other researchers have examined a related problem in two dimensions which deals with visibility from an edge [Avis 86], [O'Rourke 87].

Since algorithms to compute the *exact* solution for the viewarea problem are complex and inefficient, we have developed two complementary classes of algorithms to compute approximations to the exact solution. These are detailed in [Airey 90].

One class 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, it is implemented with the same ray-polygon intersection library used by our radiosity implementation, Section 4.

Another class establishes occlusion relationships among polygons. This is based on the computation of shadow volumes [Crow 77]. Since exhaustive computation of shadow volumes is expensive, we 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, we hope they can be combined into a more accurate algorithm in the future.

Currently, the problem these approaches are expensive. In practice we usually use only the sampling based methods, because they are less expensive than the occlusion-relation based methods.

Note that a workstation with Z-buffer hardware, the ability to scan convert polygon identifiers rather than color values, and the ability to read back the identifiers could be used to accelerate a sampling-based approach just as a radiosity solution can be accelerated.

### 3.3 Speedup Results

We 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. 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. This model consists of over 22,000 polygons. For the most part it consists 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 appears in a Siggraph '89 video. The lobby has 3949 polygons. (Modelled by Penny Rheingans.)

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

• A later version consists of over 12,000 polygons. Since radiosity increases the number of polygons that must be stored in display memory by about an order of magnitude, we were forced to use a 6037 polygon subset because of display memory limitations. This subset, which we call Church2, consists of the main meeting hall and a few adjoining rooms, including a fully furnished kitchen. See the description of Color Plates in Section 6. (John Alspaugh finished modelling the church databases).

Table 1. summarizes the results of the model-subdivision algorithm on these datasets.

The best results are from the Sitterson database. It 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. |
|---|---|---|---|---|---|---|
| Sitt. | 7125 | 269 | 230 | 2195 | 30.98 | 3.25 |
| Lobby | 3949 | 54 | 466 | 2550 | 8.47 | 1.55 |
| Chrch1 | 7812 | 108 | 291 | 2055 | 26.85 | 3.80 |
| Chrch2 | 6037 | 16 | 1887 | 3477 | 3.20 | 1.74 |

**Table1. 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. We represent the polygons once; the display list for each cell is composed of references to the polygons. From the numbers in Table 1 for the Sitterson model, we see that, on average, about ten cells can see each polygon. This means we need 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.

## 4.0 Radiosity Shading in an Interactive System

The radiosity lighting model has several properties that make it desirable for virtual building environments.

• It accurately models the diffuse interreflections that dominate the interior of a building.

• The lighting information may be pre-computed and stored as color values at polygon vertices. These values are linearly interpolated by hardware during display. This effectively eliminates any lighting calculations at display time, and yields rapid rendering.

• The process is a linear system. Thus the contributions of several different light sources may be computed independently. A linear combination of these solutions may be computed during display, allowing the user to brighten and dim lights. This gives an added dimension of interactivity at little cost.

For several years the best-known solution to the radiosity lighting model used quadratic time and quadratic space [Cohen 85]. This made it prohibitive for use in practical systems. The shooting algorithm used in the progressive refinement solution to radiosity makes radiosity

practical [Cohen 88]. The algorithm runs in linear space, and usually only linear time is required to converge to an acceptable solution. It is no longer a research curiosity but a tool for virtual environments.

### 4.1 A Ray-Casting Approach

We use a modified shooting approach to compute the radiosity solution. The sampling process uses ray-casting based on a jittered hemispherical distribution, rather than a Z-buffer based hemi-cube [Airey, Ouh-young 89].

At each iteration step, we adapt the resolution of the hemispherical sampling distribution as a function of unshot radiosity to keep the radiosity per ray constant. Airey and Ouh-young observed, empirically, that the unshot radiosity at each step decreases as a negative exponential. Thus, the number of rays fired at each step also decreases as a negative exponential.

