PARALLEL PROGRESSIVE REFINEMENT AND PROJECTION BASED DISCONTINUITY MESHING FOR RADIOSITY

BY

JON GOLDMAN
B.S., Electrical Engineering, University of Rochester, 1984

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science Electrical Engineering and Computer Science
in the Graduate College of the
University of Illinois at Chicago, August 1995

Chicago, Illinois

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

LIST OF FIGURES (Continued)

FIGURE **PAGE**

ACKNOWLEDGMENT

Computer graphics is a creative and exciting field which has enabled me to investigate complex forms of visual communication. I would like to especially acknowledge my sweetheart and wife, Anne Marie Rossi. Without her undying support and devotion this project would have not come about. When my life and my work has been in disarray, when hardware was fluky and software buggy, when I felt blue and out of sorts, Anne Marie was there for me. She provided a constant source of creative problem solving and I am indebted to her many times over for keeping me on track and focused on the problem at hand.

Special EVL Thanks,

To Tom DeFanti, my advisor and co-director of the Electronic Visualization Laboratory. Tom provided an environment rich in hardware and software resources. This allowed myself and others to engage in computational visualization at the highest levels;

To Robert Grzeszczuk for participating in my thesis committee and taking serious this responsibility, as well as providing many ideas on how to make the meshing problem more tractable;

To Craig Barnes for helping develop the VROOM/CAVE codes. Also for his useful insight and input into some of the more esoteric Unix capabilities and rendering issues, which has enabled me to take radiosity to ever higher planes of success;

To Milana Huang for being a good friend, listening, suggesting ways to break through road blocks, and commiserating on the Masters candidate struggle;

To Randy Hudson for illuminating discussions on projective geometry and related issues;

ACKNOWLEDGMENT (Continued)

To Maxine Brown for her undying support of students at EVL and useful advice she imparted to me over the years spent here;

To the rest of EVL and staff who helped or provided moral support – you know who you are and whence you came;

And finally, to the Microfabrication Laboratory and David Naylor at the University of Illinois at Chicago. This work was funded under a fellowship through David's laboratory from the Department of Education.

Abstract

This dissertation presents algorithms and techniques that are applicable to the radiosity method for computer graphics. First, radiosity and relevant background information is introduced. The reconstruction problems with uniform meshing and their solutions are discussed. This is followed by a description of an implementation of a parallel progressive refinement algorithm and its use in the CAVE Automatic Virtual Environment (CAVE). A new approach to the meshing problem is presented with a parallel planar projection discontinuity meshing algorithm. This algorithm uses perspective projection, hardware assisted visibility determination, and constrained Delaunay triangulation to solve problems with uniform meshing. The algorithm makes use of available hardware resources to *a priori* determine shadow boundaries that are essential to computing the discontinuity mesh. This method can be implemented with a minimum of commonly available graphics hardware and capabilities, namely Z-buffering and orthographic and perspective projections. Significant speedups can be obtained with systems that support stencil masks or Z-buffer through hardware. Finally, results from the parallel renderer in conjunction with the mesher are presented along with future items of research in this area.

CHAPTER 1

INTRODUCTION

1.1 Physically Based Lighting Models

Computer graphics rendering involves many tradeoffs between implementation and the perceived realism in the final image. Early computer graphics systems were limited by hardware and software algorithms and strove to give a flavor of realism in the scene rather than an exact representation. As hardware improved and new software techniques were developed it was realized that it would be possible to generate quite accurate imagery, perhaps with such realism that the image would be indistinguishable from a photograph of the real thing. Photorealism became a clarion for rendering. In order to achieve this goal the physics of light behavior in an environment needed to be understood. This led to the separation between local and global illumination models. Local illumination models are concerned with the shading of a surface at a point independent of the interaction between neighboring surfaces. Global illumination models shade a scene by taking into account the interaction of light between each object (surface) and every other object (surface) in the scene. Global illumination models are considered physically based lighting models because they account for the propagation of these effects throughout the environment. The first such application of a global illumination model is the now-famous ray tracing method developed by Whitted [31].

Early on in computer graphics it was observed that the contributions to the perceived illumination of a surface were due to different types of light-reflecting effects. Some of the light reflecting from a surface was specular in nature, and some of the reflected light was matte, or diffuse. This classification of reflection into two categories does not completely describe all aspects of light reflection, but does account for the predominate features of light behavior that govern how we perceive typical objects in everyday life.

The two major physically based lighting models that arose out of the study of specular and diffuse light reflection are ray tracing and radiosity. Classic ray tracing concentrates on handling the specular nature of light transport using the physics of optics. Classical radiosity seeks to model the interreflection of the diffuse component of light using an energy balance equation and considering all surfaces as ideal Lambertian reflectors.

This dissertation presents two areas of work in the radiosity method. These are one, a parallel implementation of a commonly used algorithm which solves the radiosity matrix. This matrix and the parallel implementation will be discussed in greater detail in chapter 2 and in chapter 3. The other and main thrust of original research presented in this thesis is the development of a new meshing technique to identify shadow boundaries. This process is often referred to as discontinuity meshing and the technique developed here is called, parallel planar projective discontinuity meshing (quite a mouthful!). Meshing is the process of slicing and dicing the polygons that comprise the geometry of a scene in order to solve a discrete formulation of radiosity (see section 2.2). Meshing and discontinuity meshing will be discussed in chapter 2 and the parallel planar projection method will be detailed in chapter 4.

CHAPTER 2

BACKGROUND

Much has been written about the radiosity method of lighting for computer generated scenes and objects. Good basic references are [4], [24], and [10]. The following sections will present some of the material found in these references as well as newer material as it pertains to the problems at hand. Sections 2.1, 2.2, and 2.3 will introduce the mathematical formulation for the radiosity method and provide background material on the various aspects of radiosity for computer graphics. Section 2.4 will introduce meshing which is the area in which this thesis presents original research (see chapter 4).

2.1 The Radiosity Formulation

Radiosity has its roots in the field of radiative heat transfer [20]. Its use for computer graphics dates to 1984 and 1985 when researchers at Cornell University and at Fukuyama and Hiroshima universities, in the United States and Japan, used it to model the interaction of the diffuse component of light in a closed environment, respectively [11] [18]. As the method evolved, it was unified with the particle transport theory, finite element methods, and the rendering equation [15].

Although the original derivation of radiosity for computer graphics was not done so, the radiosity formulation can be derived from an energy balance equation [24]:

$$\underbrace{L(x,\theta_0,\phi_0)}_{\text{total radiance}} = \underbrace{L_e(x,\theta_0,\phi_0)}_{\text{emitted radiance}} + \underbrace{\int_\Omega \rho_{bd}(x,\theta_0,\phi_0,\theta,\phi) L_i(x,\theta,\phi)\cos\theta\, d\omega}_{\text{reflected radiance}} \qquad (2.1)$$

Where,

- $L(x,\theta_0,\phi_0)$, is the radiance leaving point x in direction (θ_0,ϕ_0);
- $L_e(x,\theta_0,\phi_0)$, is the emitted radiance from point x and is a property of the surface;
- $L_i(x,\theta,\phi)$, is the incident radiance impinging on point x from direction θ,ϕ;
- Ω is the set of directions $(\theta,\ \phi)$ in the hemisphere covering the surface at point x;
- $\rho_{bd}(x,\theta_0,\phi_0,\theta,\phi)$; is the bi-directional reflectance distribution function (BRDF) describing the reflective properties of the surface at point x;

Equation 2.1 is a form of the rendering equation.

Classical radiosity makes additional assumptions [10]:

1. All surfaces are opaque.
2. Surfaces are small.
3. The radiosity (energy arriving) across a surface is constant.
4. The irradiance (energy departing from) across a surface is constant.

To convert to the standard form, we can use the relation, $B = L\pi$. Additionally a visibility term is introduced in order to allow the integration to occur over the entire scene, but taking into account only those surfaces visible from point x:

$$V(x,y) = \begin{cases} 1 & : \text{ if x and y are mutually visible} \\ 0 & : \text{ otherwise} \end{cases} \tag{2.2}$$

Equation 2.1 can be simplified by moving the reflectance term outside the integral (since the radiosity method assumes diffuse energy transfers only [11] [24]).

These simplifications yield the following:

$$B(x) = E(x) + \rho_d(x) \int_{y \in S} B(y) \frac{\cos\theta \, \cos\theta'}{\pi r^2} V(x,y)\, dy \tag{2.3}$$

2.2 Discretization

Discretizing is the process of breaking up a problem into N discrete pieces and is necessary in order for the radiosity solution to be computationally tractable.

The integral in Equation 2.3 is a double integral over all the surfaces, S, in the environment. If the environment is broken into N sub-surfaces, or patches, see Figure 1, we have the following

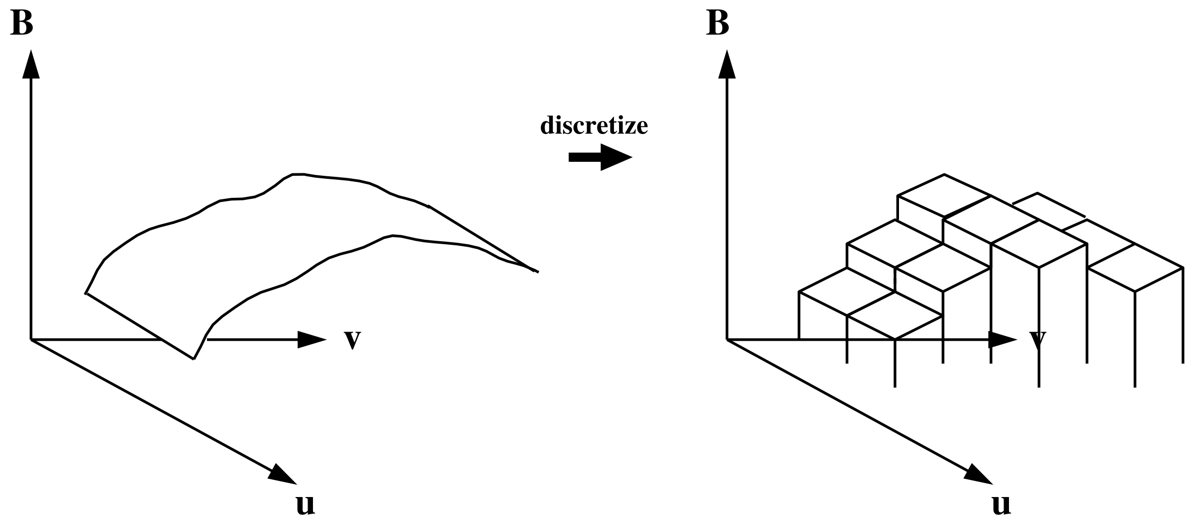

Figure 1. Discretization of radiance across a surface

formulation:

$$B(x) = E(x) + \rho_d(x) \sum_{j=1}^{N} \int_{y \in P_j} B(y) \frac{\cos\theta \, \cos\theta'}{\pi r^2} V(x,y)\, dy \tag{2.4}$$

Another simplifying assumption is that the radiosity, $B(y)$, is constant over each patch. Although this is not a requirement, it does serve to simplify the calculations. In practice both (constant) zero order and higher order basis functions representing the radiosity of a patch have been used. Higher order techniques such as the Galerkin method are an ongoing area of research in radiosity. This thesis will not cover higher order methods further, inviting the curious reader to peruse the literature for details [32].

For classical radiosity, the reflectance, ρ_d, is considered to be constant across a patch's surface. Likewise the emittance, E_i, is considered to be uniform.