A new ray-polygon intersection algorithm tuned to architectural databases accelerates the ray casting. It takes advantage of characteristics such as the large proportion of axial rectangles. The basic idea can be easily described in two dimensions. Consider the problem of computing the closest intersection of the ray and line segments depicted in figure 6.

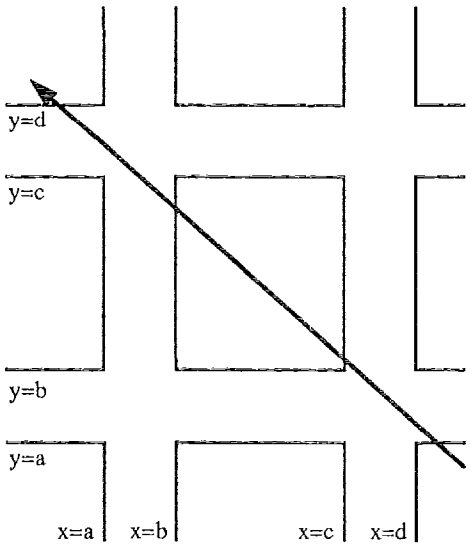



**Figure 6. Ray-Line Segment Intersection.**

The ray intersects the lines containing segments parallel to the x-axis, in order, from bottom to top. Similarly, the ray intersects the lines containing segments parallel to the y-axis, in order, from right to left.

This suggests a data structure which groups line segments lying in the same line together. Each set of parallel lines is sorted along the normal direction. This data structure can be pre-computed.

To compute the intersections in order, we check the intersection parameter for the closest line in each of the two sorted lists. In our example, the line y=a is closer than the line x=d. When we check the segments lying in the line y=a, we halt and report the intersection.

If we had not found an intersection, we would have computed the intersection parameter for the next line in the x-parallel list, y=b, and compared it with the intersection parameter for the line x=d. We continue to effectively merge the two lists until we find an intersection.

The process works in three dimensions similarly. The small percentage of non-axial polygons are put into a standard BSP tree. After an intersection is found for the axial polygons, the BSP tree is searched from front to back until we find an intersection or exceed the intersection parameter found for the axial polygons.

The primary advantage of ray-casting sampling algorithms is flexibility. We have been able to experiment with light-emitter distributions other than true diffuse emission, such as spotlight-like distributions, with only small changes in our software. Wallace, et al., use ray-casting to sample the light source from the model vertices to decrease solution errors due to limited sampling distributions [Wallace 89]. They also note other advantages, such as the ability to use exact parametric descriptions of objects.

### 4.2 Interactive Light Manipulation

We have extended our radiosity program to compute the contributions of several different light circuits. For each patch we simultaneously compute a vector of radiosities, one entry for each light circuit. Since a value for the red, green and blue channels must be stored for every patch for every independent set of lights, the storage requirement is large. On workstations used to compute the radiosity solution, large physical memories and virtual memory ease this problem. However, we did not have enough display memory for some of our models. We devised an approximation to save space. An average color is computed from the colors due to each light circuit, and an 8 bit intensity value is computed for each light circuit.

The radiosity process computes an array of color values for each vertex,

$$\langle r,g,b\rangle_k, \text{ with } 0 <= r,g,b < 256,$$

one for each of the $k$ light circuits. We compute an average color,

$$\langle R,G,B\rangle = \Sigma\ (\langle r,g,b\rangle_k);$$

max = MAX(MAX(R,G),B);
<R,G,B> = <R/max,G/max,B/max>;

Then we compute an eight bit intensity value for each of the $k$ light circuits,

$$\langle I \rangle_k = (r_k/R + g_k/G + b_k/B)/3.$$

The storage required is $k$+4 bytes rather than 4*$k$ bytes, assuming word boundary restrictions. The penalty for this savings is that the color of each surface stays constant, regardless of the light circuit settings. Since most of the lights encountered in real models are white, this has not been a problem.

During display, the user may alter global settings for each of the $k$ light groups, i.e,. turn some off, brighten others, etc. We scale the average <R,G,B> value stored at each vertex with the dot product of the global settings and the light group intensity values stored at each vertex. This takes roughly one extra frame time to compute. The result is then stored at the vertex until the user changes the global settings again. Thus, any combination of $k$ lights can be interactively modified during display. In our system, $k$ is 20.

### 4.3 Using a Physically Based Lighting Model on a Non-Physical Models

A physically based rendering method requires physically based models. Although AutoCAD is a powerful modelling tool, it does not guarantee topological consistency of the models it produces. Thus we have developed several programs to help find and fix these model inconsistencies.

- A radiosity program that keeps track of the radiosity for both sides of every polygon and reverses the orientation of polygons as necessary. This requires the modeller to correctly orient only light-emitting polygons.

- A retesselation program to transform polygonal surface tilings into planar subdivisions, a tesselation in which every edge joins two and only two polygons except at surface boundaries. This is necessary to prevent cracks in curved surfaces and shading discontinuities in planar surfaces.

- A program to filter out patches that get no light. This prevents Z-buffer problems caused by coincident co-planar surfaces. It also serves to eliminate portions of the model that ordinarily cannot be seen.

## 5. Adaptive Refinement

Pre-computation is a good strategy and should be applied to viewpoint-independent image features. Unfortunately, only a few tasks, such as visibility relations and diffuse shading in a static environment, fall into this category. To deal with image features that cannot be handled by pre-computation, and features that strain the limits of the display subsystem even with pre-computation, we turn to adaptive refinement.

An object can be approximated at various levels of detail. We use the approximation that most closely fits the needed level of interactivity at the moment. This idea is well-known and regarded as a common-sense notion among flight simulator developers. However, since our projected user, an architect, also constructs the model, we have concentrated on automatic applications of the principle

The radiosity process dices model polygons into patches. In our experience, this increases the number of display polygons by a factor of four to ten. Since our display system, Pixel-Planes 4, takes a constant amount of time to render any color-interpolated quadrilateral, regardless of screen size, a radiosity shaded model takes four to ten times longer to display than the original model. (For commercial graphics workstations, which tend to be pixel-fill limited, this effect may be much less noticable.)

The dicing due to radiosity can be used to produce levels of detail automatically. We have adopted hierarchical polygons as our display primitive (Figure 7).

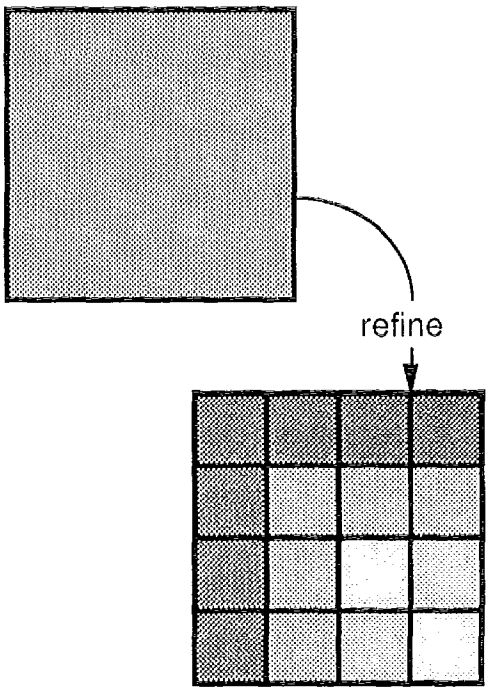



**Figure 7. A Hierarchical Polygon. In the Actual Image the Patch Values are Stored at Vertices and Interpolated to Obtain Smooth Shading.**

Each polygon has an associated list of polygons that can be

used to refine it. When the user stops, the image "sweetens." The resolution level of the hierarchical polygons displayed is increased; we display the patches.

We smooth the transition from one quality level to the next with pixel-level blending to minimize user distraction. The blending takes advantage of the huge aggregate SIMD computing power of the Pixel-Planes 4 machine by computing the blending function at every pixel simultaneously. The blending implementation uses fifty interpolation steps and occurs in a fraction of a second.