These assumptions result in the following equation:

$$B_i = E_i + \rho_i \sum_{j=1}^{N} B_j \frac{1}{A_i} \int_{x \in P_i} \int_{y \in P_j} \frac{\cos\theta \ \cos\theta'}{\pi r^2} V(x, y) \, dy dx \tag{2.5}$$

Or in a more concise format:

$$B_i = E_i + \rho_i \sum_{j=1}^{N} F_{ij} B_j, \tag{2.6}$$

where,

$$F_{ij} = \frac{1}{A_i} \int_{x \in P_i} \int_{y \in P_j} \frac{\cos\theta \ \cos\theta'}{\pi r^2} V(x, y) \, dy dx \tag{2.7}$$

In matrix form:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ . \\ . \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ E_n \end{bmatrix} \tag{2.8}$$

2.3 Form Factor Computation

Solutions to the integral in Equation 2.7 are varied. Analytic solutions exist and a closed form solution to arbitrarily oriented polygons can be found in [21]. In practice most form factor computation is done numerically using one of two methods: Hemicube, or Ray Tracing.

2.3.1 Hemicube Form Factors

In the hemicube method the form factor between patch i and patch j is computed using available display hardware. Using the Nusselt analog [25] and substituting a half cube for a hemisphere, the form factor is computed by projecting the scene onto the hemicube faces. Each face is divided up into small elements, called, "pixels" (an unfortunate use of the term since it can easily be confused with the use of pixel for "picture element" on a monitor). Each element is assigned a delta form factor value:

$$\Delta F_{dA_i} = \frac{1}{\pi\sqrt{x_i^2 + z_i^2 + 1}} \Delta A_i \tag{2.9}$$

or,

$$\Delta F_{dA_j} = \frac{1}{\pi\sqrt{y_j^2 + z_j^2 + 1}} \Delta A_j \tag{2.10}$$

The viewpoint is placed at the center of patch i and the rest of the scene is drawn from the perspective of the viewpoint. The polygons are identified with a special color. The elements are scanned via the Z- or frame buffer. Form factors are summed by adding the corresponding delta form factor for that element.

2.3.2 Ray Tracing Form Factors

Since the form factor calculation is essentially a visibility calculation, ray tracing methods lend nicely to this part of a radiosity computation. Form factors are determined by casting a set of rays from the source patch, i, to the destination patch, j, and computing the percentage of rays that hit the destination patch. One advantage to using ray tracing over the hemicube method is a reduction in aliasing. Hemicube suffers from the possibility of missed objects which fall between intra-pixel spacing. In hemicube, these elements will not be sampled. Ray tracing alleviates this problem by casting sample rays to all visible elements.

The following equation is used to compute form factors from a differential area to a finite area [29].

$$F_{ij} = \frac{1}{n} \sum_{k=1}^{N} \delta_k \frac{\cos\theta_{ik} \cos\theta_{jk}}{\pi r_k^2 + A_j/n} \tag{2.11}$$

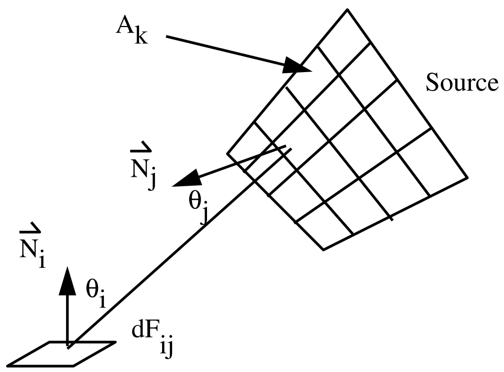

Figure 2. Form factor calculation using ray tracing

where k is the kth sample point on the source j (see Figure 2).

The differential area is defined by a vertex on the receiver, and the finite area is represented by the current source.

2.4 Meshing

Meshing is one of the most important steps in computing an accurate and realistic radiosity simulation. Determining accurate form factors is another important issue in radiosity image synthesis and has been studied extensively [21] [23] [4] [24].

Meshing involves the subdivision of the original input surface "patches" into smaller "elements". Smaller mesh elements contribute to a better representation of the radiosity by more accurately describing the gradient across the surface. Patches (mesh elements) that are constructed large compared to the scale of illumination changes will blur the image since the computed radiosity will be averaged over a larger area. This is depicted in Figure 3.

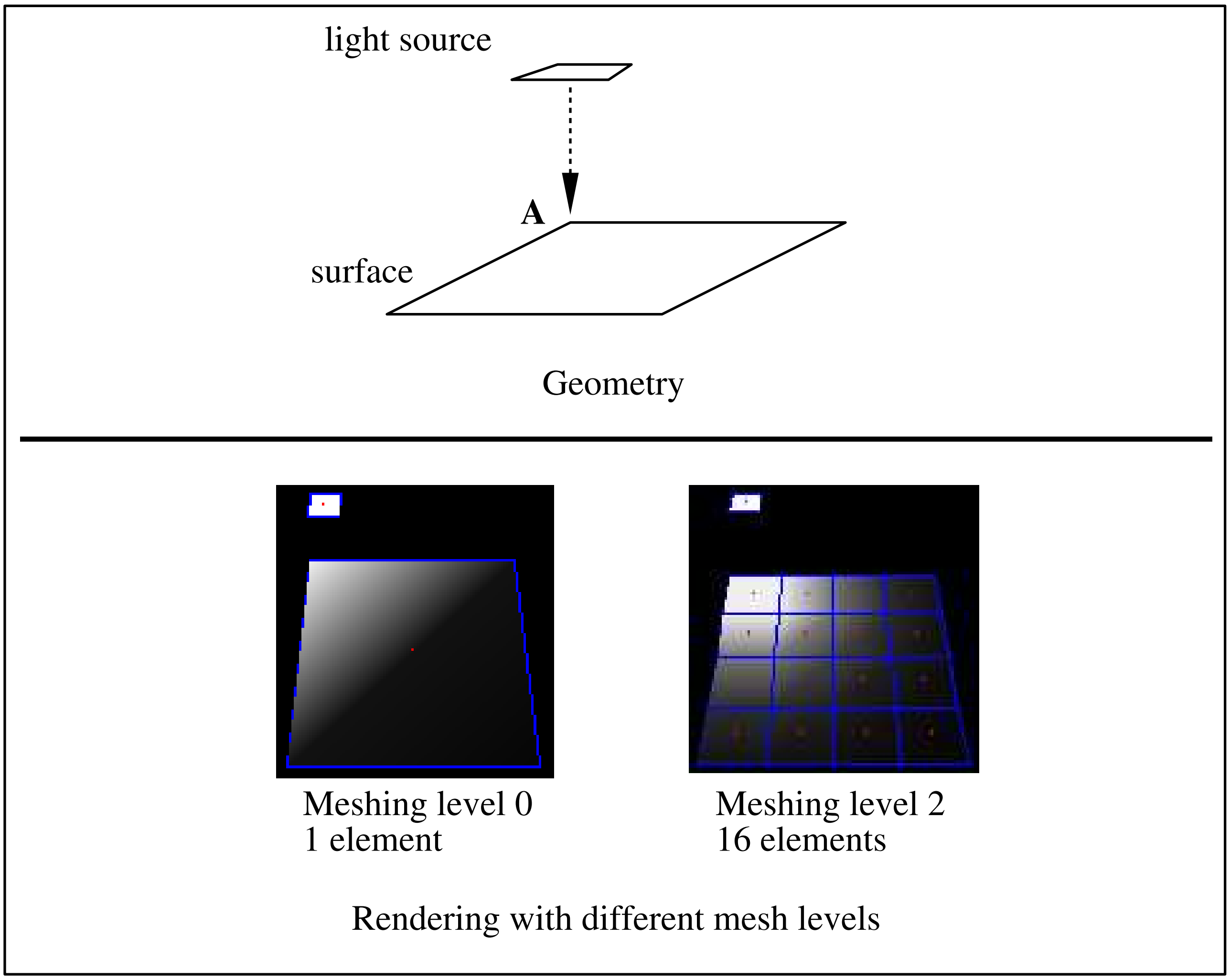

Figure 3. Radiosity gradient across mesh elements of different resolution

2.4.1 Uniform Meshing

Figure 3 shows that as the mesh size decreases, the radiance is represented more accurately across the surface. In this instance a simple, uniform mesh was chosen.

2.4.1.1 Multiblocking

Multiblocking is a type of uniform meshing that seeks to break up complex or concave geometry into a uniform mesh using a recursive subdivision scheme [5].

2.4.2 Deficiencies of Uniform Meshing

The illumination is better represented across a surface with a uniform mesh but there are problems associated with using one. The number of interactions between elements will increase on the order of N^2, with N representing the number of input patches. Although easier to implement, naive, uniform meshing can impose serious computational limitations.

Another problem with the uniform meshing approach is that it can lead to serious visual artifacts. One example is aliasing which is due to the regular and coarse nature of uniform meshing. An example of this can be seen in Figure 4.

Mach banding is another problem that occurs with a uniform mesh. Linear interpolation introduces D^1 discontinuities at the element boundaries. Heckbert uses the notation that a function has a D^k discontinuity at a particular point if the function is C^{k-1} continuous but not C^k continuous [14]. The human visual system is very sensitive to these kinds of discontinuities. They typically show up as discernable light bands in the rendered image.

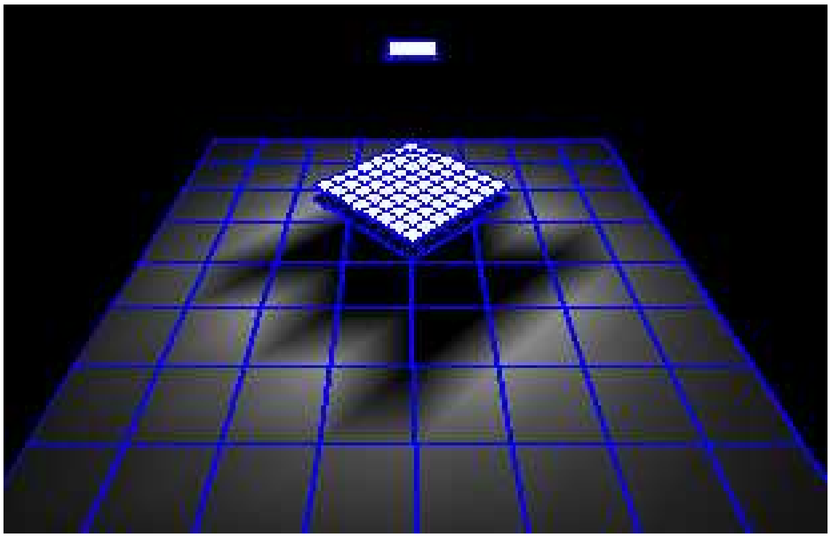

Figure 4. Uniform grid exhibiting severe aliasing artifacts

A finer mesh can alleviate some of the problems of aliasing, but eventually the N^2 patch factor dominates the computation times. What is needed are more sophisticated meshing strategies.

2.4.3 Non-Uniform Meshing Strategies

This thesis addresses this issue directly with a new method of discontinuity meshing using a hardware-assisted shadow projection technique. Before we discuss this method in detail it would serve us well to review other meshing strategies. Many researchers have tackled this problem and looking at these techniques should give some insight into the validity and usefulness of the method developed here.

2.4.3.1 Adaptive Subdivision

Adaptive meshing is an *a posteriori* method by which the mesh is re-evaluated during each iteration of the radiosity solution. The computation begins by computing the radiances of the initial uniform mesh. As each iteration of the solver completes a pass, the current computed radiance values at the node points are sampled and the mesh is adjusted if the computed radiosity of a neighbor differs by some user-preset tolerance. The first example of this method used for computer graphics can be found in [3].

2.4.3.2 Decomposition Methods

Decomposition methods strive to create a mesh from a collection of node points mapped to the geometry. They are better suited to complicated geometry and allow the mesh to be more easily manipulated during mesh refinement. There are a number of schemes described in [6]. Delaunay triangulation and BSP trees are often used in employing these methods. In fact, parallel planar projection, the technique introduced in chapter 1 and described in detail in chapter 4 is a decomposition method.

2.4.3.2.1 Delaunay Triangulation

Delaunay triangulations are often used in computational geometry and computer vision applications as well as, more recently, radiosity[1]. Delaunay triangulation is defined as follows:

---

[1] One property which is of particular consequence to computer graphics is the fact that a Delaunay triangulation minimizes the smallest angle of its triangles over all possible triangulations in the set of

Definition 2.1 *If P is a set of points on the plane, then a triangulation, T, of P is a Delaunay triangulation if for each edge in T there exists a circle, O, with the following properties:*

1: *The endpoints of each edge in T lie on the boundary of O;*

2: *No other vertex in P lies in the interior of O.*

2.4.3.3 Hierarchical Radiosity

Hierarchical radiosity is a technique that structures a mesh into a hierarchy of interacting elements [13]. At each level in the hierarchy, elements can interact with elements at any other level. The metric that determines which elements interact with each other is based upon the form factor between them. Hierarchical methods are based upon recent advances in N-body simulations where the force from a cluster of objects at a given distance can be approximated by treating the cluster as one continuous object. In radiosity, the distance term is manifest in the form factor calculation.

2.4.3.4 Discontinuity Meshing

Contrary to adaptive meshing, discontinuity meshing is an *a priori* method that attempts to identify appropriate mesh boundaries based upon the geometrical configuration of a scene.

---

points, P, that comprise the triangulation. This has the effect of reducing long, thin triangles as much as possible. This is important when triangulations are rendered with Gouraud shading. Long, thin sliver triangles interspersed with wide, short triangles will exhibit shading artifacts when using linear interpolation.

Discontinuity meshing alleviates many of the sampling problems associated with other forms of meshing by constructing a mesh that falls along the discontinuity boundaries of a particular target surface. If the discontinuity boundaries can be accurately determined, then the radiance function on either side of the boundary can be reconstructed exactly [24].

Recall from Equation 2.3 that the radiosity contains a visibility term, $V(x, y)$. When solving the discrete form of this equation, patches are classified into one of three categories of visibility with respect to the current emitter. A patch is either

1: Completely visible

2: Completely invisible

3: Partially visible

Partially visible patches are problematic. From Equation 2.2 the visibility term evaluates to either 0 or 1. For patches that are invisible, Equation 2.3 reduces to,

$$B(x) = E(x) + \rho_d(x) \sum_{j=1}^{N} \int_{y \in P_j} B(y) \frac{\cos\theta \, \cos\theta'}{\pi r^2} \, dy$$

Discontinuity meshing reduces the error associated with partially visible patches. If the shadow, or discontinuity boundary, is fully determined, then patches falling on one side of the boundary will have $V(x, y) = 1$ and patches falling on the other side will have $V(x, y) = 0$. In uniform meshing a large number of patches will likely be partially visible to an emitter patch, unless the element mesh factor is set very high. Increasing the mesh density in a uniform mesh suffers from $O(n^2)$ complexity issues as described in 2.4.2 and should be avoided if possible.

2.4.3.5 Backprojection

Another shadow boundary technique uses a *backprojection* data structure [8] [27]. This method locates mesh boundaries with the following requirement: Each mesh face has a view of the light source which is topologically equivalent. Backprojection can be used to compute the complete discontinuity mesh and is therefore important for catching subtle visibility changes called *visual events.* The types of visual events classified are VE (Vertex-Edge) and EEE (Edge-Edge-Edge) interactions [33].

Backprojection is similar to another visibility technique which determines the antiumbra and antipenumbra of a convex area light source shining through holes using a mathematical formalization called Plücker Coordinates, a five-dimensional line representation [28]. The antiumbra is the volume from which all points on the light source can be seen. The antipenumbra is the volume from which some, but not all, of the light source can be seen.

CHAPTER 3

PARALLEL PROGRESSIVE REFINEMENT

3.1 Introduction

Progressive Refinement [2] has become one of the standard techniques for implementing a usable radiosity system. It is based upon proven techniques in linear algebra and is an indispensable tool in the arsenal of the rendering writer. It allows reasonable update times as the radiosity solution emerges and provides valuable feedback early on as to whether lighting, geometry, or basic assumptions about the underlying physical model are correct.

With this in mind and the desire to create renderings at real-time frame update rates, a parallel version of progressive refinement was implemented on a shared memory multi-processor machine. This chapter provides the theory behind progressive refinement and describes the system that implements a parallel version of the progressive refinement algorithm.

The rendering system developed in this chapter was used to demonstrate the capabilities of parallel rendering for the VROOM exhibit at the ACM's Special Interest Group GRAPHics (SIGGRAPH) conference in July, 1994. The renderer was showcased in the CAVE Automatic Virtual Environment, an immersive virtual reality theatre developed at the Electronic Visualization Laboratory at the University of Illinois at Chicago.

In the context of this thesis the parallel implementaion of the progressive refinement algorithm is described. In chapter 4 it is used for rendering images of scenes meshed via application of the meshing algorithm developed there.

3.2 Progressive Refinement Theory

Progressive refinement concerns the graceful update of an image during scene rendering. For radiosity it is one of the most useful tools for providing interactive update rates during image computation. Progressive radiosity was first introduced in 1988 [2]. Ray tracing of form factors was incorporated into the model in 1989 [29].

In the original radiosity method, the set of linear equations from Equation 2.8 are solved using a Gauss-Seidel iteration method. This technique returns updated radiosity values on a per patch basis by "gathering" the incoming energy across the surfaces. Gauss-Siedel has been shown to converge to a final solution asymptotically faster than other methods due to the diagonally dominant nature of the radiosity matrix [26].

From Equation 2.6 and Equation 2.8 one can see that a single term of these equations is evaluated as such:

B_i due to $B_j = \rho_i B_j F_{ij}$

This can be interpreted as a gathering for patch B_i of the radiant energy from all other patches in the scene.

```
For each patch i
        Set unshot radiosity B_i to emittance E_i
while (not converged)
        For each patch i
                Select patch B_i with highest unshot radiosity
                For each patch j
                        Calculate form factor F_ji
                        new radiosity B_j = ρ_j F_ji unshot B_i
                        unshot radiosity B_j = unshot radiosity B_j + new radiosity B_j
                        radiosity B_j = radiosity B_j + new radiosity B_j
                Set unshot radiosity B_i to zero
```

Figure 5. Pseudocode for progressive refinement

In progressive radiosity, the energy distribution is more analogous to “shooting,” where the patch radiosities in the entire scene are updated by sending out the unshot radiosity from the current patch. This is the reverse of the gathering process and results in computation of the radiosity received by all patches, B_j, as a result of shooting from patch B_i:

B_j due to $B_i = \rho_j B_i F_{ij} A_i / A_j$

Progressive radiosity provides useful image displays much earlier on in the solution process. This enables interactive image update rates in a smoother and more graceful manner than the original method. Pseudocode for progressive refinement is shown in Figure 5.

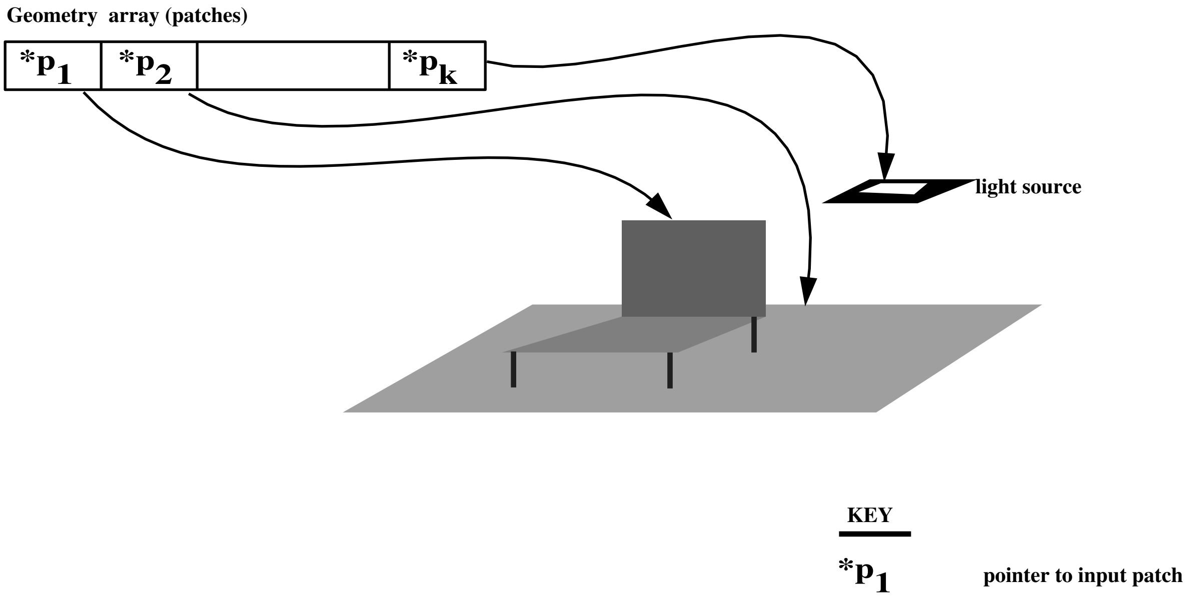

Figure 6. Initial input geometry pointed to by patch array

3.3 Implementation

The input geometry is initially composed of patches representing planar convex polygons. Two types of primitive are accepted – triangles and quadrilaterals. See Figure 6.

The system can accept pre-meshed input polygons as shown in Figure 6 and then perform meshing. Alternatively pre-meshed geometry can be read in. As patches are read into the system, they are meshed into a quadtree data structure. The leaves of the quadtree represent the polygonal elements that are inserted into a heap array and ultimately rendered to the screen. After quad-meshing, several arrays are set up to maintain order and aid in parallelization during the rendering stage.

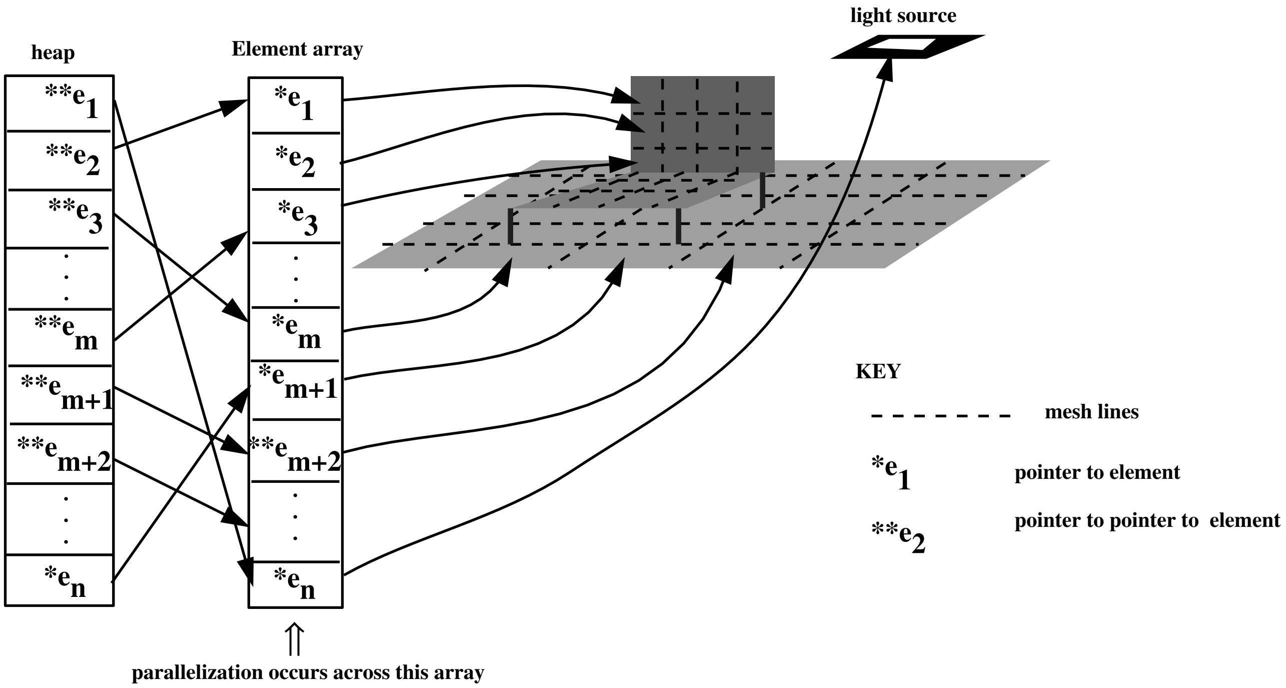

Figure 7. Heap and Element arrays for input geometry

There are two levels deep of arrays of *Element* pointers. An Element is a node in the quadtree hierarchy. A binary heap is maintained whose contents are doubly indirect pointers to leaf nodes in the hierarchy. Sorting is performed on the heap array. See figure Figure 7.