The level of resolution refinement is fixed by the choice of patch size made during the radiosity pre-computation. We have developed secondary levels of refinement that are dependent upon the current view and light circuit settings. The secondary levels of improvement are slower since they involve computation during display, but they can markedly improve an image that suffers from coarse patch sampling.

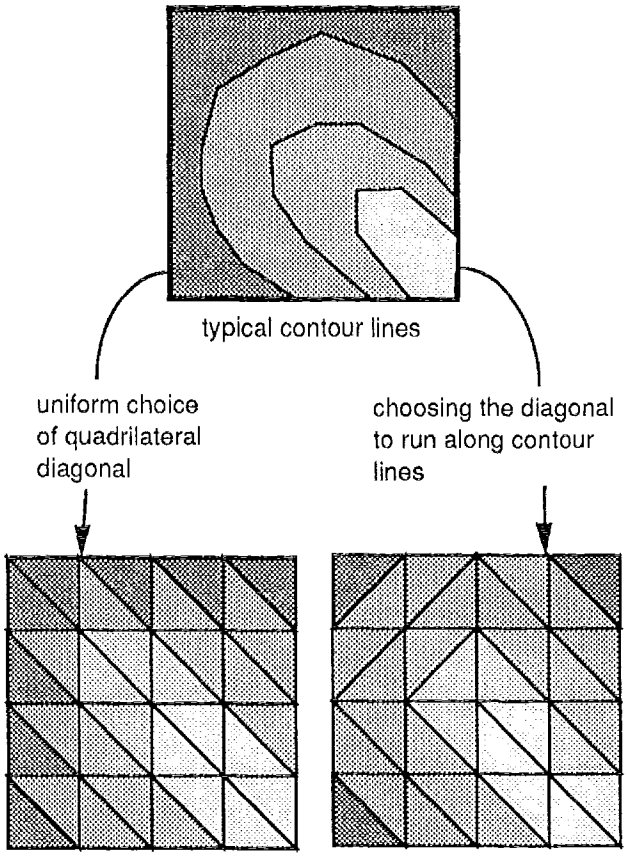



**Figure 8. Uniform Choice of Quadrilateral Diagonal vs. Difference Directed Choice. In the Actual Image, the Colors are Transfered to Patch Vertices and Linear Interpolation Provides Smooth Shading.**

We approximate bilinear interpolation across a quadrilateral patch with two triangles so the shading can be expressed as a Pixel-Planes 4 linear expression [Fuchs 85]. This can cause problems. Note that if the color values at the four corners of the quadrilateral are `a,b,c,d`, then the bilinearly-interpolated color at the center of the patch is `(a+b+c+d)/4`. Since a quadrilateral can be triangulated in two ways, the value at the center is either `(a+c)/2` or `(b+d)/2`, depending upon which diagonal is chosen.

During the first adaptive refinement step, we choose the diagonal uniformly. As a secondary adaptive refinement step, we choose the diagonal that connects the two vertices that are more closely matched in color. This tends to make the diagonals run perpendicular to the shading gradient (Figure 8).

Even after choosing the best diagonal, the approximation may be inaccurate. A patch can be subdivided into four patches. The color value at the new center vertex is computed with bilinear interpolation. The process is applied to each subpatch recursively.

Following adaptive refinement of shading, the image is anti-aliased. We use an algorithm developed by Fuchs et al. that builds the anti-aliased image using supersampling [Fuchs 85]. A new image is computed for each supersample and blended smoothly into an accumulated image using the supersample filter weights.

## 6. Color Plates

The church2 model of Orange United Methodist Church Fellowship Hall has served as one of our primary system evaluation databases. It has 6037 polygons drawn from a larger 12,000+ polygon model. The model subdivision process partitioned it into 16 cells. As a result, the display subsystem needs to process 1887 polygons on average and 3477 in the worst case. Pixel-Planes 4 can display the basic model at more than 8 frames per second.

All five color plates show the image that is produced when the viewer is stationary and adaptive refinement has replaced the coarse model with the radiosity-shaded model and anti-aliasing has smoothed the jaggies. The radiosity process produced 26,794 patches with 65,627 vertices from the original 6037 polygons by dicing at a resolution of 21 square inches. A radiosity solution was computed for 13 different lighting circuits. These may be manipulated interactively. The radiosity solution was computed overnight on a DECstation 3100.

The building was designed by Wesley McClure and Craig Leonard of McClure NBBJ. The modelling was done by Penny Rheingans and John Alspaugh with AutoCAD.

Plate 3. is a perspective plan view. We allow the user to choose whether or not back-facing polygons are rendered. In

this image they are not displayed. Since the only light sources we used were inside the model, the outward-facing roof polygons did not receive any light. We used a filter to remove black polygons from the database to remove any coincident polygons that might exist. This filtered out the outward-facing roof polygons. The inward-facing roof polygons are not displayed because they are back-facing. This has proven to be a whole new way to view the model, as it enables architects to see the entire building in a single view. The Fellowship Hall is equipped with spotlights that can be used to illuminate an individual giving a speech. The spotlight beam can be seen in the lower center of the main hall.

Plates 1 and 2 are views inside the main hall. The spotlight can be seen in Plate 2.

Plates 4 and 5 are of the kitchen. The kitchen can be located in the upper left portion of the perspective plan view in Plate 3.

## 7. References


[Airey 90] Unpublished Ph.D. Dissertation Manuscript, University of North Carolina Department of Computer Science.

[Airey Ouh-young 89] Airey, J. and M. Ouh-young, "Two Adaptive Techniques Let Progressive Radiosity Outperform the Traditional Radiosity Algorithm", University of North Carolina Department of Computer Science Technical Report TR89-020.

[Avis 86] David Avis, Teren Gum, Godfried Toussaint, "Visibility between two edges of a simple polygon", The Visual Computer, 2(6), pp. 342-357

[Bergman 86] Bergman, Larry, Henry Fuchs, Eric Grant, Susan Spach, "Image rendering by Adaptive Refinement", *Computer Graphics* 20(4) pp. 29-38.

[Brooks 86] Walkthrough- A Dynamic Graphics System for Simulating Virtual Buildings, F.P. Brooks, Jr., Proceedings of the 1986 Workshop on Interactive Computer Graphics.

[Cohen 85] Cohen, Michael F., Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg, "A Radiosity Solution for Complex Environments," *Computer Graphics*, 19(3) (Proceedings of SIGGRAPH '85), pp. 31-40.

[Cohen 88] Cohen, Michael F., Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg, "A progressive Refinement Approach to Fast Radiosity Image Generation," *Computer Graphics,* 22(4) (Proceedings of SIGGRAPH '88) pp. 75-84.

[Crow 77] Crow, F.C, "Shadow Algorithms for Computer Graphics", *Computer Graphics,* 19(3) (Proceedings of SIGGRAPH '77), pp. 242-248.

[Fuchs 85] Fuchs, H., J. Goldfeather, J.P. Hultquist, S. Spach, J. Austin, F.P. Brooks, Jr., J. Eyles, and J. Poulton, "Fast Spheres, TExtures, Transparencies, and Image Enhancements in Pixel-Planes", *Computer Graphics*, 19(3) (Proceedings of SIGGRAPH '85), pp. 111-120.

[Goral 84] Goral, Cindy M., Torrance, Kenneth E., Greenberg, Donald P., "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics,* 18(3) (Proceedings of SIGGRAPH '84), pp. 213-222

[Mehlhorn 84] Mehlhorn, Kurt, "Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry", EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1984

[Nishita Nakamae 85] Nishita, T. and Nakamae, E., "Continous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection", *Computer Graphics*, 19(3) (Proceedings of SIGGRAPH '85), pp. 23-30.