We use a binary heap sort [7] for the following reason. The total time to extract shooting elements (the ones with the greatest unshot radiosity) is $O(n + m\,lgn)$, where n is the input data set size, and m is the total number of iterations. This is better than $O(m\,n\,lgn)$ which would result in using another sorting method. This is possible because the heapify stage of a heap sort algorithm takes $O(lgn)$ and the build time $O(n)$. The build time is only required once at the beginning, and the $O(m\,lgn)$ results from executing the heapify stage for m iterations. The final heapsort stage, which takes $O(n\,lgn)$, is not needed during the iteration stage since the top of the heap (the first element of the array) contains a pointer to the element we are looking for: the one with the largest unshot radiosity (this is patch B_i as shown in psuedocode in Figure 5.

Note in Figure 7 that a uniform meshing scheme is depicted. This shows the Element array sequentially accesses the various elements associated with a parent patch.

Parallelization is implemented across the geometry array during the shooting stage of the progressive refinement solution.

The pseudocode for the parallel version looks similar to Figure 5, except that after each shooting patch, i, is determined, the shooting is distributed across the parallel processors (see Figure 8).

The heap is used to sort the elements by unshot radiosity value and the largest one is extracted. The geometry array is used to process the receivers and is distributed among the number of processors available. This is depicted in Figure 9.

In order to avoid memory contention when calculating the radiosity of an element, receivers are not (necessarily) uniformly distributed across the processors available. This is necessary because patches are meshed via a quadtree data structure with the outer vertices at each mesh level being shared (Figure 10).

For each patch i
    Set unshot radiosity B_i *to emittance* E_i
while (not converged)
    For each patch i
        Select patch B_i *with highest unshot radiosity*
        Partition the input patches n *over the processors* p *such that*
        each processor shoots to approximately n/p *patches*
        Parallel section
        For each patch j
            Calculate form factor F_{ji}
            new radiosity $B_j = \rho_j F_{ji}$ *unshot* B_i
            unshot radiosity B_j = *unshot radiosity* B_j + *new radiosity* B_j
            radiosity B_j = *radiosity* B_j + *new radiosity* B_j
        Set unshot radiosity B_i *to zero*

Figure 8. Pseudocode for parallel progressive refinement

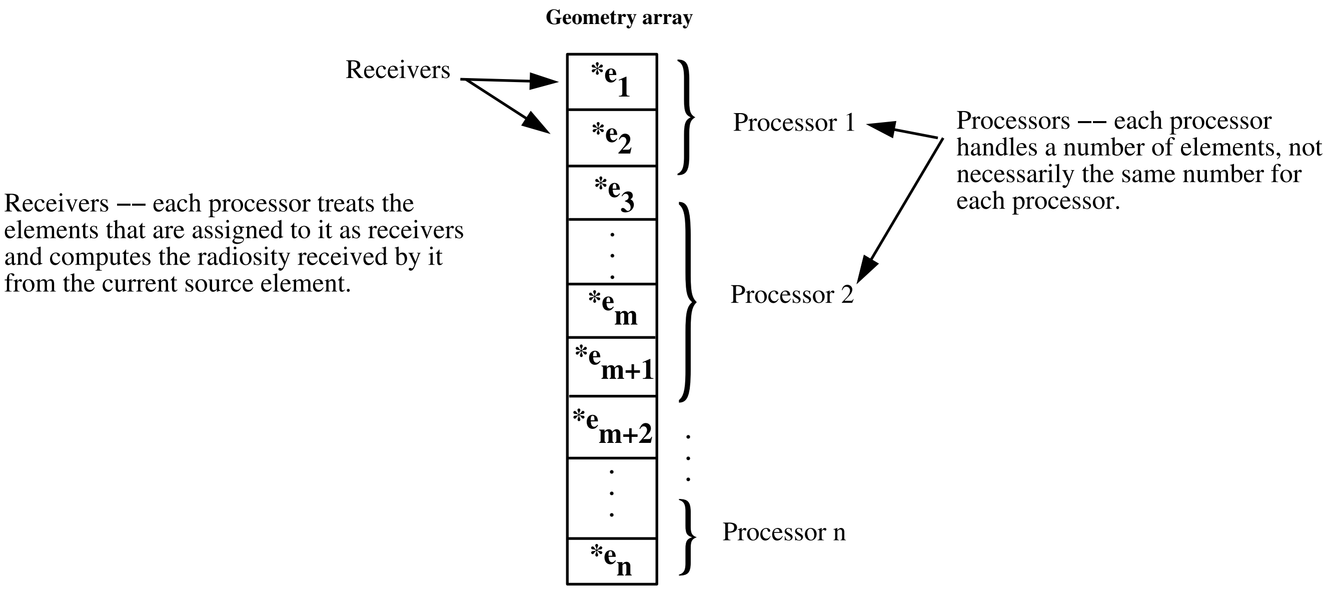

Figure 9. Parallel distribution of geometry elements across processors

The elements are distributed among processors using two parameters: 1) the number of processors available in the system, and 2) whether the element's root patch is the same as the last element on the processor before it. During an initialization stage of the parallel computation, elements are assigned to a particular processor by ranging across the Element array. This works because elements are stored in the Element array in the order their parent patch was meshed. All that is needed to ensure that neighboring receiver elements do not fall on different processors is to find parent patch transitions via a root pointer, which every element has. The root pointer points to the top of the quad tree for every patch, i.e. the topmost parent patch (see Figure 11).

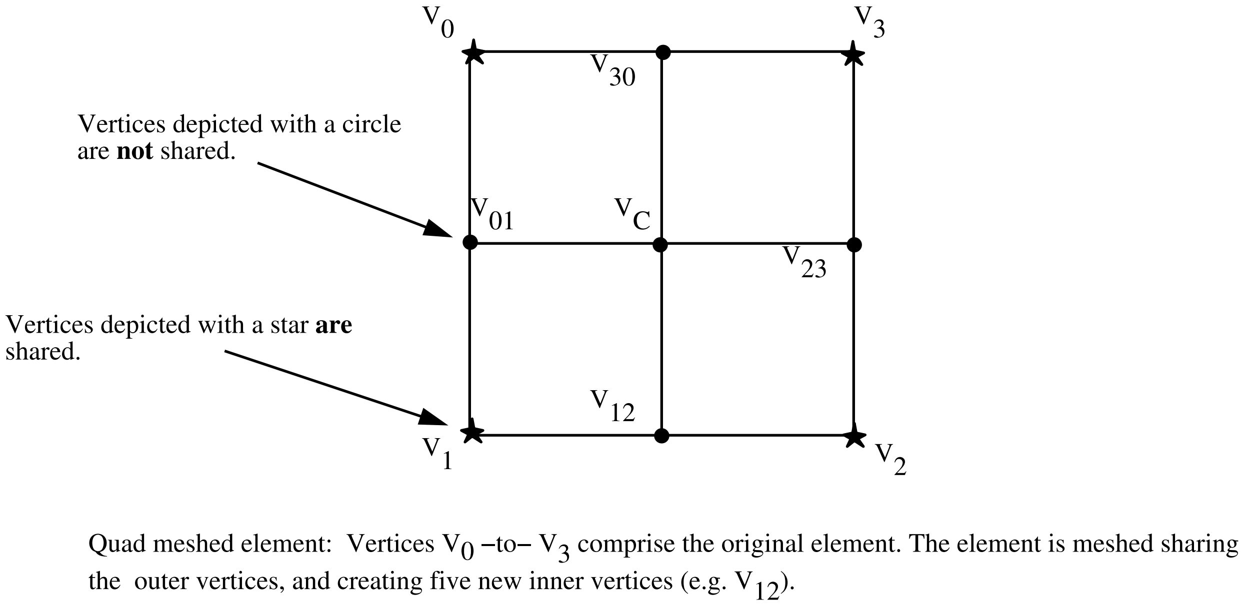

Figure 10. Quad meshing

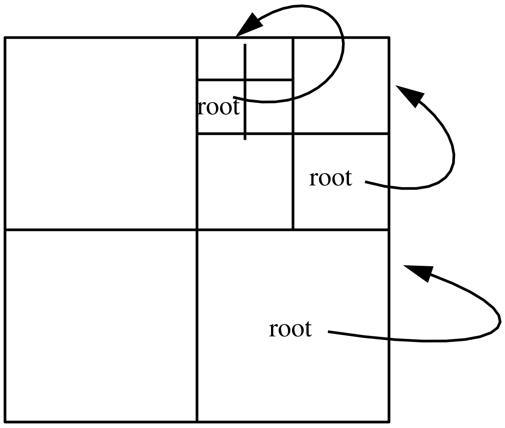

Figure 11. Quad tree mesh showing root pointer pointing to topmost parent

CHAPTER 4

DISCONTINUITY MESHING USING PARALLEL PLANAR PROJECTION

The meshing algorithm introduced in this chapter presents a new method for computing discontinuity (shadow) boundaries on polygonal elements. The aim behind this method is to use currently available graphics hardware to assist in determining the discontinuity boundaries. This should accomplish two goals. One, to reduce the size of the mesh that would otherwise be required to produce a smooth image with uniform meshing, and two, to produce a more accurate solution during rendering.

Section 4.1 presents an overview of the algorithm. This is followed by a detailed description of the algorithm and its workings in sections 4.2 through 4.2.4. The complexity is analyzed in section 4.2.5.

4.1 Overview

The process proceeds as follows: A three-dimensional model is read into a database. The database is then meshed via an elaborate preprocessing stage. This stage consists of drawing the scene from every light source's viewpoint and taking every potential receiving polygon into consideration as a *Target*. Target polygons are those elements which do not emit energy (i.e. non-light sources). Meshing information for a target polygon is extracted from the projection of all other polygons in the scene onto the target. This is accomplished by noting that the two-dimensional perspective projection of one polygon onto another specifies the discontinuity boundary formed by two regions separated by shadow. The application of this rule for all the polygons in the scene results in the discontinuity mesh.

4.2 Theory and Description

Parallel planar projection meshing is an *a priori* method and can be considered an image space algorithm. It is invoked prior to the parallel rendering system described in 3.3. The mesh is generated using parallel plane projection techniques. The basic idea is this: every polygon in the scene is considered as a potential target. All other polygons which may or may not occlude the target are considered potential "blockers." Identify the mesh by projecting all blockers onto each target in turn, and mesh the target. The projection is done from the perspective of the light sources. Light sources are used as the projection point because they are the primary contributors to high radiance gradients on receiving surfaces, i.e. shadows and discontinuity boundaries are due mostly to visual events on the surface of a target due to the light sources.

Pseudocode of the procedure is shown in Figure 12 with each stage elaborated upon in following sections of this chapter.

4.2.1 Geometry Identification

In this stage the geometry is read in and light sources are identified. As stated earlier, polygonal objects not considered light sources are identified as possible targets.

```
Stage 1: Geometry Identification (section 4.2.1)
Read a databaseand triangulate
Identify light sources
For each light source i do
    Set viewpoint at source i
    for each potential target polygon j in scene do
        Stage 2: Polygon Culling (section 4.2.2)
        Render target into stencil buffer
        Render scene minus source i with special vis-ID color
        Scan frame buffer and tag visible triangles via vis-ID color
        If target polygon j vis-ID indicates visibility do
            for each potential blocker polygon k in scene do
                for each vertex V_s of light source do
                    Stage 3: Projection/Edge Insertion (section 4.2.3)
                    if blocker k visible, project onto target j
                        and generate nodal points and 2D edge segments
        Stage 4: Triangulation (section 4.2.4)
        Mesh target polygon j via constrained Delaunay triangulation
        Back project triangulation to 3D object space
```

Figure 12. Pseudocode for parallel planar discontinuity meshing algorithm

4.2.2 Polygon Culling

The ultimate mesh can be reduced when only visible objects are considered for meshing. There are two visibility classifications in this algorithm. One is with respect to the light source. The other is with respect to the target. If a target is not directly visible from the light source, it can be ignored with regards to this meshing operation. If the blocker is completely invisible with respect to the target, it will add no direct discontinuity boundaries, and can thus be culled from the list of projectors.

A visibility pre-processing stage is performed by rendering the scene from the viewpoint of the light source. Each polygon is colored with a "visibility ID," as it is rendered into the frame buffer. This technique is sometimes referred to as rendering into an "item buffer."

The frame buffer is scanned for visibility IDs which are used to index and set a flag in a visibility array. Later during the projection stage the visibility flag array is used to determine whether a specific target should be meshed, and whether a specific blocker contributes to the meshing of a target.

Notice in Figure 13 that the polygon ID 1 (dashed polygon) will neither be targeted for meshing, nor be used to compute the mesh for either polygon ID 0 or polygon ID 3.

In a typical scene there may be many polygons that will be rendered visible while a target is being meshed, but will not offer any contribution to the meshing for that polygon. Figure 13 shows that polygon ID 0 (behind the current target) and polygon ID 1 (completely occluded) will not contribute to the meshing of polygon ID 2, while polygon ID 3 (in front of target) will

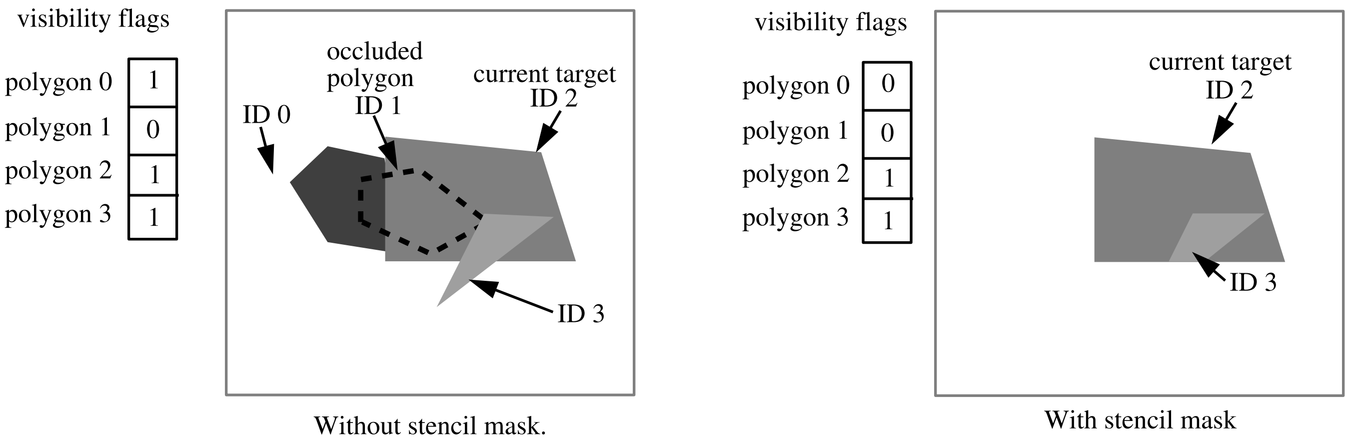

Figure 13. Visibility preprocessing and flags array

contribute meshing information. Polygons behind or outside the target can easily be eliminated by first rendering with a mask in the shape of the target, and then rendering the rest of the scene.

The masking can be accomplished via one of two methods. On systems with hardware support for stencils, the target can be rendered into the stencil plane first, which will mask out all non-overlapping polygons. On systems without hardware stenciling, the equivalent operation can be performed by first clearing the Z-buffer to its Z-maximum values. The target is rendered into it in a regular manner. The Z-buffer is then scanned and set to Z-minimum at every pixel which is equal to Z-maximum. The rest of the scene is rendered and the frame buffer scanned as before.

4.2.3 Projection and Edge Insertion

In the projection phase of the algorithm, potential targets are identified via the item buffer. Every polygon marked "visible" is considered a target. A target is meshed by projecting the visible polygons onto the target surface. In this way every polygon visible with respect to the light source is meshed by every polygon visible with respect to itself.

The perspective transformation is not an affine transformation and in general is not useful for taking measurements of an object: distances are not retained and angles are preserved only on faces which are parallel to the projection plane. We are concerned with extracting the discontinuity mesh on a target polygon by projecting blockers onto its surfaces and computing the 2D intersection points of their respective edges. If angles are preserved on the target then the object space triangle and its projection will be similar [9]. The parametric space of the 2-D intersection calculations will in turn be linear since the ratio of the projection to its world dimensions will be linear (See Figure 15). The trick is to force the target onto a plane parallel to the projection plane prior to projection. This can be achieved by eliciting a maximum of two rotations and a translation on the target.

In order to align a target polygon with the projection plane, the scene needs to be rotated in the appropriate manner for each target in question. The target should fall somewhere on the projection plane but it is not necessary that it fall in the center of projection. The transformation that makes this possible can be constructed by applying a translation of the viewpoint to the origin and performing a rotation about a single axis. The axis to perform the

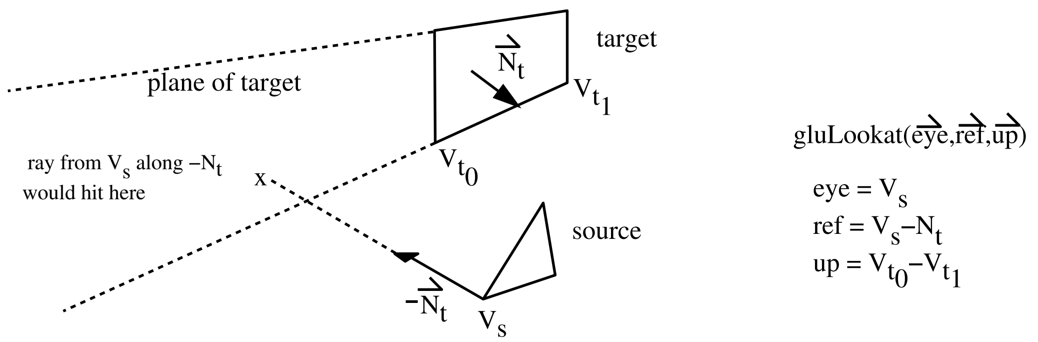

Figure 14. Aligning target to be parallel to the projection plane using gluLookat()

rotation is determined by the cross product of the Z-axis with the vector formed by the eye-to-reference point. For implementations on graphics libraries without arbitrary axis rotations, two rotations will be needed[1].

Shadow edges are determined by projecting blockers edge by edge onto the target. For blocker edges that cross the boundary of the target polygon, two dimensional line intersections are used to clip the edge. The edges derived from this stage are considered shadow boundaries and are

[1]The codes developed in this thesis make use of the *Silicon Graphics Inventor* ^{TM} and OpenGL graphics libraries. The gluLookat() command from OpenGL provides the means to accomplish the translations and rotations that are necessary to put the target onto the projection plane. The eye point is set at the vertex of the light source. To determine the reference point, the normal vector of the plane representing the target polygon is subtracted from the eye point. Calculating the reference point in this manner will map the target (plane) so that its normal is parallel to the negative z-axis. Choosing the up vector is somewhat arbitrary at this point. All we need is a vector that is perpendicular to the normal of the target. Using any of the three vectors that represent the sides of the target polygon will suffice. Figure 14 shows how the parameters for gluLookat() are determined. To extract the window coordinates of projected polygons the OpenGL gluProject() command was used. The reverse operation to back-project the polygons to their object space coordinates was accomplished with the gluUnProject() command.

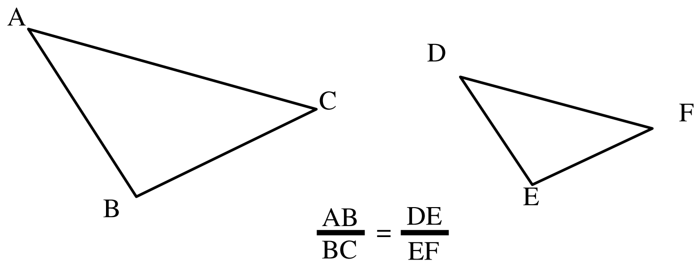

Figure 15. Similar triangles

inserted into an edge list. The edge list is used as input to a constrained Delaunay triangulation in the triangulation phase (see section 4.2.4).

Figure 16 shows the resultant edges that are inserted into the edge list after a blocker is projected onto a target. Notice that the original edges of the target triangle are always inserted into the edge list. In this example the blocker edges $\vec{EF}$ and $\vec{FD}$ are clipped but the entire edge $\vec{DE}$ is included in the edge list.

4.2.4 Triangulation

In this algorithm triangulation is applied after projection. The list of edges generated from the edge insertion stage (see Figure 12 and section 4.2.3) are sent to a constrained Delaunay triangulation routine. The Delaunay triangulation method is based upon the algorithm developed in [12] using codes that improve upon this method supplied by [22]. After this is completed the resulting triangles are back-projected to their three-dimensional object space counterparts.

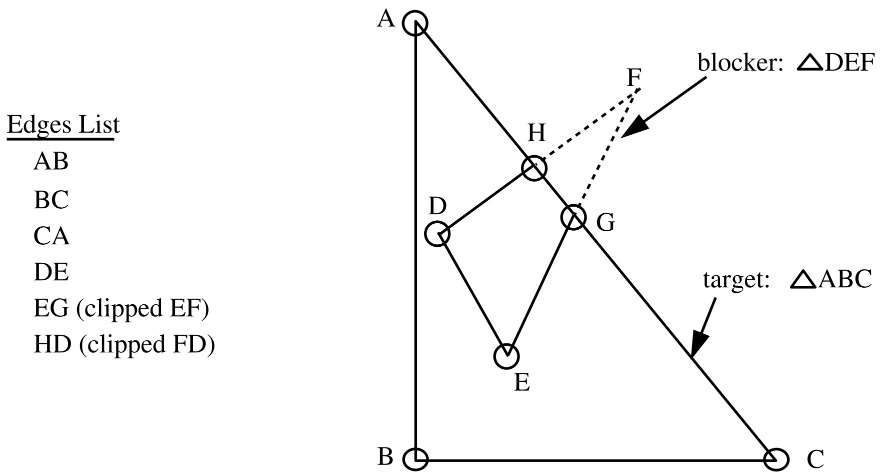

Figure 16. 2D edge clipping and insertion

4.2.5 Complexity

The complexity of the algorithm can be analyzed by looking at the pseudo code in Figure 12 and identifying stages with loops. Stage 1 contains the inner loops of light sources, stage 2 contains the render loop, stage 3 encompasses the vertices per light loop, and stage 4 has the Delaunay triangulation. We can ignore the stage 3 loop because light sources have a constant number of vertices (in this algorithm, three, since every polygon is decomposed into a triangle).

Every non-light polygon is compared against every other polygon in the scene via the loops in stage 1 and the render loop. This yields a complexity of,

$$O(m\,n^2) \tag{4.1}$$

Where,

n = p - m

mis the number of light sources, and p is the number of input polygons.

The Delaunay triangulation has a complexity of $O(n\,ln\,n)$ [12] which is greater than the (n) factor from the inner most loop of stage 2. Thus, the worst case complexity of the algorithm is,

$$O(m\,n^2\,ln\,n) \tag{4.2}$$

However, the render loop is very speedy due to its hardware nature and in practice is not a terribly limiting feature of this algorithm. As seen by the timings in chapter 5, complex scenes can be meshed in reasonable times.

CHAPTER 5

RESULTS

This chapter provides results of timings and renderings described in previous chapters, as well as implementation-specific details of parallel planar projective discontinuity meshing.

5.1 Parallel Progressive Refinement Timings

Timings for the parallel progressive refinement system are presented. For these timings, a scene consisting of one light source, a floor, and a 3D extrusion of the letter "R" was rendered (from now on referred to as the "R" scene). A wireframe drawing of this scene is shown in Figure 17. The total number of polygons processed was 23872. The number of processors in these timing tests varied from one to twelve. The timings were run on a production Silicon Graphics challenge array containing 12 150 MHZ Mips IP19 Processors. Timing tests were run during the day with normal operational loads.

Figure 18 shows a graph of timing runs on the scene depicted in Figure 17.

Graphs of the mean and standard deviation from these timings are shown in Figure 19 and Figure 20.

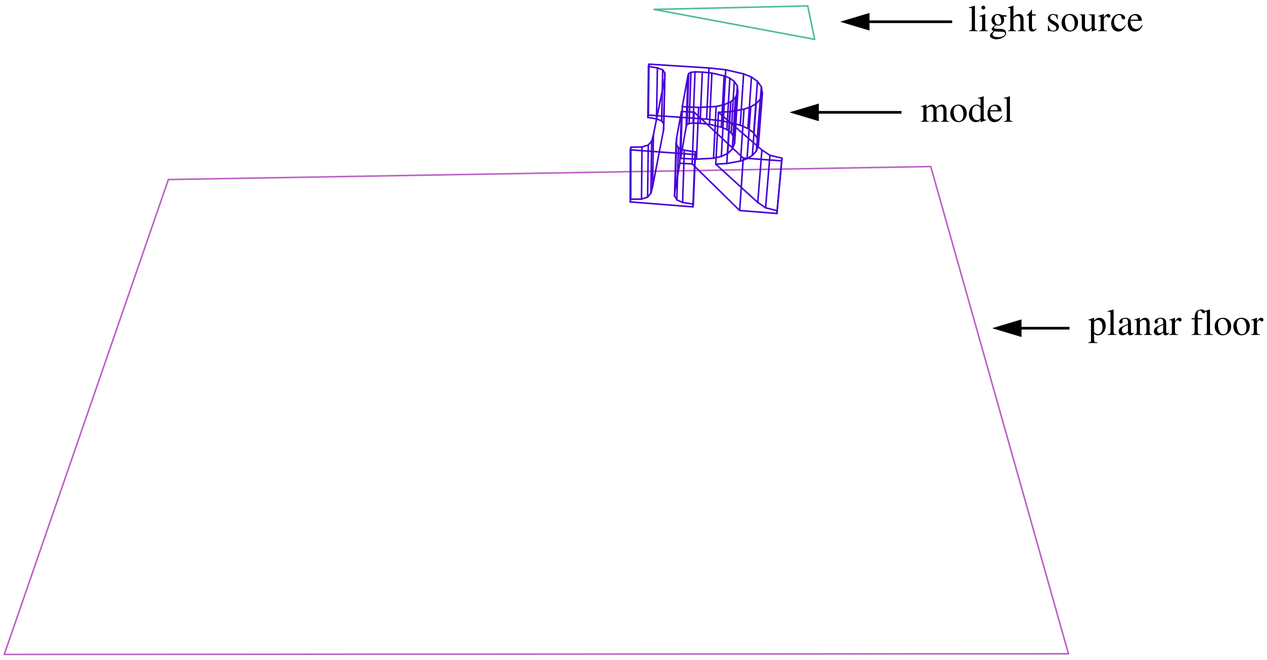

Figure 17. “R” scene: Wireframe geometry for parallel progressive radiosity timings

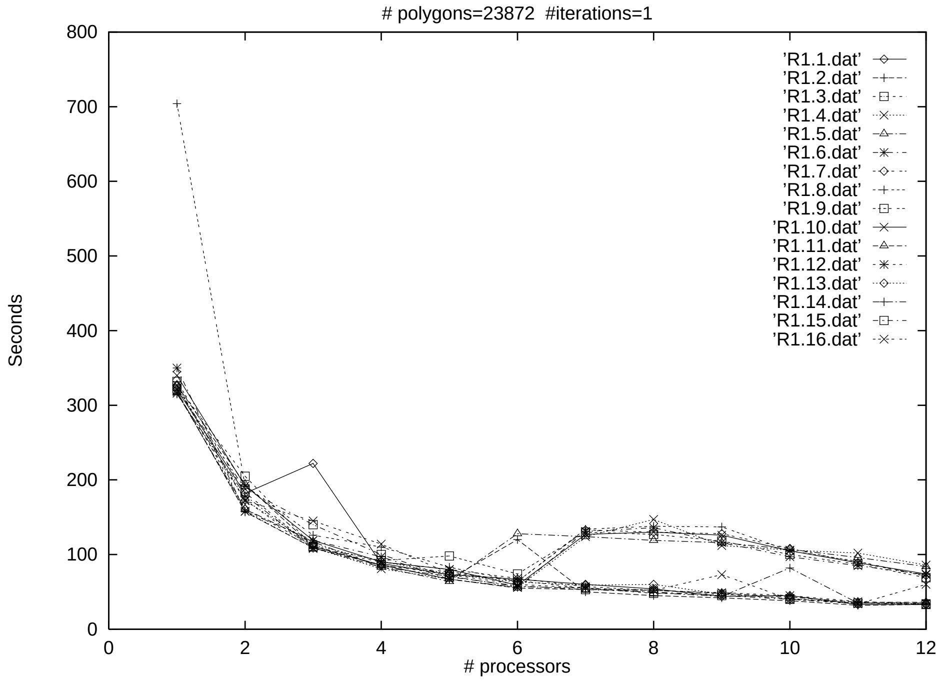

Figure 18. Parallel progressive radiosity timings for the "R" scene

5.2 Parallel Planar Projection Meshing Implementation Issues and Timings

This section covers some of the implementation issues surrounding the meshing method described in chapter 4.

5.2.1 Delaunay Triangulation

The Delaunay triangulation was made possible by a public domain implementation [22].

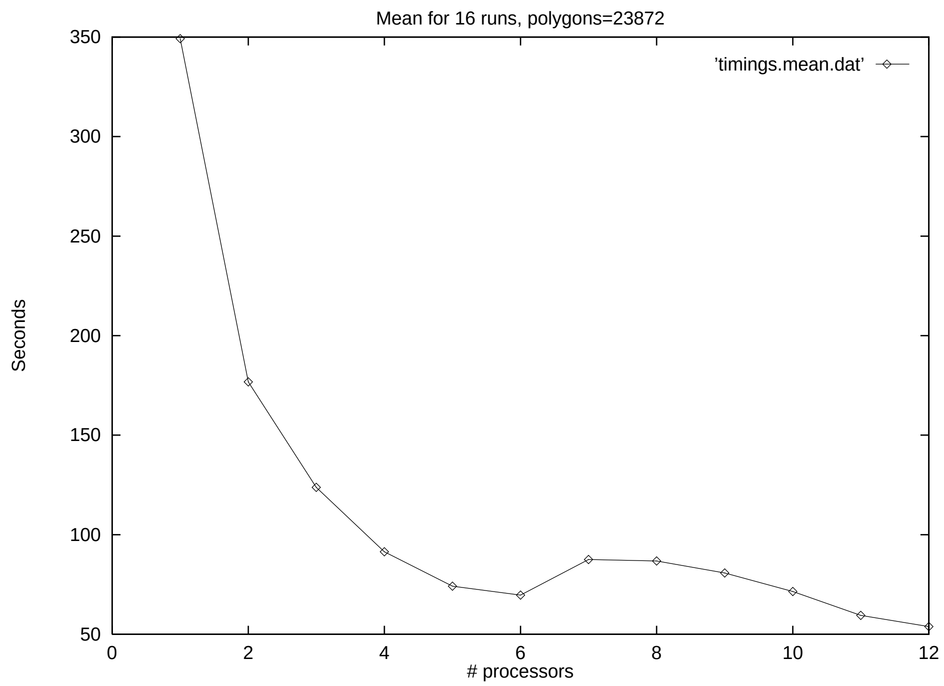

Figure 19. Mean plot from the "R" timings

5.2.2 Window Size

The window size chosen for the projection and culling stages is dynamically configurable. A square window region was used. In the culling phase a polygon is deemed visible if one or more screen pixels with its item buffer tag are rendered into the frame buffer. Polygons with small surface area will not contribute large gradients to the radiance function: Thus, a large window area is not required. As with the hemicube form factor method, aliasing is a potential problem. Larger window size reduces the chance for triangles to be missed at a tradeoff for rendering

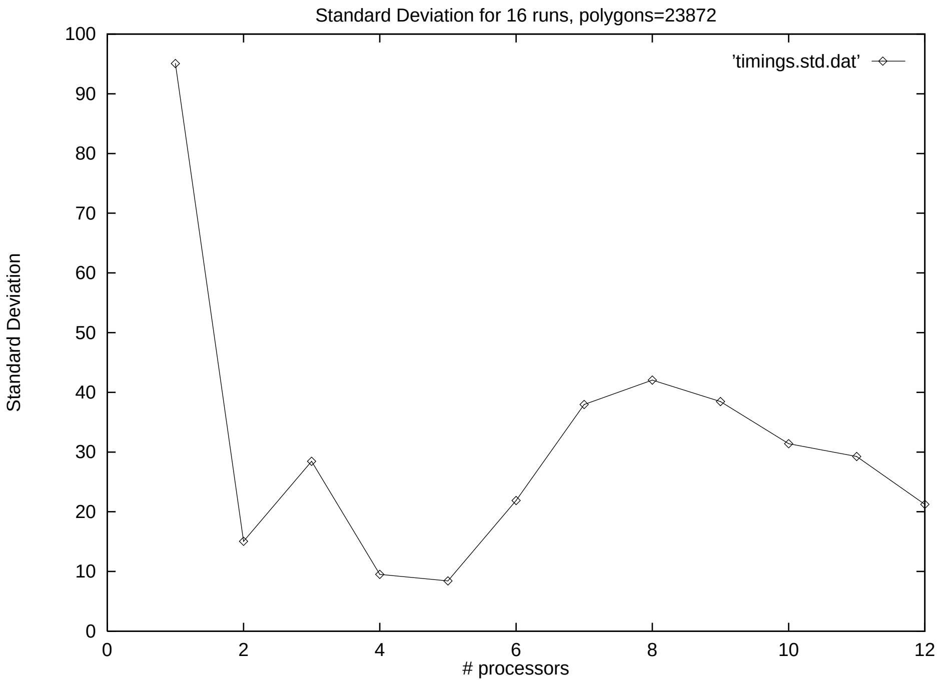

Figure 20. Standard deviation plot from the "R" scene timings

time during the culling phase. Typical window sizes used to mesh scenes in this thesis varied from 128 to 700 pixels on a side.

5.2.3 Parallel Planar Meshing and Rendering

The mesher was run across various geometrical data sets. Figure 21 shows the meshing sequence of a simple scene consisting of a triangular light source, a triangular blocker, and a rectangular floor (target). The three-image sequence depicts some of the stages in the mesher. Starting in Figure 21a we see the original data. Figure 21b shows the triangulation phase of the original input geometry (in wireframe mode so that the triangles can be identified). Figure 21c shows a wireframe display of the final projected, triangulated, mesh.

Figure 22 shows a close-up, top view of the meshed floor from Figure 21. Figure 23 shows renderings of Figure 21. Going from top to bottom, the first three images of the scene are meshed with a combination of the mesh generated from the parallel planar meshing system and the application of multiblocking as discussed in 2.4.1.1. The mesh level represents the amount of times the uniform meshing algorithm was applied. The images are rendered with the parallel rendering system described in Chapter 3. The additional meshing of the planar projected mesh is necessary in order to represent finer shading details across the surface. This is because the parallel planar mesher has no concept of separation distance. This is essential to the computation of accurate form factors. For patches that have a large area relative to their separation distance, the form factor approximation used in Equation 2.11 breaks down [30].

Figure 24 was generated from the same geometry used in the parallel progressive refinement timings in 5.1. Figure 25 shows a radiosity rendering of the mesh in Figure 24 using the parallel radiosity system described in chapter 3.

5.2.4 Timings

Table I shows parallel planar meshing statistics for scenes of various levels of detail. The timings were performed on an Onyx with a Reality Engine level II graphics engine.

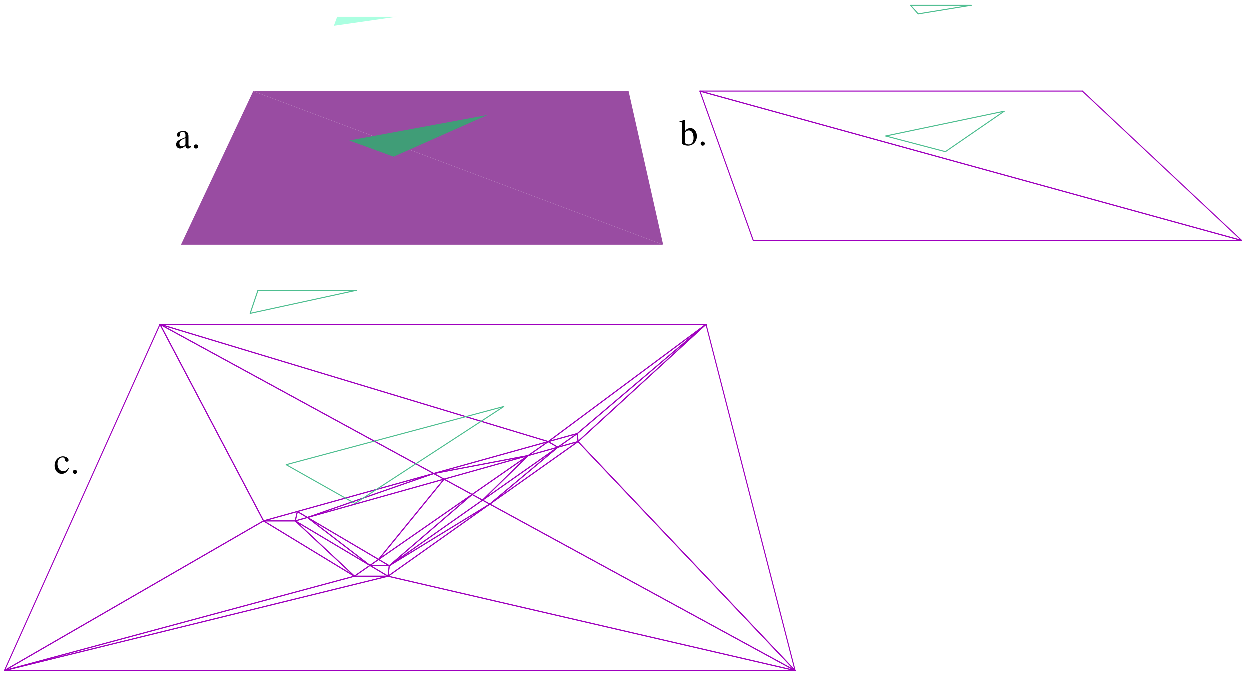

Figure 21. Meshing sequence for simple scene

TABLE I

PARALLEL PLANAR MESHING STATISTICS FOR SAMPLE SCENES

Scene	Input polygon count	Polygon count after meshing	CPU Clock (secs)	Wall Clock (secs)
Simple	3	40	0.170000	1
“R”	138	1729	5.030000	7
Vise	1050	7263	181.180000	232

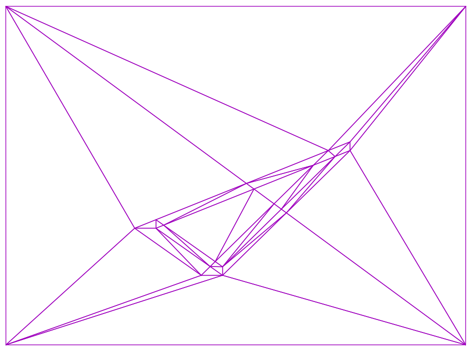

Figure 22. Top view of simple scene floor mesh

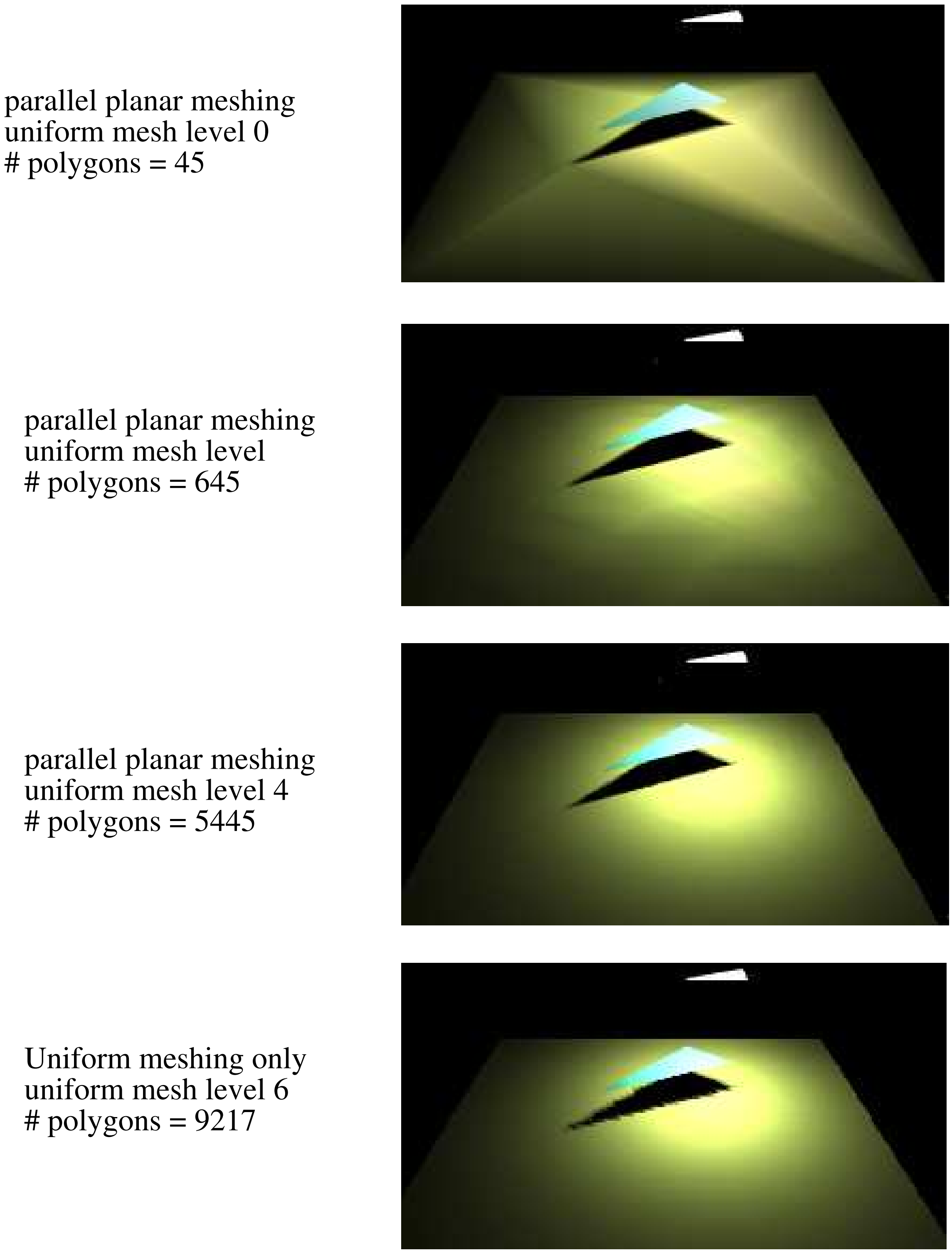

Figure 23. Renderings of simple scene at various mesh resolutions

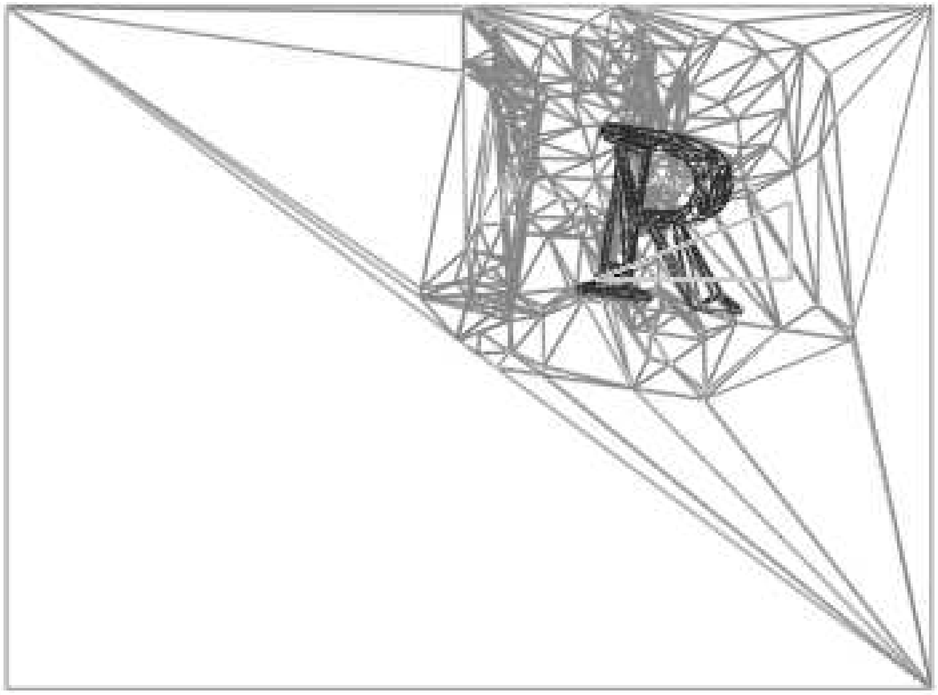

Figure 24. Resultant mesh from parallel planar projection of the “R” scene

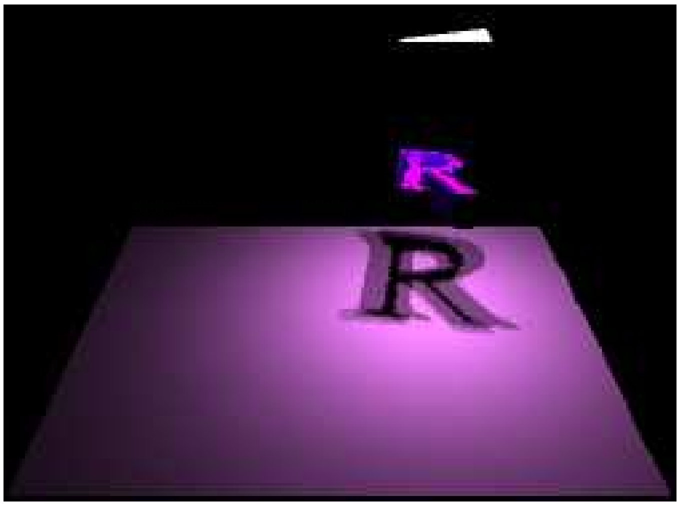

Figure 25. Radiosity rendering of the “R” scene

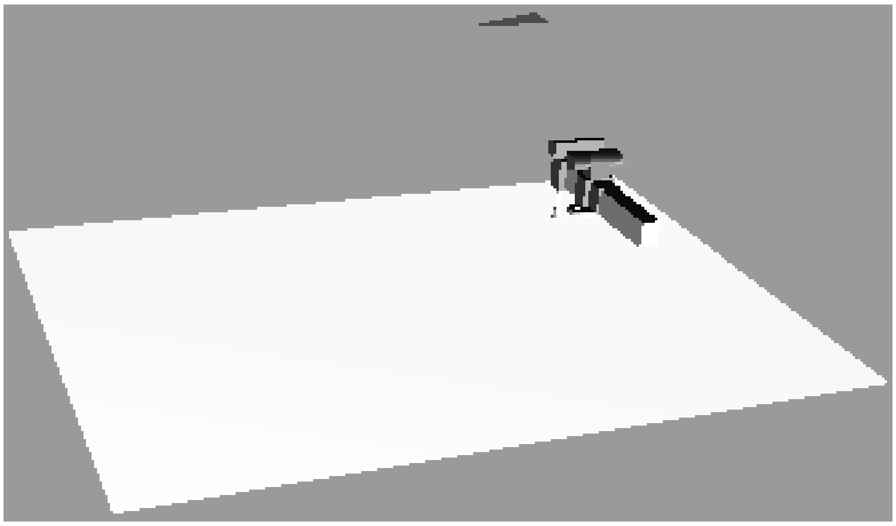

Figure 26. Vise scene used in meshing timings

CHAPTER 6

DISCUSSION AND FUTURE WORK

6.0.5 Discussion

This thesis approached the radiosity problem from two sides.

A parallel implementation of the progressive refinement algorithm was accomplished. Looking at the data depicted in the timing plots of Figure 19 we can see that adding more processors does in fact speed up rendering in a fairly linear fashion. For several of the timing runs there was actually an increse in rendering time by adding more processors. This is probably due to a combination of reasons: memory contentions and/or cache misses, and processor usage. As the number of processors increase, the frequency with which memory bank overlaps and cache stalls will increase.

The hump in the timings around the processor count of six can be most likely attributed to a busy system. This is evident in Figure 18 which shows a sudden increase in rendering time around the six processor mark. Probably six of the twelve processors in the system were being used by other processes during some of the timing runs. We can see in Figure 18 that for both the busy (top set of lines after the hump) and available (bottom set of lines after the hump) processors that there is a steady decrease in rendering time as more processors are added.

The parallel planar projection discontinuity mesher presented a new approach to the meshing problem. Certain stages rely on computation to be performed in graphics hardware. This differs from other approaches that compute shadow and discontinuity boundaries such as, shadow volumes by Nishita and Nakamae [19], back projection by Drettakis and Fiume [8], and Stewart, and Ghali [27], and discontinuity meshing by Lischinski, Tampieri, and Greenberg [17]. The use of specialized hardware for speeding up rendering algorithms goes back to the inception of the field. One such example is the use of the Z-buffer to generate a first hit list for ray tracers. The hemicube method as described in section 2.3.1 is another good example of a method that benefits from graphics hardware acceleration. If state-of-the-art hardware is available then parallel planar projection discontinuity meshing is an atractive alternative to other meshing strategies. For example, this algorithm was run on Silicon Graphics Reality Engine level II hardware. Level II Reality Engines are capable of generating up to 1.6 million triangle meshes per second. This kind of hardware performance enables an $O(n^2 \ln n)$ algorithm to run in reasonable times.

The example images that were generated from parallel planar projection and shown in the results section are of higher quality than their uniform meshing counterparts. However, there are some limitations and problems with this method. If we look closely at Figure 25 we can see some shading artifacts. This is due to the following reasons. The algorithm only handles Vertex-edge visual events, and the triangulation phase was not completely robust. The Delaunay triangulation produced copious amounts of triangles, some of which were degenerate in nature. There were many duplicate vertices which create shading artifacts when sent to the renderer.

A post processing phase to clean up the final triangulation did not always succeed in producing a completely topological equivalent of the initial input geometry. Thus, sometimes small holes form, edges overlap, and vertices were duplicated. In order to actually render the scene, the geometry had to be converted from Inventor to BOFF (Barnes Object File Format). The two formats were not completely compatible which may have contributed to some of the artifacts.

6.0.6 Future Work

Future work and research for the parallel planar mesher would be to speedup of the basic algorithm and extending it to handle a wider variety of visual events. Another improvement would be to put the mesher under a hierarchical umbrella as did Lischinscki et. al. in [16]. It is important to bound the form factor computation which would obviate the need for additional meshing that the renderer now performs. Another area for further research would be to experiment with a topological data structure, such as the winged edge [1], which would likely enable better handling of discontinuity segments and overlapping regions.

Parallelization of the parallel planar mesher is another area of future work. This could be accomplished by implementing the algorithm across multiple geometry engines. For example, our system consisted of three Silicon Graphics Reality Engines on an eight-processor Onyx. One process could be assigned to each geometry pipe. The database could be distributed accross these processors resulting in significant speedup of the algorithm.

CITED LITERATURE

[1] Baumgart, B. G. A polyhedron representation for computer vision. In *Proceedings of the National Computer Conference* (1975), pp. 589–596.

[2] Cohen, M. F., Chen, S. E., Wallace, J. R., and Greenberg, D. P. A progressive refinement approach to fast radiosity image generation. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (August 1988), J. Dill, Ed., vol. 22, pp. 75–84.

[3] Cohen, M. F., Greenberg, D. P., Immel, D. S., and Brock, P. J. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications 6*, 3 (March 1986), 26–35.

[4] Cohen, M. F., and Wallace, J. R. *Radiosity and Realistic Image Synthesis.* Academic Press Professional a Division of Harcourt Brace & Company, San Diego, CA, 1993. Excellent book on radiosity algorithms.

[5] Cohen, M. F., and Wallace, J. R. *Radiosity and Realistic Image Synthesis.* In [4], 1993, ch. 8, p. 212. Excellent book on radiosity algorithms.

[6] Cohen, M. F., and Wallace, J. R. *Radiosity and Realistic Image Synthesis.* In [4], 1993, ch. 8, pp. 216–221. Excellent book on radiosity algorithms.

[7] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* The MIT Press, 1992.

[8] Drettakis, G., and Fiume, E. A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)* (1994), pp. 223–230. available via the World Wide Web at: http://safran.imag.fr/Membres/George.Drettakis/pub.html.

[9] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer Graphics: Principles and Practices (2nd Edition).* Addison Wesley, 1990.

[10] Glassner, A. S. *Principles of Digital Image Synthesis*, vol. II. Morgan Kaufmann Publishers, Inc., 1995.

[11] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. Modeling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (July 1984), H. Christiansen, Ed., vol. 18, pp. 213–222.

[12] Guibas, L., and Stolfi, J. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics 4*, 2 (Apr. 1985), 74–123.

[13] Hanrahan, P., Salzman, D., and Aupperle, L. A rapid hierarchical radiosity algorithm. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)* (July 1991), pp. 197–206. Published as Computer Graphics (ACM SIGGRAPH '91 Proceedings), volume 25, number 4.

[14] Heckbert, P. S. *Discontinuity Meshing For Radiosity.* Technical report, UC Berkeley, 1991.

[15] Kajiya, J. T. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)* (August 1986), D. C. Evans and R. J. Athay, Eds., vol. 20, pp. 143–150.

[16] Lischinski, D., Tampieri, F., and Greenberg, D. P. Combining hierarchical radiosity and discontinuity meshing. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pp. 199–208.

[17] Lischinski, D., Tampieri, F., and Greenberg, D. P. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications 12*, 6 (Nov. 1992), 25–39.

[18] Nishita, T., and Nakamae, E. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), B. A. Barsky, Ed., vol. 19, pp. 23–30.

[19] Nishita, T., and Nakamae, E. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In *Computer Graphics (ACM SIGGRAPH '85 Proceedings)* (July 1985), pp. 23–30. Published as Computer Graphics (ACM SIGGRAPH '85 Proceedings), volume 19, number 3.

[20] Robert Siegel, J. R. *Thermal Radiation Heat Transfer*, second ed. Hemisphere Pub. Corp., 1981.

[21] Schröder, P., and Hanrahan, P. On the form factor between two polygons. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 163–164.

[22] Shewchuk, J. R. *Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator.* Available via World Wide Web at URL, http://www.cs.cmu.edu/q̃uake/triangle.html.

[23] Siegel, R., and Howell, J. R. *Thermal Radiation Heat Transfer.* Hemisphere Publishing Corp., Washington, DC, 1981.

[24] Sillion, F. X., and Puech, C. *Radiosity and Global Illumination.* Morgan Kaufmann Publishers, Inc., 1994.

[25] Sillion, F. X., and Puech, C. *Radiosity and Global Illumination.* In [24], 1994, p. 48.

[26] Sillion, F. X., and Puech, C. *Radiosity and Global Illumination.* In [24], 1994, p. 36.

[27] Stewart, A. J., and Ghali, S. Fast computation of shadow boundaries using spatial coherence and backprojection. In *Computer Graphics Proceedings, Annual Conference Series 1994 (ACM SIGGRAPH '94 Proceedings)* (1994), pp. 231–238.

[28] Teller, S. J. Computing the antipenumbra of an area light source. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 139–148.

[29] Wallace, J. R., Elmquist, K. A., and Haines, E. A. A ray tracing algorithm for progressive radiosity. In *Computer Graphics (SIGGRAPH '89 Proceedings)* (July 1989), J. Lane, Ed., vol. 23, pp. 315–324.

[30] Wallace, J. R., Elmquist, K. A., and Haines, E. A. A ray tracing algorithm for progressive radiosity. In *Computer Graphics (ACM SIGGRAPH '89 Proceedings)* (July 1989), pp. 315–324. Published as Computer Graphics (ACM SIGGRAPH '89 Proceedings), volume 23, number 3.

[31] Whitted, T. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June 1980), 343–349.

[32] Zatz, H. R. Galerkin radiosity: A higher order solution method for global illumination. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 213–220.

[33] Ziv Gigus, J. M. Computing the aspect graph for line drawings of polyhedral objects. *IEE Transactions on Pattern Analysis and Machine Intelligence 12*, 2 (Feb. 1990), 113–122.

VITA

NAME:	Jon Goldman
EDUCATION:	Bachelor of Science in Electrical Engineering, University of Rochester, Rochester, New York, 1984
	Master of Science in Computer Science, Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, 1995
HONORS:	Department of Education Fellowship, University of Illinois at Chicago - Award for research in computer graphics, 1994-1995
PROFESSIONAL MEMBERSHIP:	Association for Computing Machinery
PUBLICATIONS:	Goldman, Jon, and Roy, Trina M., "The Cosmic Worm," Visualization Blackboard, Computer Graphics and Applications, July 1994