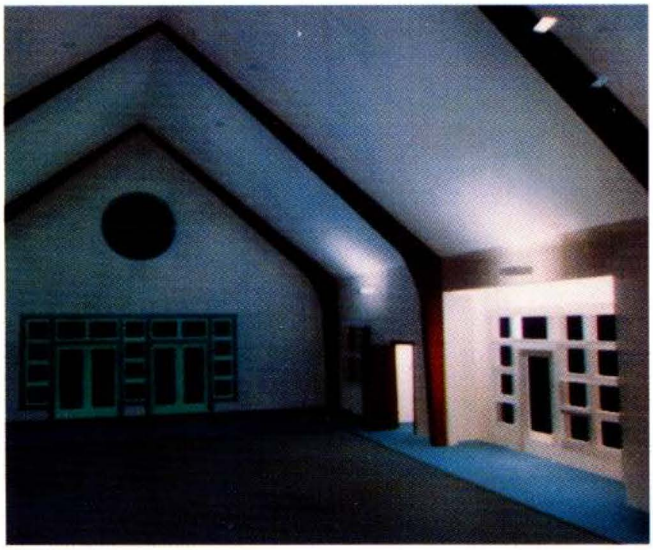
[O'Rourke 87] O'Rourke, Joseph, "Art Gallery Theorems and Algorithms", International Series of Monographs on Computer Science, Oxford University Press 1987

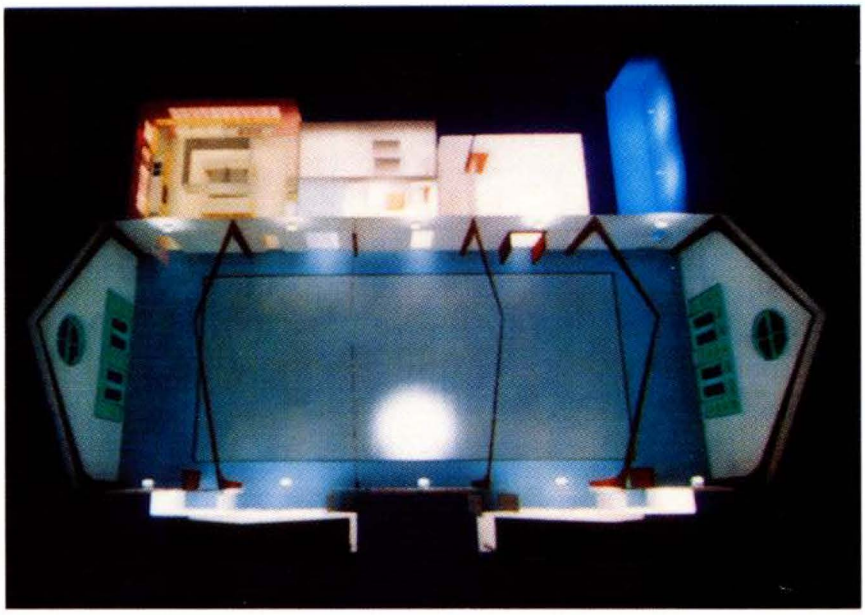
[Ottman 85] Ottman, Thomas, Widmayer, Peter, and Derick Wood, "A Fast Algorithm for the Boolean Masking Problem", Computer Vision, Graphics and Image Processing, 30 pp. 249-268, 1985

[Wallace 89] Wallace, John R., Elmquist, Kells A., Haines, Eric A., "A Ray Tracing Algorithm for Progressive Radiosity", *Computer Graphics* 23(4) (Proceedings of SIGGRAPH '89), pp.315-324.

[Weiler 81] Weiler, Kevin, "Polygon Comparison using a Graph Representation", *Computer Graphics* 15(4), (Proceedings of SIGGRAPH 81), pp. 10-18


Color images for this paper can be found in the color plate section.





1. 2.

3.

4. 5.

Airey, Rohlf and Brooks, "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments".