Adaptive Mesh Generation for Radiosity Methods

by

Uwe M. Nimscheck

A Thesis submitted to the faculty of The University of Stuttgart in partial fulfillment of the requirements for the degree of “Diplom-Ingenieur”.

Department of Computer Science

The University of North Carolina at Chapel Hill

United States of America

1992

UWE M. NIMSCHECK. Adaptive Mesh Generation for Radiosity Methods (Under the direction of Professor Henry Fuchs and Professor Frederick P. Brooks, Jr.)

Abstract

The radiosity method is one of the main approaches to the generation of realistic images. The generation of accurate radiosity solutions imposes numerous constraints on the input data. Generally, a radiosity pipeline can be subidivided into modeling, preprocessing, mesh generation, radiosity solution and display. This thesis investigates the modeling requirements for valid radiosity solutions, reviews existing mesh generation algorithms for finite element methods, and introduces a new mesh generation algorithm for interactive radiosity approaches. The algorithm consists of a spatial decomposition by a modified quadtree and a subdomain meshing by removal of individual patches. It runs in O(n) time, with n being the number of generated patches. The use of a tree structure allows one to increase and decrease the density of the generated mesh locally and makes the mesh generator capable of adapting the mesh to the current lighting situation. This makes the proposed algorithm suitable for adaptive refinement and interactive radiosity solutions.

To my parents,

Acknowledgements

First of all, I should like to thank my thesis adviser Prof. Henry Fuchs, whose enthusiasm and competence made this whole work possible. He introduced me to the graphics group at UNC and supported my work in every way. I shall keep his unselfish way of support as one of the more memorable experiences at UNC.

Furthermore, I should like to thank Prof. Frederick P. Brooks, Jr. for his insightful comments on my thesis. His willingness to share his wisdom and knowledge and his systematic approach to the solution of scientific problems shall be of lasting impression to me.

Thanks are also due to Prof. Roland Rühle and Dr. Ulrich Lang, my thesis advisers at Stuttgart University for enabling me to write my thesis here at Chapel Hill. Their unbureaucratic support was one of the main factors to make my stay in Chapel Hill a successful one.

I am also grateful to Prof. Stephen M. Pizer of UNC, who admitted me here at UNC and Prof. Jan Cornelis of Free University of Brussels, who encouraged me to go to Chapel Hill.

Furthermore, I should like to thank UNC graduate students Amitabh Varshney for proofreading my thesis and many helpful suggestions; Andrew Bell who explained the Pixel-Planes radiosity to me; David Lines, my roommate; Greg Turk for some most helpful pointers to the mesh generation literature; Hans Weber, for innumerable helpful comments and for reading the first draft of my thesis; Hong Chen for many helpful discussions; John Alspaugh, who helped me with all the existing code and proofread my thesis; Ron Azuma for his support during my whole stay in Chapel Hill; and Yulan Wang for her help with the Walkthrough code.

Contents

List of Figures

Chapter 1

Introduction and Overview

One of the main problems in the technical design of geometric objects and spaces is a realistic visualization in order to obtain a better understanding of the geometric properties and appearance of the designed object. This holds for the design of merely functionally oriented applications (such as designing a gearbox) as well as consumer-oriented applications (architectural design of a house, for instance). In the first case, the engineer might try to obtain a realistic impression of an object of possibly complex geometric shape, in the second case it might be the architect's intention to give his client an idea of the appearance of his proposed design of a house.

Lighting models give the displayed object a more realistic appearance and thus lead to a better perception of its actual shape and properties. Lighting models can be divided into *global* and *local* lighting . Global lighting determines the color at a point dependent on light directly emitted by light sources and light reflected from and transmitted through other surfaces [FOLE90]. The radiosity method [GOHA84] is a global lighting model that is especially suited for the modeling of lighting interreflections and color bleeding. One of the obstacles to a wide acceptance of the radiosity method is the complex modeling process required to obtain accurate radiosity solutions. Traditionally, the generation of a radiosity solution includes several preprocessing steps preparing the output of a modeling system for the calculation of the radiosity solution. These steps require a user well familiar with the radiosity method and its specific demands. Generally, users do not fulfill these requirements and therefore are unable to obtain accurate solutions.

A design and visualization process may consist of the following steps:

- Generating the model on a CAD-System such as AutoCAD
- Preprocessing the model in order to generate a topologically and geometrically valid model

- Generating the mesh

- Computing the radiosity solution

- Displaying the computed solution.

There are two loops conceivable in this scheme:

If the results of the computed radiosity solution are used for a modified meshing of the preprocessed model leading to a new, improved radiosity solution, the procedure is called *adaptive refinement*. These two steps can be repeated until an image of sufficient quality is obtained [COHE88].

Allowing the user to add or move objects in the modeling system might lead to a completely different lighting situation; if so, it results in a loop over all five steps. This may be called an *interactive radiosity* method and implies new demands on the speed of the processes in the pipeline.

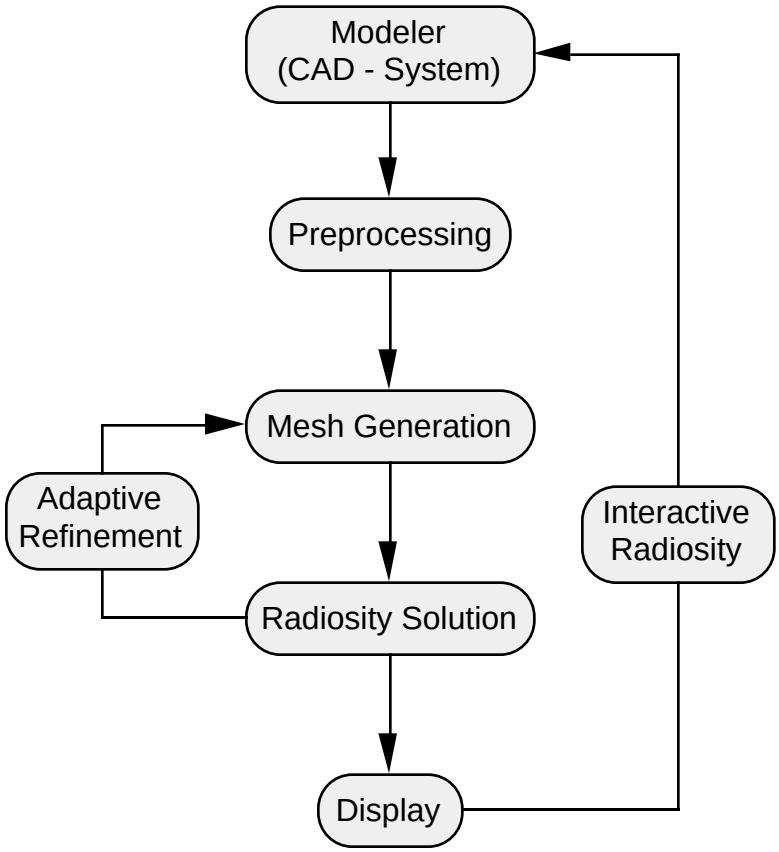

Figure 1.1: The radiosity pipeline

Figure 1.1 shows the radiosity pipeline from the modeling to the display as well as the adaptive refinement and the interactive radiosity loop.

The first step is the generation of the model on a modeling system (modeler). The choice of this modeler has a great impact on the quality of the model in terms of visualization. Most of the commercially available low-end CAD packages such as AutoCAD were conceptualized as an aid for the production of technical drawings and are not well suited for the design of realistic models. Generating and displaying an accurate radiosity solution imposes many geometrical and topological requirements on its input. Hence complex preprocessing is necessary to make such a model usable for radiosity methods. A newer approach to the design process is solid modeling, where models are generated by boolean set operations (such as union, difference, or intersection) on simple geometric primitives. The output of these modelers is better suited for the design of radiosity solutions, however the modeling process itself is generally more complex.

This thesis uses a new radiosity pipeline, consisting of:

- A modified version of the commercially available spatial design tool VIRTUS WalkThrough [VIRT91]
- A new, quadtree-based approach to mesh generation
- An interactive radiosity solution and display on Pixel-Planes 5, a parallel graphics engine[FUCH89].

The model generation is done by VIRTUS WalkThrough. This is a spatial design tool for the Apple Macintosh and supports solid modeling in 2 1/2 D. Models consist of 3-dimensional primitives. These spatial primitives are polyhedra, in particular extruded convex polygons. Primitives can be attached to each other, thus allowing objects of complex shape. WalkThrough maintains a strong hierarchy in containment. One primitive can be inside or outside of another primitive, but no intersection is allowed. Furthermore, a primitive may have polygonal holes attached to its surface. The output of this modeler consists of the polygons that describe the boundaries of the spatial primitives together with the features (holes) attached to them.

The output polygons are generally not suitable for direct use in a radiosity solution, so they have to be preprocessed. Depending on the quality of the original model, a variety of steps may be required in order to yield a physically valid model. This can include processes such as merging adjacent vertices or dealing with polygon intersections. These steps, however,

are strongly dependent on the kind of modeling. The choice of the modeler can reduce the number and complexity of these preprocessing steps significantly.

The process of converting the output primitives of the modeler into small convex quadrilaterals and triangles (*tiles*) suitable for the radiosity solution is called *mesh generation*. The radiosity method is basically a finite element method (FEM), hence subdivision algorithms used for finite elements can be adapted for the preprocessing of the radiosity input. Shephard [SHEP88] surveys four common approaches to mesh generation in finite element analysis:

- Point placement followed by triangulation
- Removal of individual subdomains
- Recursive subdivision of the domain
- Spatial decomposition followed by subdomain meshing.

In the approach of point placement followed by triangulation, the generation of the tiles is done in two distinct steps. The first is to place points throughout the area of interest. In the second step, these points are triangulated into a mesh. Delaunay triangulation is an example of this strategy.

Mesh generation based on sub-domain removal operates by splitting tiles from the region one at a time until the domain is reduced to a single tile. There are also schemes available which initially subdivide a complex region into simple parts and triangulate these regions by a different technique (such as those above). This may help to handle complex shapes such as concave polygons or polygons with holes.

The recursive subdivision algorithm repeatedly splits a region into smaller subregions, until it is either a tile of acceptable shape and size, or of such simple shape that valid tiles can be generated easily.

Spatial decomposition followed by subdomain meshing is a two-step approach. First the initial domain is subdivided into simple regions and then these regions are meshed individually. The initial decomposition is typically done by a quadtree in the two dimensional case or an octree in the three-dimensional case.

The approach used in this thesis work is an example of such a modified quadtree technique in two dimensions. A two dimensional data structure can be used since the input model consists of two dimensional primitives (polygons). Motivations for the choice of this

particular meshing technique are given in chapter 4. The mesh generation is done as follows: One region of interest (a polygon of convex or concave shape, with or without holes) is subdivided into the quadrants of a quadtree with the quadrant size defining the density of the mesh. This is done by taking each edge of the polygon (or hole) and recursively subdividing it until each fragment fits within a quadrant of the tree. A quadrant is allowed to be only one level finer then the adjacent quadrants. This leads to a certain homogeneity in the mesh. Each quadrant is classified into one of three classes:

- Inside the polygon. No edge falls in this quadrant. This quadrant might be used as a single tile or subdivided by a predefined scheme.

- Outside the polygon. No edge falls in the quadrant. It is outside the polygon and therefore yields no tiles.

- Boundary region. There are one or more edges stored in this quadrant, so the inside and outside parts of the quadrant have to be determined and the inside part has to be further processed by one of the above mentioned mesh generation methods to yield a set of valid tiles.

Then the tree is traversed quadrant by quadrant and the tiles of each quadrant are generated according to the above rules.

Figure 1.2 shows a polygon and the generated mesh.

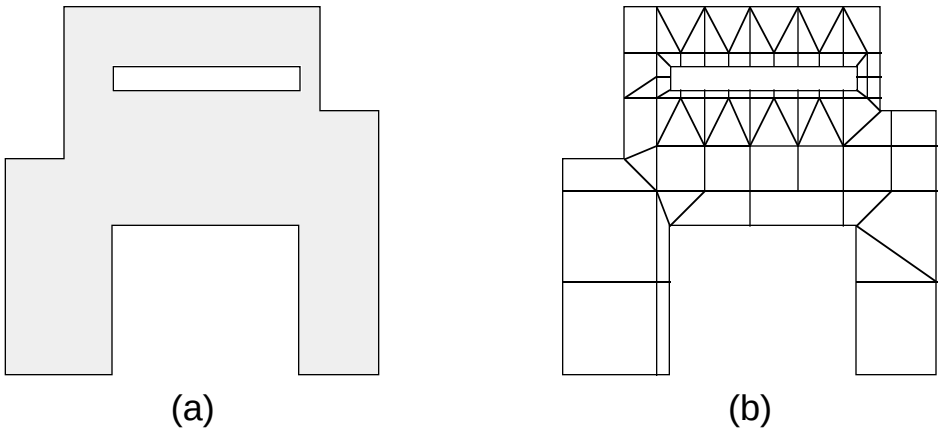

Figure 1.2: Example of a surface (a) and a generated mesh (b)

The next step in the pipeline is the radiosity computation. Radiosity is one of the common approaches to rendering and a powerful tool for the generation of high quality images.

Based on a physical model, the heat-transfer principle, it is the most efficient way to handle lighting interreflection and color-bleeding phenomena. One of the main advantages of radiosity is the view-independence of the generated solution, making it suitable for applications where the observer is moving around in the scene. Furthermore, radiosity solution computation can take significant advantage of the hardware of modern graphics workstations for its faster implementation. This makes it suitable for use in applications involving interactive model modifications. The proposed pipeline uses a parallel implementation of the radiosity algorithm on Pixel-Planes 5, a hybrid SIMD / MIMD high-end graphics engine [FUCH89].

The radiosity solution can be displayed on conventional graphics hardware using shading models such as Phong Shading or Gouraud Shading. This step involves the selection of the viewing parameters, determination of the hidden surfaces and the interpolation of the radiosity values.The proposed pipeline uses the Pixel-Planes 5 graphics engine for real-time rendering and displays on a high resolution monitor or a head-mounted display (HMD).

The remainder of this thesis is organized as follows:

Chapter 2 describes the fundamentals of the radiosity method and delineates the specific requirements of the modeler.

Chapter 3 dicusses the demands radiosity methods impose on the modeling process. Various modeling artifacts caused by invalid models are discussed as well as preprocessing steps necessary to obtain valid data.

Chapter 4 provides an overview of existing modeling approaches to mesh generation in FEM methods and evaluates these methods with respect to the specific requirements of radiosity methods.

Chapter 5 describes the algorithms employed in the modified quadtree mesh generator, an approach well suited for radiosity methods

Chapter 6 shows meshes generated with the modified quadtree approach and the corresponding radiosity solutions. The quality of the meshes and radiosity solutions is discussed.

Chapter 7 contains the conclusions and gives an overview of possible extensions to this work.

Chapter 2

The Radiosity Method

The radiosity method is one approach to the generation of realistic images of virtual scenes. This chapter describes the fundamentals of the radiosity method and delineates the specific requirements of the modeler.

The radiosity solution of a given model determines how to shade surfaces based on position, orientation, and characteristics of the surfaces and light sources illuminating them. Based on a thermal engineering model for the emission and reflection of radiation, the radiosity method was introduced to computer graphics by Goral et al.[GORA84]. It provides an accurate treatment of interobject reflections, making it an effective solution to the problem of lighting interreflection and color bleeding for diffuse surfaces.

2.1 The Radiosity Equation

Radiosity algorithms assume the conservation of energy in a closed environment. All surfaces are supposed to be *Lambertian diffuse reflectors or emitters*, i.e., the incident light is reflected from a surface with equal intensity in all directions. Specular reflections are not taken into account. The rate at which energy leaves a surface is called its *radiosity*. It is the sum of the rates at which the surface emits energy and reflects or transmits it from other surfaces. The radiosity of a differential area dA_i is determined by:

$$B_{dA_i}\, dA_i = E_{dA_i}\, dA_i + \rho_{dA_i} \int_j B_{dA_j} F_{dA_j - dA_i}\, dA_j \qquad (2.1)$$

with: B_{dAi} = Radiosity of differential area dA_i

dA_i = Differential area i

E_{dAi} = Emission of differential area dA_i

ρ_{dAi} = Reflectivity of differential area dA_i

$F_{dAj - dAi}$ = Form factor from dA_j to dA_i

The problem can be made tractable by discretizing the surfaces of the environment into *finite patches* A_i. This leads to

$$B_{A_i} A_i = E_{A_i} A_i + \rho_{A_i} \sum_j B_{A_j} F_{A_j - A_i} A_j \qquad (2.2)$$

This expression assumes homogeneous distribution of the radiosity over the patch area. The accuracy of the solution is dependent on how well this assumption is fulfilled. The underlying problem is a sampling problem: A continuous function has to be approximated by discrete samples, and the sampling rate affects the accuracy of the approximation. Choosing a small patch size should lead to more accurate solutions, but it increases the computational expense as well. One of the main criteria for computing a good radiosity solution is the right choice of this patch size.

The *form factor* $F_{dAj - dAi}$ defines the fraction of energy leaving patch A_j that arrives at patch A_i. For Lambertian surfaces the following relationship holds:

$$F_{A_i - A_j} A_i = F_{A_j - A_i} A_j \qquad (2.3)$$

Substituting (2.3) into (2.2) and dividing by A_i yields the basic radiosity equation for finite area patches:

$$B_{A_i} = E_{A_i} + \rho_{A_i} \sum_j B_{A_j} F_{A_i - A_j} \qquad (2.4)$$

Figure 2.1 gives a geometric interpretation of the above equation.

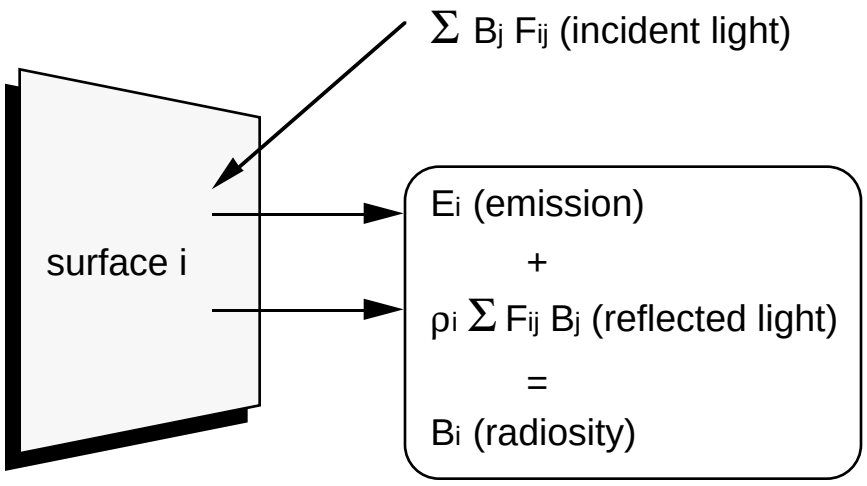

Figure 2.1: Radiosity relationships

The following system of equations states the interaction of light among the patches:

$$\begin{bmatrix} 1 - \rho_1 F_{1-1} & - \rho_1 F_{1-2} & \cdots & - \rho_1 F_{1-n} \\ - \rho_2 F_{2-1} & 1 - \rho_2 F_{2-2} & \cdots & - \rho_2 F_{2-n} \\ \vdots & \vdots & \vdots & \vdots \\ - \rho_n F_{n-1} & - \rho_n F_{n-2} & \cdots & 1 - \rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \qquad (2.5)$$

The color of a patch i is a function of the reflectivity ρi and emission E_i of each wavelength in the visible light spectrum, so (2.5) must be formed and solved for each wavelength considered in the lighting model. The form factors F, however, are merely functions of the model geometry and are thus independent of the wavelength.

The number of equations n in the above system is determined by the number of patches. Generally, a fine grid of patches can approximate the inhomogeneous distribution of the radiosity better than a coarse grid and thus leads to a more accurate solution. However, the exact solution of the system requires $O(n^2)$ space and $O(n^3)$ time, making it quite expensive even for systems with a few hundred patches. If all patches are assumed to be planar, all F_{n-n} are 0 and the matrix becomes strictly diagonally dominant. This makes iterative methods such as Gauss-Seidel suitable for its solution. These methods typically converge rapidly, thus leading to $O(n^2)$ computation expenses. A major shortcoming of this approach is that in order to obtain a first radiosity estimate for all patches, one whole iteration cycle

has to be completed. A continuous update of the patch radiosities would be favorable for interactive applications, trading off accuracy for time in the early stages of the solution.

2.2 Progressive Refinement

Cohen et al. [COHE88] propose a progressive refinement method for the radiosity algorithm. It is also based on the Gauss-Seidel algorithm, but the operations in the iteration cycle have been changed in order to obtain useful, although inaccurate solutions early in the iteration process. This is accomplished by two changes to the above algorithm:

- The radiosity of all patches is updated simultaneously
- Patches are processed in sorted order according to their light energy contribution to the environment, i.e., patches with high energy are processed first.

The Gauss-Seidel approach solves system (2.5) row by row. The radiosity B_{Ai} for patch i is estimated on the basis of the current estimates of the radiosities of all patches, it is the sum of the contributions of all patches to the radiosity of patch i.

Thus, in this approach the radiosity of a patch is determined by gathering the radiosity contributions of the other patches, i.e. the environment. The progressive refinement approach reverses this procedure by determining patch i's contribution to the radiosity of all patches, called shooting. Figure 2.2 shows the principal difference.

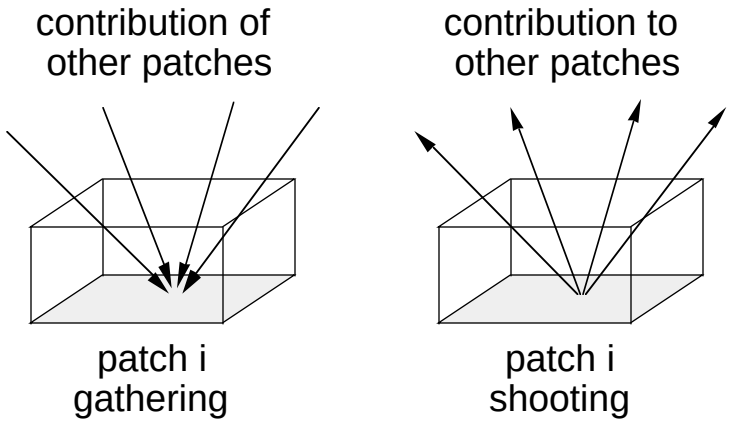

Figure 2.2: Gathering and shooting

Instead of evaluating (2.4) for each patch, the progressive refinement approach calculates

$$B_{A_j} = B_{A_j} + \rho_{A_j} B_{A_i} F_{A_j - A_i} \qquad (2.6)$$

for all j, thus each step of the solution now consists of *shooting* the stored radiosity of one patch into the environment and adding the respective contributions to all patches of the environment. This yields a new estimate for the radiosities of all patches at each step of the iteration. After updating the radiosities of the patches, a new shooting patch is chosen. Since one patch can be chosen more than once during the iterations, only ΔB_{Ai}, the difference in radiosity needs to be shot. One common practice is to chose the patch with the highest unshot energy to shoot next. Initially, the radiosity of a patch is set to its emission,

$$B_{A_i} = \Delta B_{A_i} = E_{A_i} \qquad (2.7)$$

which is zero for all patches except emitters.

This algorithm is suitable for interactive applications, since it provides new updates for all patches at each step.

2.3 Computing the Form Factors

The form factor determines the fraction of the radiosity leaving one patch which reaches another.

$$F_{A_i - A_j} = \frac{1}{A_i} \int_{A_j} \int_{A_j} \frac{\cos \phi_i \cos \phi_j}{\pi r^2} \, dA_j \, dA_i \qquad (2.8)$$

Figure 2.3 gives a geometric interpretation.

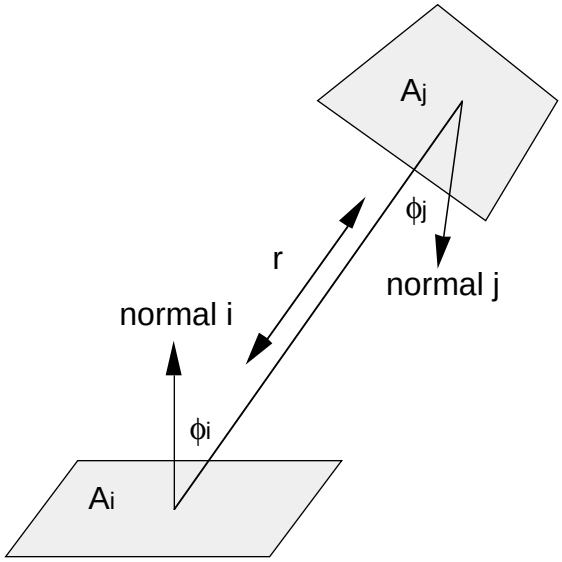

Figure 2.3: Form factor between two patches

Computing the form factors is the computationally most expensive part of the radiosity solution, it can be done either analytically or by using geometric approximations.

Nusselt [SIEG82] has introduced a geometric analog for the form factor. The form factor for a patch is represented by the ratio of the area of the patch, projected first on the hemisphere and then down on the base, to the area of the whole base. This holds, however, only if the distance between the two patches is large compared to their sizes. Figure 2.4 shows the Nusselt Analog.

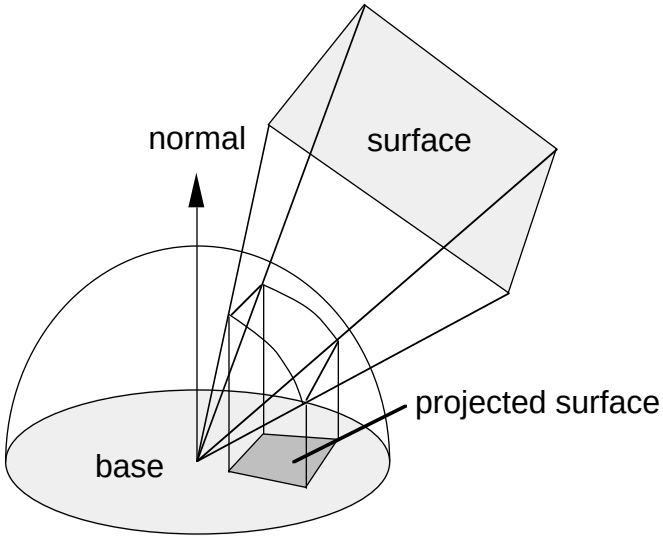

Figure 2.4 The Nusselt Analog

There are different ways of determining form factors from this analog. Some high-end graphics computers such as Pixel-Planes 5 can evaluate quadratic expressions such as spheres directly in hardware. If the hardware does not support these features, a computationally less expensive method such as the hemicube [COHE85] can be used.

2.4 Determination of Vertex Radiosities

The radiosity solution gives radiosity values for the whole patch, but most lighting models (such as Phong shading or Gouraud shading) require radiosity values for each vertex. Cohen [COHE85] determines the vertex radiosities using the following process of extrapolation and interpolation (Figure 2.5):

- For vertices interior to a polygon (vertex i), the vertex radiosity is the average of all the patches sharing this vertex.

- Boundary vertex radiosities (vertex a) are extrapolated from interior vertices and adjacent patch radiosities.

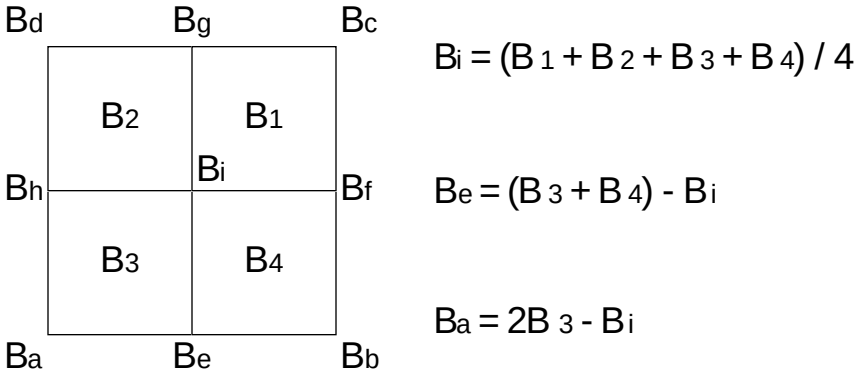

Figure 2.5: Interpolation and extrapolation of vertex radiosities

This strategy for the determination of vertex radiosities imposes certain constraints on the subdivision of polygons into patches. Realistic rendering requires at least first order continuity at patch edges. Figure 2.6 shows an example of a subdivision that does not guarantee a first order continuity.

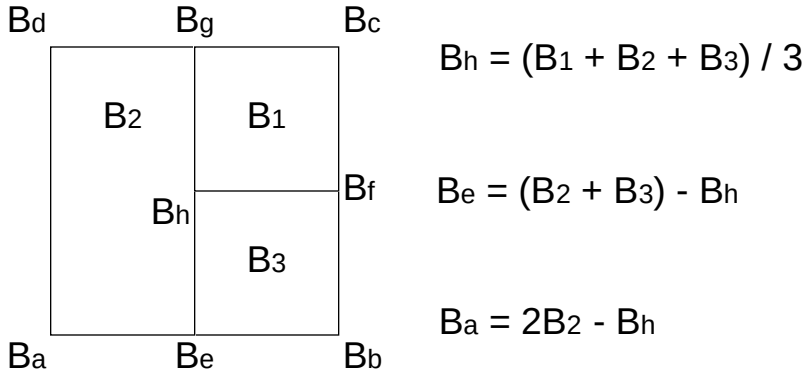

Figure 2.6: Generation of t-vertices

The determination of radiosities along the edges (Bg-Be) is done by linear interpolation of the vertex radiosities of the two vertices defining this edge. Assume $B_1 = 3$, $B_2 = 4$, $B_3 = 8$. This yields:

$B_h = (3 + 4 + 8) / 3 = 5,$

$B_e = 4 + 8 - 5 = 7,$

$B_g = (3 + 4) - 5 = 2.$

Linear interpolation along edge (B_g-$B_{e)}$ gives ($B_g + B_e$) / 2 = 4.5 for a vertex B_h, as opposed to $B_h = 5$ for the other side of edge. When displaying, this would be visible as a sharp change in brightness across boundary ($B_g + B_e$).

Vertices leading to these artifacts are known as *t-vertices*. Polygon subdivision algorithms have to avoid the generation of such vertices.

Chapter 3

Modeling

This chapter discusses the requirements radiosity methods impose on the modeling process. Various artifacts caused by invalid models are discussed as well as the preprocessing steps necessary to obtain valid input data. Furthermore, the influence of the modeler concept (polygon-based or solid-based) on the quality of the input model is investigated. The chapter closes with a discussion of the VIRTUS WalkThrough modeling system.

A *geometric modeling system* or *modeler* is a computer-based system for creating, editing, and accessing representation of solid objects. Objects are created and modified through an interactive graphics command language. Most input languages are CSG (Constructive Solid Geometry), sweep, or boundary based. The sequence of input statements is itself an object representation, but generated objects are stored as an internal representation of the model. It is this internal representation that largely determines the usefulness of the modeler for the application program, because it contains the data continuously accessible to the application program.

The demands of the radiosity algorithm on the input data can be divided into model geometry requirements and radiosity mesh requirements.

3.1 Geometry Requirements

The generation of the model represents the first step in the radiosity pipeline. The radiosity method is a physically based illumination model, thus it imposes several demands on the physical validity of the input data. Most radiosity users, however, are not aware of these specific radiosity requirements. There are two ways to generate suitable models for high quality radiosity solutions without the need of user interaction:

- Employing suitable modeling software that enforces the generation of geometrically and topologically valid input data

- Preprocessing of the input model database in order to convert it into a form suitable for the radiosity algorithm.

A physically valid representation of a solid is called an *abstract solid*. Requita [REQU80] surveys the requirements to an abstract solid as follows:

- Rigidity: An abstract solid must have a shape or configuration which is independent of the solid's location and orientation.

- Homogeneous three-dimensionality: A solid must have an interior, and a solid's boundary cannot have isolated or dangling portions.

- Finiteness: A solid must occupy a finite portion of space.

- Closure under rigid motions and Boolean operations: Rigid motions or Boolean set operations on valid solids must yield other valid solids.

- Finite describability: Solids must be representable by a finite number of planes.

- Boundary determinism: The representation must determine unambiguously if a point is inside, outside, or on the surface of a solid.

There are different types of valid representations of solids, and they are based on standard mathematical set operations. Examples of representations are cell decomposition, constructive solid geometry, sweep representations, and boundary representation schemes. Boundary representations are most convenient for computer graphics applications such as radiosity, since the object boundaries are the natural object of interest for these algorithms.

Most of the low-end CAD software packages (such as AutoCAD) do not enforce the generation of suitable output for radiosity solutions. They typically use polygons (3D-faces) as modeling primitives. It is the user's responsibility to use these primitives for the description of physically valid scenes. Solid-based modelers force the user to generate valid models, but these modelers are generally more complex and thus computationally more expensive.

One of the main artifacts of polygon based-modelers are *facades*, surfaces that are visible from both sides. The radiosity algorithm determines the radiation energy per surface, hence surfaces that are exposed to radiation from two sides give inaccurate results. Lamp shades made up of polygons are an example of this artifact (Figure 3.1). Facades can only occur in

modelers that use polygons or lines as primitives. One straightforward fix would be to make every surface consist of two polygons, one facing inward and one outward. However, this doubles the model size and is undesirable, since the radiosity solution runs in $O(n^2)$ time, with n being the number of patches.

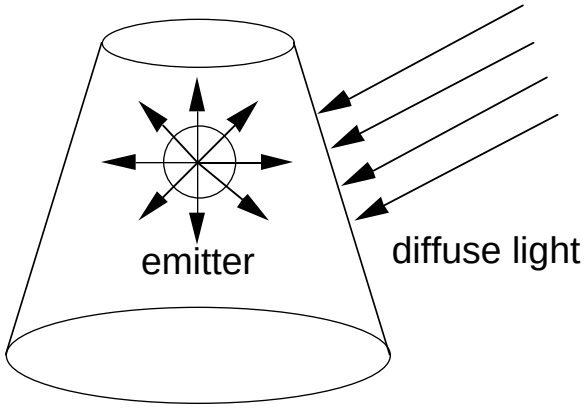

Figure 3.1: A lamp shade as an example of a facade

Another artifact occurs if a facet is coplanar with a solid. Figure 3.2 shows a solid model where the boundaries of two solids are adjacent (a), and a suface coplanar with a solid (b). Case (a) does not represent a problem for the rendering, since the area of coincidence is not visible to the observer. In case (b), however, both faces are visible. This causes problems for rendering on z-buffer hardware, since the algorithm cannot determine which face is visible and which is occluded.

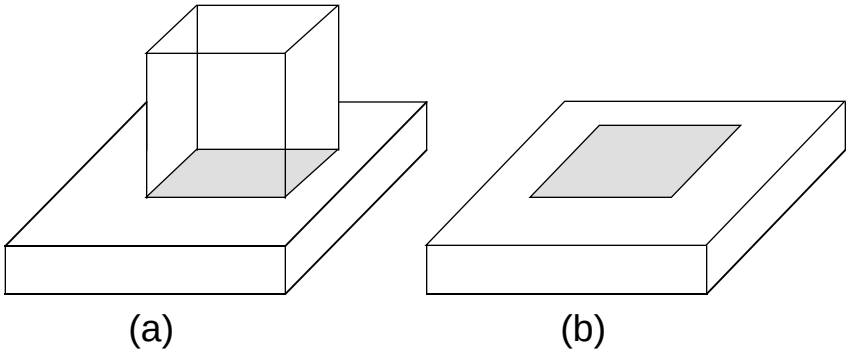

Figure 3.2: Coplanar polygons

Both situations in Figure 3.2 should be avoided, however, because they give rise to erroneous radiosity solutions. The radiosity algorithm has to determine the mutual visibility of all patches. Since two patches are coincident, there is no way to determine which is occluded. One obvious way to avoid this problem is simply to leave a small space between the two polygons, i.e. to let the box “float” over the ground. If the space between these two polygons is sufficiently small, it would not be visible to the observer. The drawbacks of this solution will be discussed later in this chapter.

Interpenetrating polygons are another artifact of physically invalid models. They result in shadow leakages across the intersections. Figure 3.3 shows the intersection of two surfaces. Polygons ABGH, BCFG, and CDEF are coplanar. Polygon KLMN intersects BCFG. The computed radiosity solution will be incorrect for two reasons:

- While computing the patch radiosity, the form factor of the partially occluded polygon CDGH can not be computed correctly.

- The exposed polygon ABGH will get a high radiosity value, whereas the completely occluded polygon CDEF will not get any radiosity. The exposed part of polygon BCFG is supposed to have the same brightness as ABGH, whereas the occluded part should have the same brightness as CDEF. After extrapolating the vertex radiosities from the adjacent polygons, the polygons are shaded by linear interpolation of the radiosity values of their vertices. This results in polygon BCFG gradually changing from bright to dark rather than having a sharp change at the intersection. This artifact is referred to as *shadow leakage*, because the shadow of the occluded part of CDEF leaks into the exposed part.

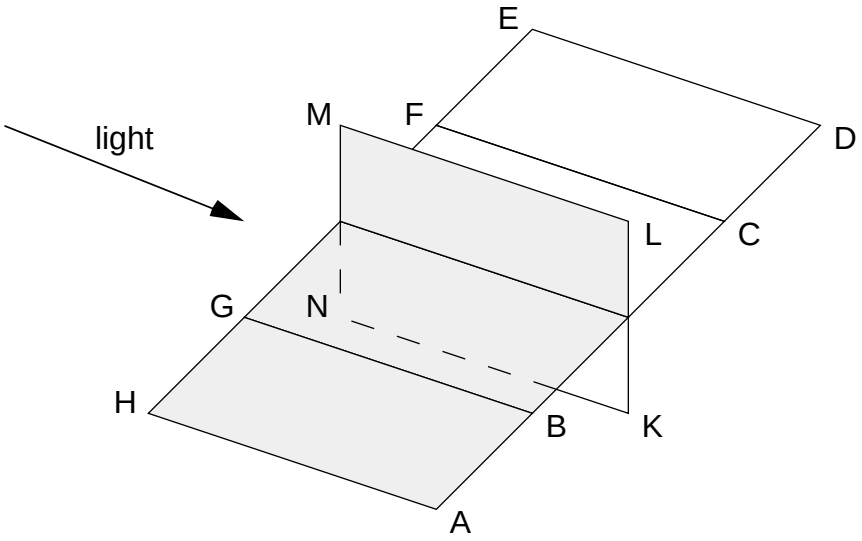

Figure 3.3: Intersecting polygons

Polygon intersection is not the only example of invalid models resulting in shadow leakage. Figure 3.4 shows a common way of modeling two adjacent rooms [WALL91]. If there is only one patch from A to B, then it crosses two illumination discontinuities: from room 1 to the inside of the wall and from the inside of the wall to room 2. The shading across one polygon, however, is always continuous.

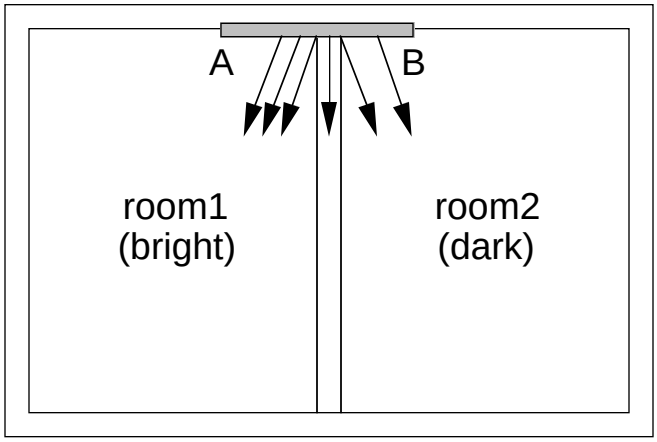

Figure 3.4: Patch across three (!) different zones of illumination

This will lead to a distribution of light from room 1 into room 2 and will be visible as a shadow leakage in room 1. The only way to achieve discontinuities at the junction of the walls is to separate the patches there or to use a modeling approach that does not allow the generation of polygons across intersections with other polygons.

3.2 Mesh Generation Requirements

The Radiosity method is a finite element method where a continuous function is approximated by discrete samples.The continuous function is the light intensity across the surface and the discrete samples are the patches into which the surface is subdivided. A surface can be a polygon of any shape. The subdivision process of the polygons into patches is called mesh generation. A finer subdivision generally leads to a better the approximation of the actual light distribution, but it increases the complexity of the solution and the computational expenses. The mesh density necessary to capture the changes in lighting across the polygon does not have to be constant across the model and generally changes with the lighting situation. An interactive change of the model can change the lighting situation significantly, thus demanding a new subdivision.

Figure 3.5 shows the the object-on-wall problem [WALL91]. The object partially occludes some patches (A) of the floor and the wall. As discussed previously, this will result in shadow leakages from under the box. However, this artifact is not caused by a physically invalid model. The actual lighting function is discontinuous across the border object-wall (Figure 3.6 a), whereas the computed lighting function is not (Figure 3.6 b). One solution to make the lighting function discontinuous across the border object-background is to cut out the occluded parts of the background. This has two serious drawbacks:

- The topology of the model has to be changed. The resulting model is physically valid only if the box actually touches the surfaces, but not if there is a small gap between them.

- If the user moves the object (interactive radiosity), the hole has to be filled again.

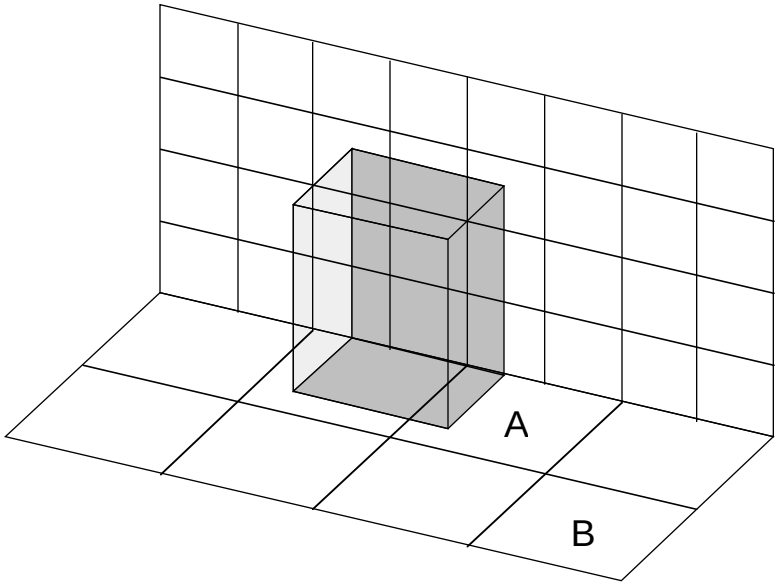

Figure 3.5: The object-on-wall problem

Another approach is to adaptively subdivide the partially occluded patches into smaller parts. This allows more rigorous changes in lighting across the surface, but the radiosity function will always be continuous, thus resulting in shadow leakages. Adaptive subdivision requires that the initial mesh density be fine enough to capture the intensity gradient between the neighbor patches. If the initial mesh is too coarse, the discontinuous energy distribution might fall between receiver nodes , thus leaving the resulting shadow undetected. Therefore, it is necessary to initially subdivide all surfaces into sufficiently small patches.

The advantage of the adaptive subdivision technique is that it preserves the topology of the model. It allows the generation of a refined mesh where the local lighting situation requires it and does not increase the model complexity unnecessarily. Care must be taken, however, to ensure that the meshing process creates a valid mesh. This includes avoiding t-vertices and *slivers*, patches with a small aspect ratio[1].

[1]the aspect ratio is defined as the ratio of the radius of the inscribed circle to the radius of the circumscribed circle of the patch

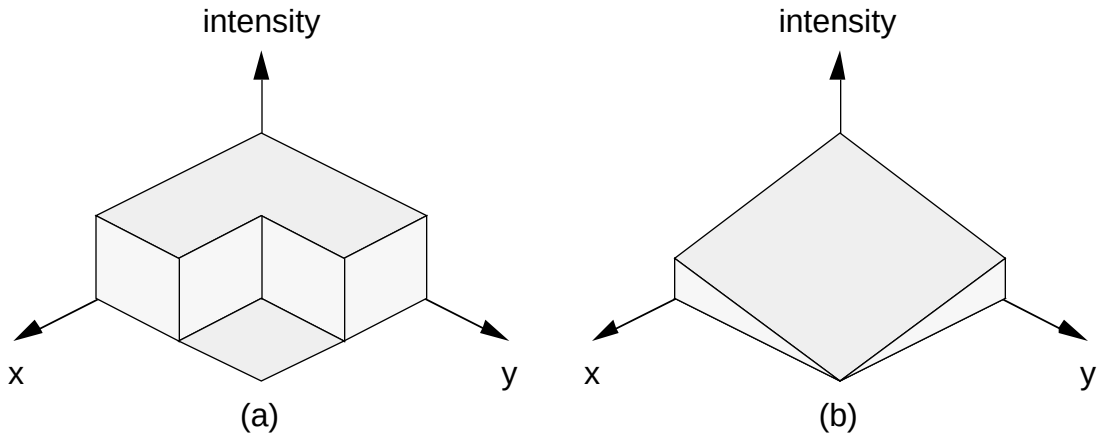

Figure 3.6: Actual and modeled intensity distribution for a partially occluded patch

The generation of t-vertices results in shading discontinuities where they are not desired. A common way of generating complicated surfaces is to compose them of a number of simple shaped polygons. There are two common approaches to surface subdivision: meshing against an external reference (global meshing) or meshing each polygon separately. Figure 3.7 shows the subdivision of convex shape composed of three polygons. Both approaches are unable to avoid t-vertices and will therefore cause shading discontinuities, making the edges of the polygons visible to the observer.

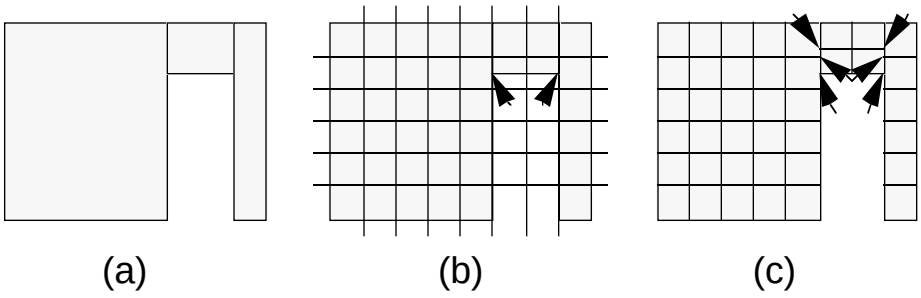

Figure 3.7: Generation of t-vertices by global (b) and local (c) meshing

In order to avoid these shading discontinuities, t-vertices have to be *anchored* [BAEH87] [BAUM91]. This means removing the t-vertex by splitting the adjacent patch (Figure 3.8 a,b).

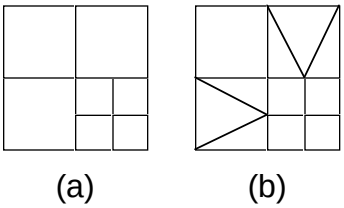

Figure 3.8: Anchoring of t-vertices

The anchoring algorithm requires knowledge of the adjacent patches. This can be obtained easily if both patches are on the same surface, but it becomes complicated if they are boundary quadrants of different surfaces. Discontinuities between different surfaces are allowed as long as the surfaces are not coplanar and of the same material; they are not allowed across coplanar surfaces of the same material. Therefore the input to the subdivision algorithm should consist of *maximally connected planar faces*, so all adjacent coplanar polygons of one material have to be merged prior to the mesh generation.

Most of the problems described in this chapter could be avoided by requiring the user to provide a valid model. However, imposing too many demands on the input data reduces the number of suitable models from existing data sets significantly. Most of these existing models are suitable for ray-tracing solutions, so the additional demands of the radiosity are likely to be seen as a shortcoming of the algorithm rather than the model data.

3.3 Modeling with Virtus WalkThrough

Virtus WalkThrough [VIRT90] is a spatial design tool for the Apple Macintosh. It consists of a modeling tool and a rendering tool, allowing the user to model 3-D scenes, display them on the screen, and move around in the scene. Both modeling and display are graphic-interactive and use the keyboard and mouse as input devices.The rendering is done by flat-shaded polygons. Regarding the quality of WalkThrough as a modeler for radiosity algorithms, the rendering part is of little importance, but it allows the user to get a first impression of how the generated scene will look.

WalkThrough allows the modeling of 3-D scenes made up of polyhedra. Polyhedra are created in a *translational sweep.* They are specified as polygons in a plane and then swept along a trajectory orthogonal to this plane (Figure 3.9). WalkThrough allows the creation of polygons in planes defined by the three axis of the world coordinate system. All

polygons have to be convex and planar. The size of the polygon does not have to stay constant while sweeping. There are options make the object converge linearly or radially thus allowing the design of cones and spheres. These operations usually generate valid solids and provide a fast and easy way of creating solid objects.

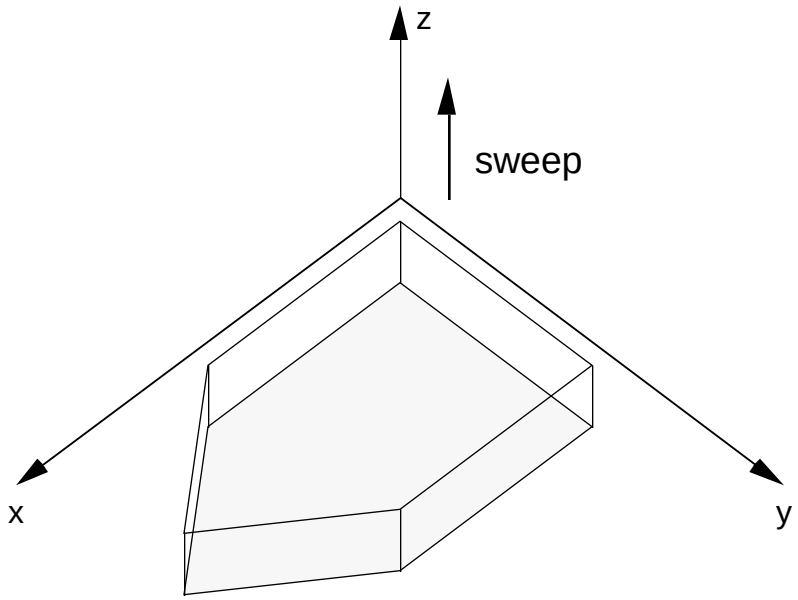

Figure 3.9: Creation of a polyhedron by a translational sweep of a polygon

The internal dataset of the model is a boundary representation, i.e., a solid is stored by representing its boundary by a finite number of surfaces. Each surface is represented by its edges and vertices. This type of internal representation is convenient for radiosity methods, since mesh generation requires surfaces as input. A valid boundary representation of a solid, called an *abstract solid*, has to fulfill two types of conditions, *topological* and *metric* [REQU88]. The topological conditions are:

t1- Each face must have at least three edges.

t2- Each edge must have precisely two vertices.

t3- Each edge must belong to an even number of faces.

t4- Each vertex in a face must belong precisely to two of the face's edges.

Condition t1 ensures that each surface is at least a triangle, condition t3 avoids "dangling" surfaces and condition t4 avoids dangling edges. Fulfilling these requirements does not

necessarily yield a valid solid, in particular theses conditions do not prohibit the self intersection of solids. The following metric conditions must also be satisfied:

s1- Each vertex must define a distinct point in R^3, the world coordinate system.

s2- Edges must either be disjoint or intersect at a common vertex.

s3- Faces must either be disjoint or intersect at a common edge.

These conditions ensure that the resulting solid is not self intersecting and has a finite volume. Conditions s2 and s3 are computationally expensive because they require face to face comparison. These conditions can be satisfied most efficiently by storing the surface representations in a winged-edge structure [BAUM72].

Creating a scene means combining spatial primitives to a model. Virtus uses a strong hierarchy with respect to containment. A primitive can be inside or outside of other primitives, but no intersections are allowed. Explicit boolean set operations such as union or subtraction on primitives are not possible, all geometric relations have to be inferred from the context. One way of changing the shape of an object is provided by the *slice tool*, which allows a planar cut in an arbitrary orientation.

WalkThrough allows the definition of holes in surfaces. The resulting object, however, does not fulfill the requirements of an abstract solid, i.e. it is not physically valid (Figure 3.10 a). The resulting solid most likely violates condition t3, since the new edges created by the cut do not meet any other face.

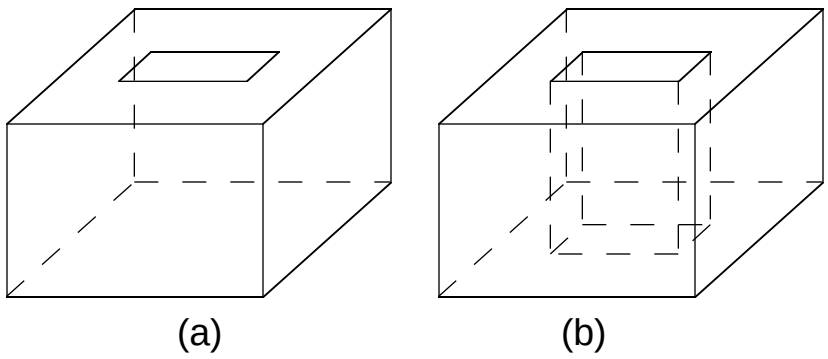

Figure 3.10: Invalid and potentially valid solid

Figure 3.10 b shows a potentially valid solid. Note that it can not be inferred from this wireframe representation if the object is actually an abstract solid. One conceivable set of operations that yields an abstract solid with a hole is:

- Create a solid A
- Cut holes in two faces of this solid
- Create another solid B inside solid A so that two faces of this solid B are coincident with the holes of solid A
- Remove said coincident faces from solid B.

Allowing this cut-operation on polygons rather than on whole solids leaves it in the responsibility of the user to generate an abstract solid. It is extremely difficult and time-consuming to analyze the validity of such a model mathematically.

Another consequence of the internal representation of objects in WalkThrough is that it does not allow the classification of a volume to be inside or outside a solid without taking all surrounding solids into account. This information is necessary, however, to determine the outward facing normal of a surface, which in turn is a prerequisite for the computation of the radiosity solution.

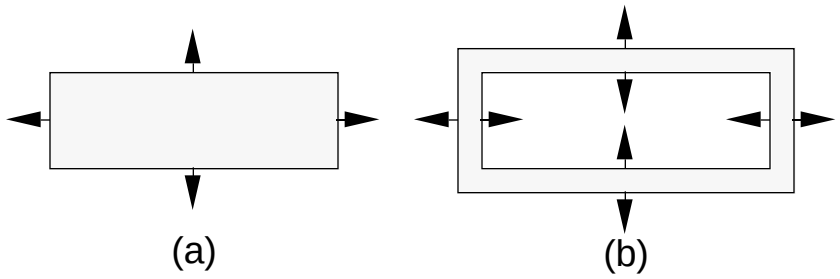

Figure 3.11: Direction of outward facing normals

Figure 3.11 shows the direction of the outward facing normals of a solid. The direction of the normal can not be inferred from one of the polyhedra alone. If the polyhedron is not surrounded by any other polyhedron, the direction of the normals is as shown in Figure 3.11a. If there is one other polyhedron enclosing said polyhedron, the direction of the normals of the enclosed polyhedron is reversed (Figure 3.11 b).

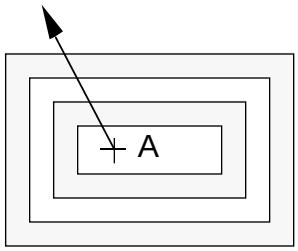

Figure 3.12: Definition of the interior of a box

Figure 3.12 shows the top view of a hollow box contained in a second hollow box. In order to classify a volume to be inside or outside a solid, a three-dimensional version of the *odd-parity rule* can be applied: Choose any point inside the particular volume as a test point. Next, choose a ray that starts at this test point and ends in infinity and that does not go through any vertices or edges. If this ray intersects an odd number of faces, the volume is considered to be interior, otherwise, the volume is exterior. This scheme has two drawbacks, one is the computational expense of computing the number of intersections, the second is the restriction that the odd-parity rule only holds for abstract solids.

This tedious way of determining the normals is not caused by the fact that one polyhedron is allowed to contain other polyhedra, it is the lack of explicit set operations on polyhedra that is responsible for these ambiguities. WalkThrough allows the combination of polyhedra without defining how they have to be combined (subtract volume A from B, for instance). This means all geometric relationships have to be inferred from the context, resulting in complex computations and allowing invalid or ambiguous models.

The intention to display the model interactively makes it infeasible to compute the orientation of all surfaces “on the fly”. Therefore, a heuristic rule for the orientation of the surfaces has to be employed: A priori, all surfaces are supposed to face outward. The direction of the normal is defined by the order of the vertices. Outside is the side of the polygon from where the listed vertices can be seen in a counterclockwise order. If a surface is supposed to have opposite orientation, the user has to specify this explicitly by assigning a special tag to this surface during the design process. This requires diligence from the user, since an inconsistent definition will result in an invalid model. [RICH88] discusses automated processing of the model to make this user interaction unnecessary.

Since the methods of the modeler alone do not guarantee the generation of abstract solids, the user and the preprocessing software for the radiosity solution have to ensure that the final model meets all constraints.

In the proposed pipeline, the user has to ensure the following properties:

- No coincident polygons that are visible to the observer.

- The defined surfaces must comprise the solid and hence define unambiguously what is inside (boundary determinism). Care has to be taken when holes are inserted.

- All surface normal are supposed to point outwards, inwards pointing normals have to be tagged accordingly.

The preprocessing ensures that

- The size of the patches will be small enough to capture shadows and lighting inhomogeneities to a specified level of detail.

- There will be no t-vertices generated by the subdivision.

The problems mentioned above are not shortcomings of WalkThrough, they are shortcomings related to the use of WalkThrough as a modeler for lighting models that require physically valid models. WalkThrough is a combination of modeler and renderer, hence the modeler is optimized to fulfill the requirements of the built-in renderer. Radiosity methods, however, impose different constraints on the model and therefore require a special preprocessing of the model.

Among the positive features of WalkThrough are its intuitive user interface and the ease with which models can be created. All modeling processes are fully graphic-interactive and do not require experience in the use of CAD-systems. The user interface is similar to MacDraw and other Macintosh drawing programs and hence is familiar to most Macintosh users. The integrated renderer provides an adequate three-dimensional view of the generated model without requiring a special graphics workstation or complex rendering computations. WalkThrough includes a library of predefined three-dimensional objects that can be scaled and included in the scene. Together, the integrated library and the intuitive modeling methods allow the user to generate scenes of moderate complexity quite rapidly. However, the lack of tools such as user-defined coordinate systems, the small number of drawing primitives, and the restriction to orthogonal drawing planes aligned with the axes of the world coordinate system make it difficult to design complex scenes. As for the physical validity of the models, a CSG-based modeling system would probably perform

much better, but more complex preprocessing steps are required to convert the CSG database into a boundary representation suitable for the radiosity solution. Moreover, CSG systems typically allow modelling primitives such as spheres, cylinders and cones, which have to be approximated.

Chapter 4

Mesh generation

Once an abstract solid, the valid geometric representation of a physical object, has been created, it has to be converted into a representation suitable for input to the radiosity algorithm. The radiosity method is a finite element method where a continuous function is approximated by discrete samples. The continuous function is the light intensity across the surface, and the discrete samples are the patches into which the surface is subdivided. A surface can be a polygon of any shape. Generally, radiosity solutions require convex quadrilaterals or triangles as input. The process of subdividing the polygons into such triangular or quadrilateral patches is called mesh generation. It can be done manually or automatically. If the mesh generation is fully automatic, it becomes feasible to include the mesh generation-radiosity computation process in a feedback loop in which the mesh geometry can be automatically adjusted to optimize the radiosity solution. In this case, the mesh generation is used for the *initial generation of the mesh* as well as for the the *adaptive mesh refinement* that ensures the accurate radiosity solution. The adaptive refinement method changes the density of the mesh on the basis of prior results of the radiosity solution or updates the mesh because the lighting situation has changed. Since the radiosity method is basically a finite element method (FEM), subdivision algorithms used for FEM-methods can be adapted for the preprocessing of radiosity input. This chapter gives an overview of existing approaches to mesh generation in FEM methods, evaluates these methods with respect to the specific requirements of radiosity methods, and introduces a new approach to mesh generation well suited for radiosity methods. The proposed mesh generator for interactive radiosity methods uses a spatial decomposition approach and is based on algorithms used for the FEM mesh generation by Yerry and Shephard [YERR83] and Baehmann et al.[BAEH88].

4.1 Automated Mesh Generation for Finite Element Methods

The difficulties in generating valid finite element models have been among the major barriers to the common use of finite element methods. The manual preprocessing of a complex model requires a considerable amount of time and a user well familiar with the constraints that the FEM algorithm imposes on the input model. Therefore, much research has been done in order to reduce or eliminate the user interaction necessary to preprocess the input model. Early approaches to the generation of finite element meshes are commonly referred to as *mapped mesh generators*. They require the input model to be composed of a set of mappable regions of individual, fixed topology mesh patches and are generally able to produce well controlled meshes within these regions. However, partitioning the model into a set of patches of the required topology requires user interaction. Moreover, the spatial decomposition of surfaces of common material into a set of individual subdomains can lead to t-vertices between these adjacent subdomains. These meshes are referred to as *nonconforming* in FEM terminology. As discussed in chapter 3, discontinuities across boundaries can lead to display artifacts during the rendering process.

Due to these difficulties and shortcomings, automatic mesh generation techniques, capable of meshing domains of arbitrary shape without user interaction, have been considered. Shepard [SHEP88] and Ho-Le[HOLE88] give surveys on automatic mesh generation algorithms. There are numerous ways [YERR83] [SHEP88] [Ho-LE88] of categorizing the different approaches. Shepard [SHEP88] deals with algorithms that are either three-dimensional or for which the possibility of an extension to three dimensions is obvious. He classifies these algorithms as being based on one or more of the following approaches:

- Point placement followed by triangulation
- Removal of individual subdomains
- Recursive subdivision of the domain
- Spatial decomposition followed by subdomain meshing.

4.1.1 Point Placement followed by Triangulation

In the approach of point placement followed by triangulation, the generation of the tiles is done in two distinct steps. The first is to place points throughout the area of interest. In the second step, these points are triangulated into a mesh. The initial placement of points

largely influences the quality of the mesh. There are several approaches to ensuring the optimal placement of these initial points. One way is to place a grid over the polygon and points may be placed at regular or irregular locations within the regions defined by this grid. Frey [FREY 86] gives an overview of the different approaches to point placement. Once these initial points have been set, they have to be connected to form a mesh. *Delaunay triangulation* is a well known technique of triangulating a set of points.

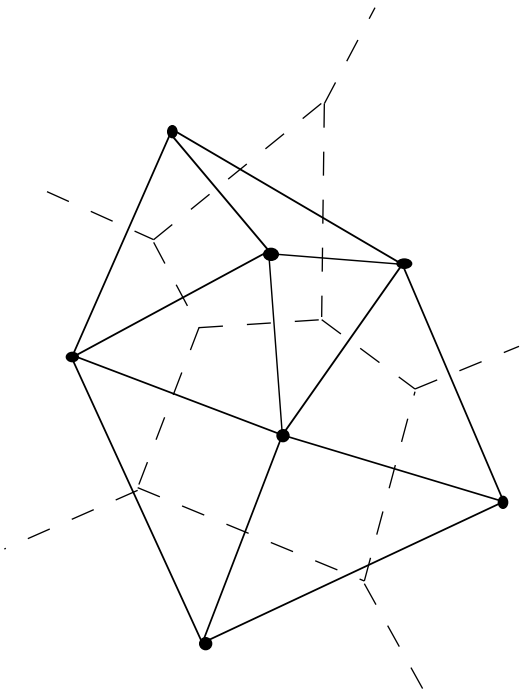

Figure 4.1: Delaunay triangulation (solid) and Voronoi diagram (dashed)

The construction of the *Voronoi diagram* is convenient way of computing the Delaunay triangulation of a set of points P in a plane. A Voronoi diagram is a subdivision of a plane into polygons, each associated with a point $p \in P$, such that every point in its region is closer to p than to any other point of P. The Delaunay triangulation of the region is computed by connecting a pair of points if they share a Voronoi boundary. Figure 4.1 shows the Voronoi diagram (dashed lines) and the corresponding Delaunay triangulation (solid lines) for a given set of points. Shamos [SHAM85] and Edelsbrunner [EDEL87] provide detailed treatments of this technique. The Delaunay triangulation has numerous properties which make it useful for mesh generation purposes, in particular, there are no internal points in the circle defined by the three vertices of a triangle. This insures the

generation of a set of triangles that is as equilateral as possible. The refinement of the mesh is done by inserting new vertices in the interior and updating the mesh [CAVE74], [BANK83], [BYKA83], [FREY87].

Unfortunately, Delaunay triangulation does not handle concave regions or regions with holes. This problem is due to the fact that Delaunay triangulation is based on points rather than on edges. There is no guarantee that the edges which define holes will be represented in the final triangulation. *Constrained Delaunay triangulation* [FLOR88] addresses this problem by generating a mesh that is as close as possible to a Delaunay Triangulation given that certain user-specified edges must be included in the mesh.

Chen [CHEN88] proposes an algorithm that, given an input with no two data points closer that h and all edges between h and ($\sqrt{3}$)h (h being a user-defined parameter), guarantees a triangulation with the following properties:

- All edges of T have length between h and 2h
- All angles of T are between 30° and 120°.

The problem is to guarantee the required distance of the input points between h and ($\sqrt{3}$)h. Furthermore, the guaranteed quality of the mesh is not extensible to three dimensions.

4.1.2 Removal of Individual Subdomains

Mesh generation based on sub-domain removal operates by splitting tiles from the region one at a time until the domain is reduced to a single tile. There are also schemes available which initially subdivide a complex region into simple parts and triangulate these regions by a different technique (such as the above mentioned method). This may help to handle complex shapes such as concave polygons or polygons with holes. The criteria on which the algorithm relies in choosing which subdomain to remove next are generally complex, making the generation of fine meshes computationally expensive. Shepard [SHEP88] proposes a sub-domain removal algorithm for two-dimensional regions that subdivides a complex region into simple parts that are suitable as input to a computationally efficient mapped mesh generation (Figure 4.2).

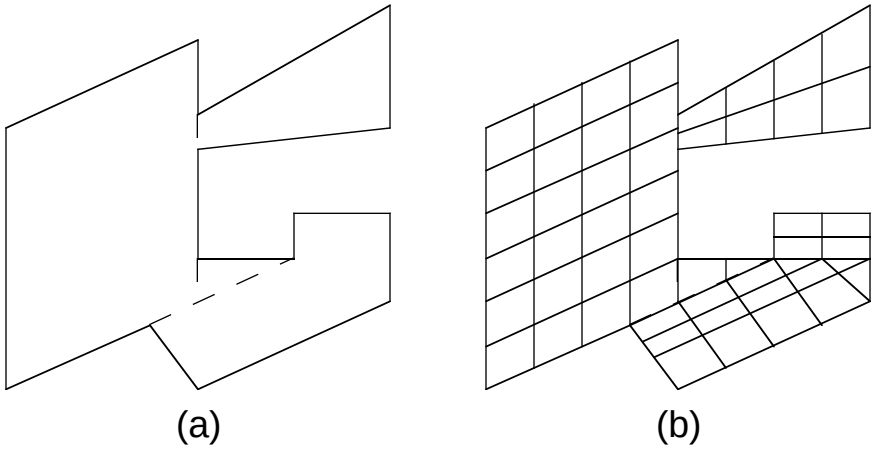

Figure 4.2: Subdomain removal and meshing by a mapped mesh generator (adapted from [SHEP88])

Care must be taken, however, not to generate t-vertices at the border between two adjacent, individually meshed regions, since this would result in nonconforming meshes.

4.1.3 Recursive Subdivision

The recursive subdivision algorithm repeatedly splits a region into smaller subregions, until it is either a tile of acceptable shape and size, or of such simple shape that valid tiles can be generated easily. This approach is similar to the above sub-domain removal algorithm and it shares its main disadvantage, i.e. the computationally complex process of deciding which subdomain to subdivide next.

4.1.4 Spatial Decomposition Followed by Subdomain Meshing

Spatial decomposition followed by subdomain meshing is a two-step approach. A detailed explanation of the implementation of this algorithm is given in the following chapter. First the initial domain is subdivided into simple cells and then these cells are meshed individually. The initial decomposition is typically done by a quadtree in the two dimensional case or an octree in the three-dimensional case. The approach used in this thesis work is an example of such a modified quadtree technique in two dimensions. A single region of interest (a polygon of convex or concave shape, with or without holes) is

subdivided into the quadrants of a quadtree with the quadrant size defining the density of the mesh. This is done by taking each edge of the polygon (or hole) and recursively subdividing it until each fragment fits within a quadrant of the tree. A quadrant is allowed to be only one level finer then the adjacent quadrants. This leads to a certain homogeneity in the mesh. Each quadrant is classified as either an interior, exterior or boundary region. There are different meshing rules assigned to each type of quadrant. Quadrants classified as interior or boundary regions are meshed according to these rules, whereas exterior regions are simply discarded. Since interior quadrants are of simple shape, the quality of the interior elements is very convenient. During the patch generation process, quadrants can use the tree structure in order to obtain information about their neighbors, thus avoiding the generation of discontinuities across adjacent elements. The meshing of the boundary regions requires some additional work in order to avoid slivers - elements with small interior angles - which lead to errors due to computational artifacts in the radiosity solution.

4.2 Evaluation of the Mesh Generation Approaches

Although the radiosity method is basically a finite element method, it imposes additional constraints on the mesh generation. Common criteria for the evaluation of a mesh generation algorithm for finite element methods are:

- Element type
- Element shape
- Mesh density control
- Time efficiency.

With respect to the element type, finite element methods are far more demanding than radiosity algorithms. Not just any element type will do, some algorithms require quadrilaterals, some even need brick-shaped elements [HUGU87]. As for radiosity methods, both triangles and convex quadrilaterals represent valid input.

The element shape is of importance for the radiosity method as well, since poorly shaped elements (slivers) lead to both radiosity and display artifacts. Delaunay triangulation guarantees triangles as well shaped as possible for a set of given nodes, thus leading to good element shape. Spatial decomposition produces excellent interior elements, but boundary elements tend to be problematic [KEJA86]. Ho-Le [HOLE88] maintains that subdomain removal schemes may result in poor element shapes.

As for mesh density control, interactive radiosity methods impose special constraints on the mesh generation. In most finite element methods, an initially coarse mesh is further refined based on preliminary results of the finite element analysis. This approach is used for *adaptive refinement* techniques in radiosity methods as well. In this case, adaptive refinement is the adaption of the mesh density to the current lighting situation. If the radiosity gradient along adjacent elements is too high, the mesh has to be refined. Meshes with varying density are referred to as *graded meshes* in finite element applications. However, if the lighting or the model geometry is allowed to change over time (as in interactive radiosity methods), the computed mesh has to be adapted to the new lighting situation. Shadow boundaries that require high mesh density might have changed location (Figure 4.3). In this case, the mesh has to be refined at other locations in order to reflect the new lighting situation. If the mesh generator is allowed only to make the mesh finer, this will lead to representations with more and more patches as time goes by, and the mesh density will be high even at regions where such density is not required by the current lighting situation. Since the time for the radiosity solution increases with the number of patches, one is wise to avoid unnecessarily high numbers of patches. In summary, a mesh generation scheme suitable for interactive radiosity methods should allow the increase of the mesh density, where lighting inhomogenieties require smaller patches. It might be beneficial to be able to decrease the mesh density in areas where the changed lighting situation has led to a more homogeneous light distribution. However, there is currently no data to show if the resulting reduced number of patches reduces the radiosity time to such an extent that it justifies the time required for changing and rebalancing the mesh.

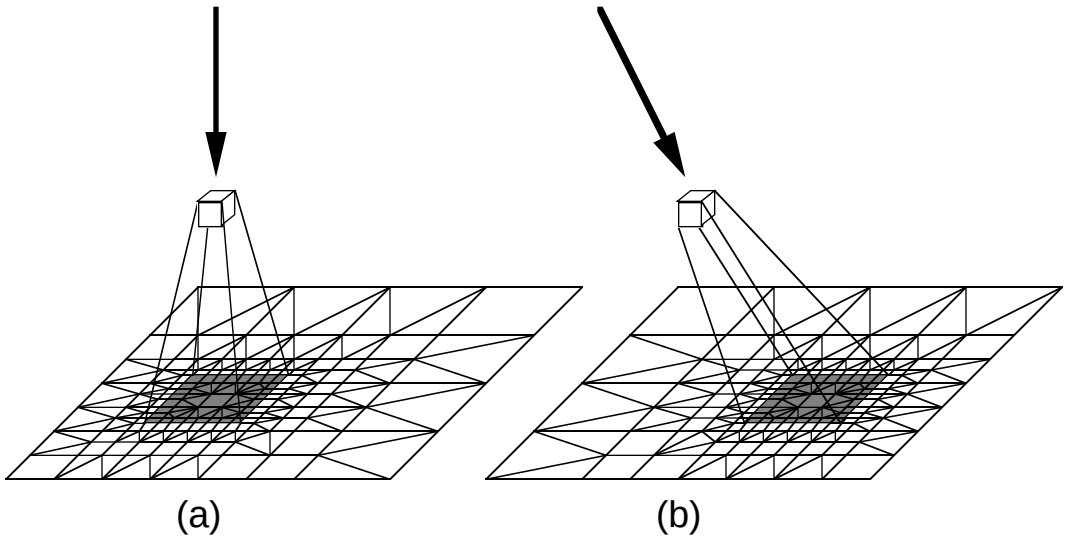

Figure 4.3: Mesh subdivision dependent on the current lighting situation

There is no obvious way to implement mesh density control in the subdomain removal algorithm. Triangulation schemes allow control of the mesh density by inserting new points in the surface and subsequent remeshing of the region around this point [FLOR84]. Unfortunately, there are no algorithms for the reverse process, i.e. obtaining a coarse mesh from a fine mesh while preserving a certain degree of continuity. The spatial decomposition approach, when implemented in form of a quadtree or octree, is well suited for adaptive meshing. Quadrant subdivision (descending one level in the tree) leads to an increased density of the mesh, whereas joining four quadrants (ascending one tree level) decreases the mesh density.

The time efficiency of the algorithms is of particular importance for “on-the-fly” mesh generation. Chew [CHEW89] proposes an O(n) algorithm for his Constrained Delaunay triangulation approach, where n is the number of data points.Wordenweber’s subdomain-removal-based algorithm [WORD90] runs in $O(n^2)$ time. By exploiting the spatial addressing capability of the tree structure, Baehmann et al. [BAEH87] observe O(n) time for their quadtree based spatial decomposition algorithm.

In order to obtain information of the speed on a per-element basis, the interaction of the mesh generator with the geometric database has to be considered. Mesh generation algorithms require either *geometric interrogation* or *geometric interrogation and modification*. Geometric interrogation consists merely of data retrieval, whereas geometric modification results in changes of the database. Since reading from a database is almost always computationally less expensive than changing it, mesh generators that read and modify a model database are typically less efficient, on a per-element basis [SHEP88]. Delaunay triangulation and spatial decomposition schemes require geometric interrogation only. Subdomain removal algorithms require both interrogation and modification, because they have to read the model geometry, split up a suitable subdomain, and store the updated model without the subdomain.

Another important consideration for adaptive mesh generation is the accessibility of the patches. Since element subdivision is based on the radiosity gradient across adjacent elements, a fast identification of adjacent elements is desirable. Spatial decomposition approaches based on quadtrees or octrees provide a powerful data structure for a fast retrieval of patches based on their geometrical position, whereas the other approaches generally lack such a description.

The above evaluation indicates that approaches based on Constrained Delaunay triangulation or spatial decomposition methods are suitable tools for the generation of well behaved meshes for radiosity methods. Both are able to produce meshes in linear time, they do not require geometric modification of the database and they are able to produce graded meshes. Among the advantages of Constrained Delaunay triangulation is the quality-guaranteed shape of the patches, although it is not obvious how to satisfy the preconditions for this guarantee. Spatial decomposition methods are especially well suited for interactive radiosity solutions, because their data structure allows an efficient adaptive refinement of the existing mesh. Since this thesis work is primarily concerned with the mesh generation for interactive radiosity methods, a mesh generator based on a spatial decomposition approach seems to be a promising choice.

Chapter 5

A Modified Quadtree Algorithm for Mesh Generation

This chapter describes the algorithms employed in the modified quadtree mesh generator. The mesh generation process consists of two steps: the first is the spatial decomposition into subdomains, the second is a subdomain meshing by removal of individual patches. The modified quadtree is used to decompose the object into disjoint quadrants. Quadtree based approaches have been used for mesh generation first by Yerry and Shepard [YERR82] and improved by Kela et al.[KELA86] and Baehmann et al.[BAEH87]. Samet [SAME90] provides a detailed treatment of the properties of quadtrees and octrees as spatial data structures.

The design of the mesh generator is strongly dependent on the type of data to be processed. Most models for radiosity methods consist of boundary representations, where the boundaries are presented as planar polygons. Each surface is processed separately. As discussed in chapter 3, this can lead to various artifacts if there is no continuity across adjacent coplanar polygons of the same material. In order to avoid a nonconforming mesh, these polygons have to be merged into one surface and processed together. As a consequence, the mesh generator has to be able to handle concave surfaces and holes in surfaces. Also, since the mesh generation is designed to work with an interactive radiosity method, it has to be capable of adapting the mesh to the varying lighting conditions in a time efficient way.

Since a quadtree is a two-dimensional data structure, the first step in mesh generation consists of a transformation of the polygons into a common plane where the decomposition takes place. The polygon is scaled so that it fits in a region of predefined size. This region represents the root quadrant of the quadtree. It is subdivided into four quadrants of equal size, each being a child of the above root quadrant. These child quadrants can in turn be subdivided, until all quadrants have the desired size. The tree has to be balanced in order to

avoid high gradients in mesh density. A quadrant can be either inside, outside, or on the boundary of a polygon. The quadrants flagged as outside are simply discarded, because they are of no importance for the mesh. All the interior quadrants can be meshed easily, since they are of rectangular shape. Boundary quadrants are problematic; since the shape of the interior regions in such quadrants are dependent on the specific boundaries stored in those quadrants, they can have arbitrary shapes.

Early mesh generation approaches for finite element analysis [SHEP83] approximated the interior regions of these boundary quadrants by template quadrants of predefined shape, or they changed the stored boundary description of the surface [BAEH87]. This is a legitimate procedure for finite element methods, since these methods are primarily concerned with the shape of the object rather than the precise geometric size. A vertex may be moved to a more suitable position, if the movement is small compared to the size of the object. Mesh generation algorithms for radiosity methods, however, have to preserve the exact outline of the model, even if this results in poorly shaped patches. Each surface is part of a complex model, and changing its geometry might result in display artifacts such as visible cracks between two adjacent surfaces.

The mesh generator input consists of a non self-intersecting polygon of concave or convex shape and an arbitrary number of holes. The holes have to be contained in the polygon and are not allowed to intersect each other or the polygon. The polygon and its holes are represented by lists of three-dimensional vertices, arranged in counterclockwise order for the polygon and in clockwise order for each of the holes. Each hole is associated with a polygon. This provides the distinction between front and backface. Also, each polygon has a number of features associated with it, including data for the RGB color of the front and backface, texture information, and emitter information.

5.1 Setup of the Tree

The first step in the mesh generation consists of transforming the input polygon and holes into the x-y plane and aligning the longest edge of the polygon with the x-axis. The mesh generator cannot process arbitrarily large polygons; the polygons must be smaller than some predefined maximum size. This size is dependent on the maximum depth of the tree. The size of the quadrant edges in the tree is expressed in integers, hence the smallest quadrant has an edge length of 1. If the tree depth is 2, the root quadrant has edge length

2^2. Generally, the edge length of the root quadrant (tree level 0) equals 2^β, with β the maximum depth of the tree. The size σ of a quadrant at tree level λ is defined as

$$\sigma = 2^{(\beta - \lambda)} \qquad (5.1)$$

Input polygons have to be scaled so that they fit in a bounding box the size of the root quadrant. To accomplish this, first the extents of the transformed surface along the x and y axis are determined. Next the image is scaled and translated with different factors for each axis. The direction with the longer extent is mapped to the length of the root quadrant. The shorter extent is initially scaled by the same factor, but then the next integer that is a power of two has to be found. The scaling factor for the axis with the shorter extent is this power of two divided by the shorter axis. Consider a rectangle with extents $x_{min} = 3$, $x_{max} = 35$, $y_{min} = 2$, $y_{max} = 14$, and let the size of the root quadrant be 64. The extent in x-direction is 35 - 3 = 32, this yields a scaling factor of $scale_x = 64 / 32 = 2$. The extent in y-direction is 12, scaling gives 12 * 2 = 24. The next power of 2 is 32, hence the scaling factor for y is $scale_y = 32 / 24 = 4 / 3$. Figure 5.1 shows the scaling.

The advantage of this scaling procedure is that rectangles, by far the most common input, can be meshed without producing boundary quadrants in the tree structure and the resulting interior quadrants are as equilateral as possible after re-transformation. Note that mapping both axes to the size of the root quadrant would avoid problems in boundary quadrants. However, it might lead to badly distorted patches after retransformation if the initial surface has a bad length to width ratio, that is, scaling factors for both axis are very different. The above algorithm ensures that that length to width ratio is between 0.5 and 2.

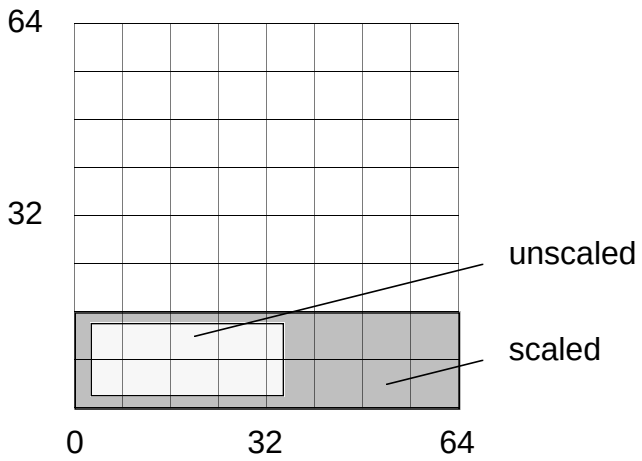

Figure 5.1: Scaling of the input polygon

In order to provide sufficient receiver node density for capturing shadow boundaries, an initial mesh of sufficient patch density has to be defined by the user. Given a patch size, the corresponding tree depth is

$$\lambda = \min (\beta , \text{ceiling} (\log_2 (e_{max} / \text{patchsize}))) \qquad (5.2)$$

with e_{max} being the edge length of the longer axis. The tree depth cannot be higher that the allowed maximum level of the tree β.

The next step is to create an initial quadtree representation of the domain by subdividing the tree down to the previously derived depth parameter λ. This will guarantee the required maximum patch size. At this level, no spatial information is stored in the tree.

Now the polygon and hole edges have to be stored in the tree. The tree structure used in this approach is similar to the PM quadtrees proposed by Samet and Webber [SAME85]. Developed for the storage of polygonal maps, this representation satisfies the following criteria:

- It stores polygons without loss of information. This is essential, since the mesh generator has to preserve the exact geometry of the polygon.

- It can be efficiently manipulated, i.e., it can be expanded or pruned, and stored elements can be retrieved easily. This is important for interactive collaboration with radiosity methods.

The following C-code shows the implementation of the data structure for the quadtree:

```
typedef struct _QTREE {
    char   level;
    enum {node,leaf_undefined,leaf_out,leaf_boundary,leaf_in}
status;
    struct _QTREE *qtree[4];
    LINE *line;
    LONG_VERT_2D center;
} QTREE;
```

The tree consists of nodes and leaves. Each element can be either node or leaf, with each node having four children (Leaves are the terminal elements of the tree). A node contains the location of its center and level, this information could be inferred from the position in

the tree, but it is included for time efficiency. Line contains a linked list of lines assigned to a leaf, and each line is stored in form of two vertices representing its start and end points. Lines can only be stored in leaves.

5.2 Inserting Edges in the Quadtree

Surface edges (the terms line and edge are used interchangeably in this thesis) are stored in the tree according to a set of *decomposition rules*:

- A leaf can have the status outside, inside or boundary.

- Lines can only be stored in boundary leaves.

- A vertex can only be stored if it lies inside the extents of a leaf.

- A leaf can contain only one distinct vertex.

- If a region contains a vertex, it can only store lines that are connected to this vertex. (Since an abstract solid permits only two lines to share a common vertex, at most two lines can be stored in a quadrant.)

- If a region contains no vertex, it can only store one line.

The algorithm, that inserts a line in the tree, starts at the root quadrant of the tree and is based on the *divide-and-conquer* strategy. The procedure is the same for all quadrants in the tree. As already mentioned, lines can only be stored at terminal quadrants (leaves). If the quadrant under observation is not a terminal quadrant, the line has to be subdivided. Figure 5.2 shows the subdivision procedure. The line a-b is split into line segments a-c, c-d, and d-b and these line segments are given to the respective quadrants. If these quadrants are nodes, the whole procedure is invoked recursively. If they are terminal quadrants, two actions are possible:

- Inserting the edge does not violate the decomposition rules. The edge is stored at this quadrant and the quadrant is tagged as *boundary*.

- Inserting the edge violates one of the above rules. The quadrant has to be further subdivided. This edge, and all other other edges that had been stored in this leaf previously, have to be distributed to the new quadrants. The status of the quadrant is changed from leaf to node.

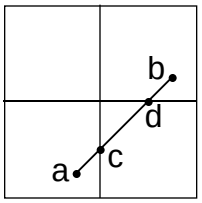

Figure 5.2: Inserting a line into the tree

The maximum tree depth β imposes a limit on the number of subdivision processes. If two vertices are so close that they cannot be distributed in two different quadrants after expanding the tree to its maximum level, the current tree structure is not capable of storing this polygon without loss of information. There are two ways to solve this problem; one is to increase the maximum depth of the tree. Changing the maximum depth of the tree, however, requires a new setup of the tree, since the depth β determines the size of the initial quadrant. Another solution would be to merge these two points, if they are vertices of a common line, but this leads to an undesired change in the polygon's geometry. Since changing the geometry is not allowed, the problem has to be solved by defining a new tree with increased depth.

Figure 5.3 (a) shows a concave surface with a hole. Meshing this surface with the above rules yields the quadtree representation in Figure 5.3 (b); the size of the quadrants is dependent on the predefined maximum patch size.

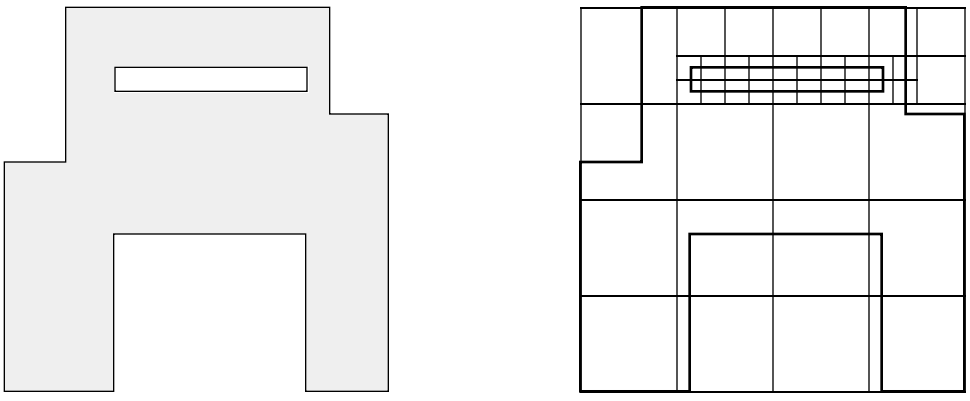

Figure 5.3: Concave surface with hole (a) and corresponding quadtree representation (b)

Now all of the quadrants have to be classified as *interior*, *exterior* or *boundary*. New quadrants are flagged as *undefined*. Since boundary quadrants have been flagged during the line insertion process, undefined quadrants can be either exterior or interior. The input data for the mesh generation consisted of vertices arranged in counterclockwise order for the polygon and in clockwise order for the hole. Therefore, seen in direction of the edge, the region to the right side of the edge is interior and the left side is exterior. Using this knowledge, every boundary quadrant is visited and the direction associated with each edge allows the mesh generator to determine which sides of the quadrant are inside or partly inside the surface. Next, the quadrants adjacent to these sides are examined. If they are undefined, their status is changed to interior. This process goes on recursively until an already defined quadrant is encountered. Figure 5.4 (a) shows the quadtree after defining the interior, exterior and boundary patches.

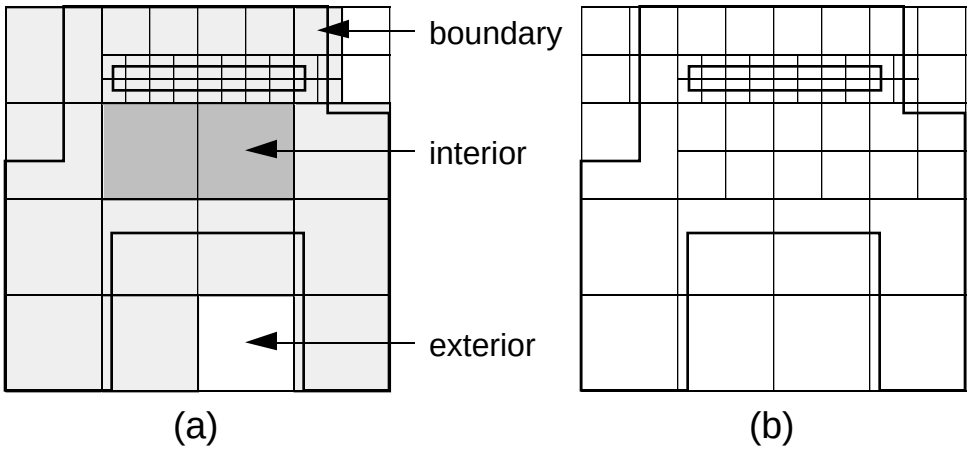

Figure 5.4: Determining the interior (a) and balancing the mesh (b)

In order to assure the generation of well shaped meshes, the quadtree structure has to be balanced, that is, only one tree level difference is allowed between quadrants belonging to either boundary or interior classifications. This is done by traversing the whole tree β times. Initially, all elements of level β have to be visited, and all of the neighbors of each element must be refined to at least level β-1. This same procedure is applied to all quadrants of level β-1, that is, their neighbor quadrants have to have at least level β-2 refinement, and so forth. Neighbor quadrants in this procedure are adjacent quadrants that share a common boundary. Figure 5.4 (b) shows the balanced tree.

Upon completion of the tree balancing, the element mesh is created. Every boundary and interior quadrant has to be visited and meshed. Since there are only a limited number of different interior quadrants, time-efficient template meshing can be employed. Boundary quadrants, however, can have an infinite number of shapes, and therefore a subdomain removal procedure is used to mesh these quadrants.

5.3 Processing Interior Quadrants

When processing the interior quadrants, care must be taken not to generate t-vertices. T-vertices are caused by adjacent quadrants which do not share the same level of refinement. These vertices have to be anchored. Anchoring a t-vertex is done by generating a vertex in the adjacent quadrant in the same position. Figure 5.5 shows this anchoring process.

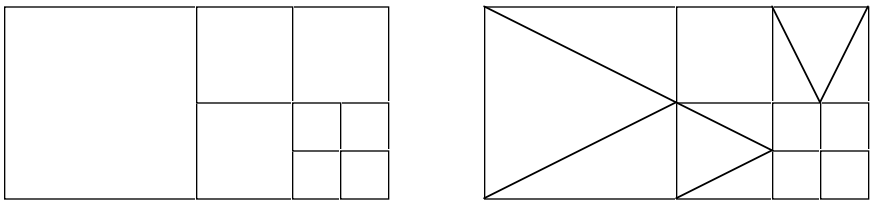

Figure 5.5: Anchoring t-vertices in interior quadrants

Depending on the number and position of vertices to be anchored, one of 16 different templates may be required. By taking advantage of symmetry, this number can be reduced to 6. Figure 5.6 shows these templates. Using only these six templates allows the mesh generator to mesh the internal quadrants in a time-efficient manner.

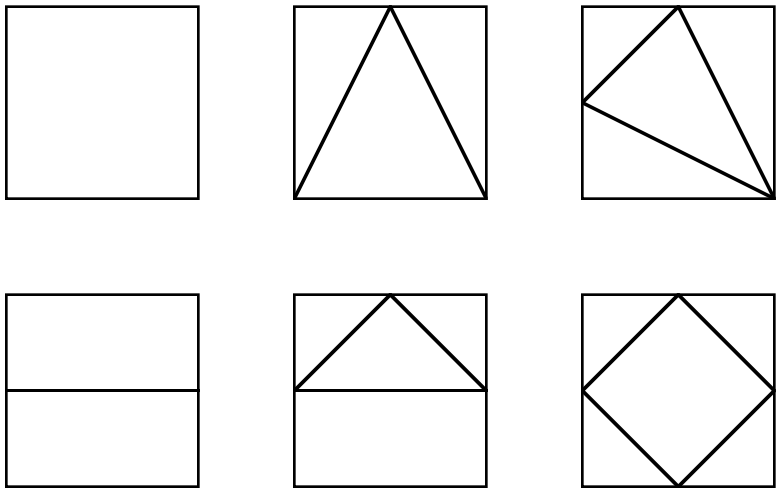

Figure 5.6: Mesh templates for interior quadrants

5.4 Processing Boundary Quadrants

The task of meshing the boundary quadrants is done by subdomain removal. A boundary quadrant can contain one or two edges, can have up to three t-vertices that need to be anchored, and, by definition, consists of a region not belonging to the surface (outside) and another region belonging to the surface (inside). The inside part of the quadrant is determined by performing a loop over the edges stored in the quadrant and the quadrant border that is located to the inside of the stored edges. The subdomain removal is done according to the following rules:

- A suitable subdomain is either a triangle or a quadrilateral of convex shape
- Removing quadrilaterals is preferred to removing triangles
- All removed subdomains have one common vertex, the *hinge vertex*
- If there are two edges stored in the quadrant, their shared vertex is the hinge vertex
- If only one line is stored, the end vertex of this line is the common vertex
- Anchoring t-vertices has precedence over the above rules.

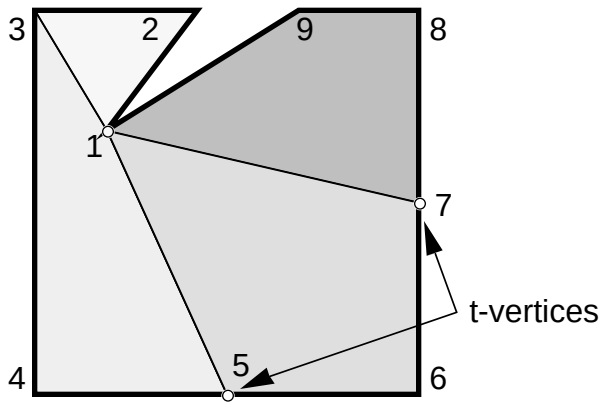

Figure 5.7: Meshing a quadrant by subdomain removal

Figure 5.7 shows a quadrant meshed according to the above rules. Let Vertices 5 and 7 be t-vertices caused by adjacent quadrants. Since there are two lines stored in the quadrant, point 1 is the hinge. Quadrilateral 1234 is concave, therefore triangle 123 is removed first. Quadrilaterals 1345 and 1567 are convex and can be removed next. The remaining polygon 1789 is also convex. Hence the region is meshed in four convex quadrilaterals and one convex triangle. Note that quadrilateral 1568 is also convex, but anchoring the t-vertex 7 has priority. Figure 5.8 shows the example surface before (a) and after (b) anchoring the t-vertices.

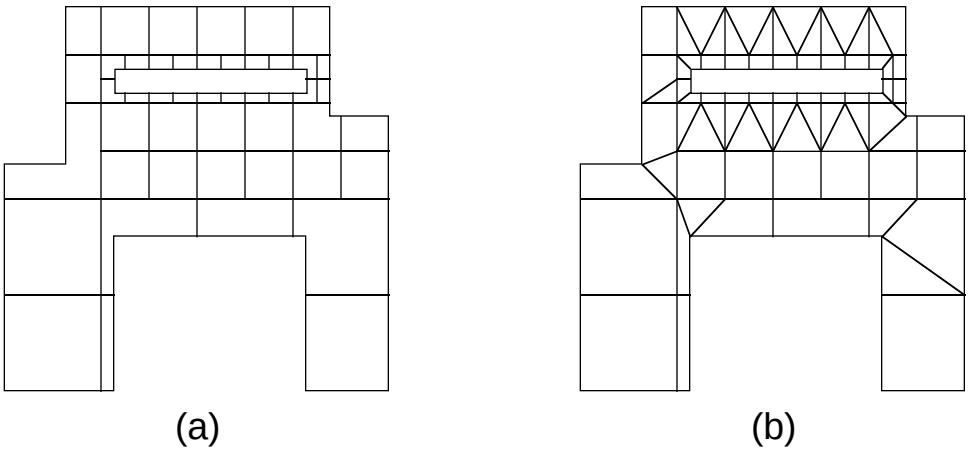

Figure 5.8: Element mesh before (a) and after (b) processing the quadrants

Although the shape of the resulting interior patches is quite good, the shape of some boundary patches might be far from satisfactory. Patches of poor shape, i.e. with one small interior angle, are generated by

- Edges intersecting a quadrant boundary close to a quadrant corner (Figure 5.9 (a))
- Interior vertices located close to the boundary of a quadrant (Figure 5.9 (b)).

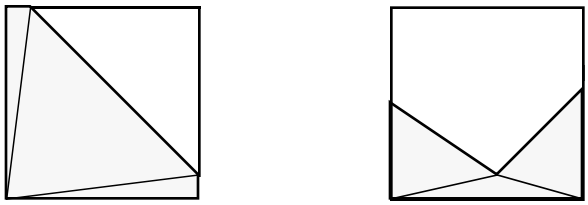

Figure 5.9: Boundary patches with a small interior angle

Baehman et al. address this problem by moving boundary intersection points to quadrant corners and pulling internal vertices to quadrant boundaries [BAEH87]. This results in better shaped patches, but it changes the model geometry. Since these changes are small compared to the model geometry, this is acceptable for finite element meshes. In mesh generation for radiosity methods, however, trading off an exact model representation for mesh quality is not appropriate. Radiosity methods typically consist of numerous interconnected surfaces and changing the geometry of one surface without updating adjacent surfaces leads to visible artifacts such as light leakage through cracks between surfaces.

Kela et al. [KELA86] improve the shape of the boundary elements by moving quadrant corners. This preserves the exact geometry of the surface, but it increases the computational expenses significantly, since quadtree quadrants do not have a static, predefined size any more.

Another approach proposed by Shepard and Yerry [YERR83] and Baehman et al.[BAEH87] consists of a Lagrangian relaxation procedure applied on the entire mesh. This leads to a significantly increased mesh quality, but the spatial information provided for each patch by the quadtree is destroyed. Furthermore, mesh relaxation procedures are time intensive because they require that iterative procedures be applied to all vertices of the

mesh. If one region of the mesh is changed, the entire mesh relaxation has to be recomputed. This makes its use for interactive mesh generation techniques with dynamically variable mesh density infeasible.

Since the above approaches did not seem promising for fast and dynamic mesh generation, the poorly shaped boundary elements are improved by simply further subdividing quadrants that contain these patches. This only alleviates the problem and it may increase the number of patches significantly, but it keeps the patch generation algorithm efficient and fast. The higher number of patches, however, makes the radiosity solution more complex and hence more time consuming.

5.5 Changing the Mesh Density

In order to add the mesh generator to a closed feedback loop with radiosity algorithms (adaptive refinement), two operations on the mesh have to be defined. The first is increasing the density of the mesh, and the second is decreasing it. First the radiosity solution for the existing set of patches is computed. Then a radiosity value is assigned to each patch. Changes of the mesh density are based on the difference in radiosity intensity between adjacent patches. This is done by visiting all patches in the tree and comparing their radiosity values to the adjacent patches.

If the difference in radiosity between adjacent patches exceeds a certain value, the mesh density has to be increased in order to provide a better approximation of the underlying lighting situation. This is done by checking if the quadrant to be subdivided is already on the lowest possible tree level. If not, it is subdivided and the stored edges are redistributed to the newly generated quadrants. If there are a number of adjacent patches with very similar radiosity values, these patches should be combined to avoid unnecessarily complex models. Only quadrants that have a common parent can be merged, and the quadrant that has been generated has to fulfill the above decompostion rules. Changing the mesh density in one area, however, demands a rebalancing of the whole tree.

The following pseudocode shows the the implementation of the main functions of the mesh generator:

```
mesh_generator() {

     for (all_surfaces) {
          /* transform the surface into x-y plane */
          transform (surface);

          /* store all edges in quadtree */
          quadtree = discretize (surface);

          /* determine which quadrants are inside surface */
          determine_inside (quadtree);

          /* determine which quadrants are outside surface */
          determine_outside (quadtree);

          /* allow maximum 1 level treedepth difference
             between adjacent quadrants */
          smooth_mesh (quadtree);

          /* use element mesh library to generate patches
             for interior quadrants */
          generate_patches_interior (quadtree);

          /* generate boundary patches by
             subdomain removal */
          generate_patches_boundary (quadtree);

     }
```

```
    /* compute radiosity solution */
    solution = compute_radiosity (quadtree);

    /* display generated radiosity solution */
    display (solution);

    /* adaptive refinement loop: if gradient between
       adjacent patches is too high, refine the mesh
       and compute a new radiosity solution */
    while (gradient (solution) > LIMIT)) {
         refine(quadtree);
         solution = compute_radiosity (quadtree);
         display (solution);
    }
}
```

Chapter 6

Results

This chapter shows and evaluates the radiosity meshes generated by the proposed quadtree-based mesh generator. The algorithm has been implemented in C on a SUN SPARC1+ workstation under UNIX. Two models have been processed:

- "Room 365 office", modeled with AutoCAD by UNC-Chapel Hill graduate student John Alspaugh. This model is composed of 2,443 polygons.

- "Brooks' kitchen", a model of the proposed kitchen of Professor Brooks' residence, modeled with Virtus WalkThrough by UNC-Chapel Hill gradutate student Hans Weber. It consists of 933 polygons.

The color plates in Appendix A show the meshed models as well as the computed radiosity solutions. In order to show the generated mesh, these models are displayed using flat-shaded polygons, with white lines indicating the patches.

The quadtree mesh-generator is designed for collaboration with an interactive radiosity on Pixel-Planes 5, but the implementation is not operational yet. As a first step, the radiosity computations are done using the radiosity program of UNC's Walkthrough project. The unsatisfactory results show that not only the radiosity imposes constraints on the mesh generation, but the mesh generation has in turn specific demands on the radiosity program. In particular, the radiosity must be able to deal with small patches. The UNC-Walkthrough radiosity was designed in 1989 and is primarily concerned with time effectiveness. It assumes that the input polygons are large compared to the desired patchsize, and relies on an integrated mesh generator that subdivides the input polygons on a global grid. If the input has been preprocessed by a mesh generator, the input polygon size and desired patch size are approximately equal. Unfortunately, the patches created by the quadtree meshgenerator typically overlap grid regions and thus are subdivided again, this time according to the global grid. This superposition of meshing algorithms leads to patches of very unsatisfactory shape and size. In addition, the radiosity program does not work well

on patches with significantly different areas because it assumes patches of equal area and therefore does not scale radiositiy values by patch area. These problems create the various artifacts visible in the color plates.

The displayed meshes and radiosity solutions are not the result of an adaptive refinement, rather, the mesh generation and radiosity solution are implemented independently and connected by a pipeline. First the model is meshed to a maximum patchsize of 12*12 inches, and then the radiosity solution is computed.

Color Plate 1 shows the meshed Room 365 office model, consisting of 4,111 patches. 3,482 of these patches are quadrilaterals and 629 are triangles. Meshing this model took about 42 sec of cpu time. In the door region, the anchoring of the t-vertices (in order to avoid shading discontinuities) is visible. Color Plate 2 shows the radiosity solution. There is no shadow leakage from behind the bookshelf, but some shadow leakage from behind the file cabinet is visible.

Color Plate 3 shows the kitchen model. Although initially consisting of only 933 polygons, the meshed model contains 18,615 patches. This is due to the fact that the initial average polygon size was considerably larger than in the Room 365 office model. The mesh generation took 241 sec of cpu time. Large Polygons with holes, partly occluded surfaces and a number of different light sources make this model challenging for the mesh generation. The radiosity solution is shown in color plate 4.

In order to investate the computational growth rate of the algorithm, the kitchen model was meshed with patchsize 24, 20, 16, 12, and 8 inches. The time in order to complete meshing has been taken as well as the number of patches in the meshed model. Figure 6.1 shows the result.

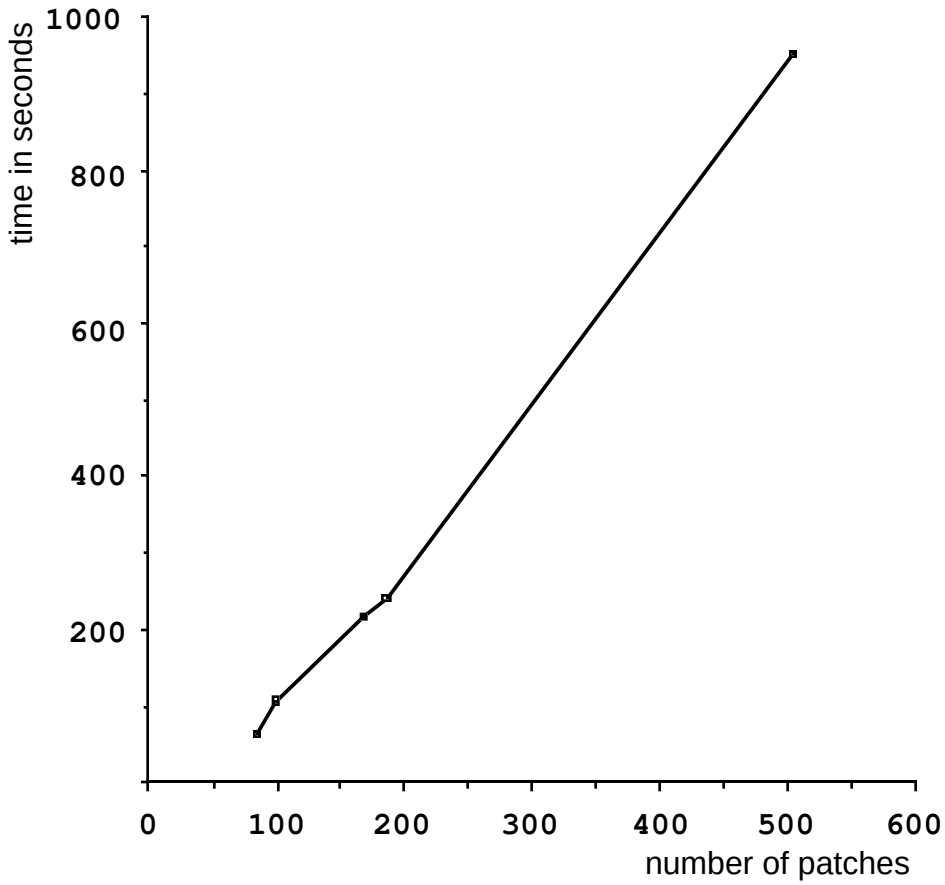

Figure 6.1 Computational growth rate of the mesh generation

The quadtree based mesh generator appears to run in O(n) time, with n being the number of patches in the processed model. This result has also been observed by other implementions of modified quadtrees such as [BAEH87].

Chapter 7

Conclusions and Future Work

The proposed modified-quadtree based mesh generation algorithm produces meshes appropriate for radiosity solutions, provided the input model meets certain geometrical constraints. These constraints can be satisfied by model preprocessing or by the use of a modeler that enforces the generation of physically valid models. The proposed mesh generation algorithm is well suited for interactive radiosity solutions, since the spatial addressabililty property of the quadtree structure employed makes operations such as element retrieval or tree expanding and pruning easy and time-efficient, O(n).

In order to obtain faster and more accurate solution, the proposed algorithm could be implemented on parallel hardware. Since each surface is processed individually, such a parallel extension would be fairly straightforward.

One of the main advantages of this quadtree over the conventional quadtree algorithm is the preservation of the exact shape of the surface stored and the spatial adressability of its patches. Therefore, this algorithm should show its advantages in hierarchical algorithms such as [HANR91] or [HECK90].

Although the generated meshes are generally well behaved, boundary patches are occasionally of poor shape. There are two ways of improving these patches, the first is to process the boundary quadrants by a more powerful method such as Delaunay Triangulation. The second method would be to avoid boundary quadrants of problematic shape by using a different spatial decomposition technique. The present algorithm uses a grid of rectangular shape for this inital decomposition. It is conceivable that triangular grids or grids composed of triangles and quadrilaterals may generate boundary quadrants of better shape. However, this improved mesh generator would demand more complex data structures and algorithms, thus resulting in a less time efficient solution. For the use in interactive radiosity solutions, time efficiency is critical. The decision, how far to trade off mesh quality for time performance, has to be made dependent on the individual application.

The current input to the mesh generator consists of planar polygons. However, many models in engineering are described by analytically defined surfaces such as splines, Bézier, and quadratic surfaces. A possible extension of the algorithm would allow the input of these analytically defined surfaces in 3-D space. Since the input primitives were no longer planar, the quadtree data structure had to be replaced by its three-dimensional extension, the octree. Furthermore, the strategies for meshing quadrants have to be replaced by techniques appropriate for the treatment of octants.

Appendix A

Color Plates

Color Plate 1: Mesh for “Room 365 office”

Color Plate 2: Radiosity solution for “Room 365 office”

Color Plate 3: Mesh for “Brooks’ kitchen”

Color Plate 2: Radiosity solution for “Brooks’ kitchen”

References

[BAEH87] P. L. Baehmann, S. L. Wittchen, M. S. Shephard, K. R. Grice, and M. A. Yerry, “Robust, Geometrically Based, Automatic Two-Dimensional Mesh Generation”, *International Journal for Numerical Methods in Engineering*, Vol. 24, 1987, pp. 1043-1078.

[BARN83] R. E. Barnhill, “A Survey of the Representation and Design of Surfaces”, *IEEE Computer Graphics and Applications*, October 1983, pp.9-16.

[BAUM72] B.G. Baumgart, “Winged-edge Polyhedron Representation”, Technical Report STAN-CS-320, Computer Science Dept., Stanford University, Palo Alto, USA, 1972.

[BAUM91] D. R. Baum, S. Mann, K. P. Smith, and J. M. Winget, “Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions”, *SIGGRAPH ‘91 Proceedings, ACM Computer Graphics*, Vol. 25, No. 4., July 1991 pp. 51-60.

[BOWY81] A. Bowyer, “Computing Dirichlet Tessellations”, *The Computer Journal*, Vol. 24, No. 2, 1981, pp. 162-166.

[CAMP90] A. T. Campbell, III, and D. S. Fussell, “Adaptive Mesh Generation for Global Diffuse Illumination”, *SIGGRAPH ‘90 Proceedings, ACM Computer Graphics*, Vol. 24, No. 4. August 1990, pp. 155-164.

[CHEW83] L. Paul Chew, “Constrained Delaunay Triangulations”, *Algorithmica*, Vol. 4, 1983, pp. 97-108.

[CHEW89] L. P. Chew, “Guaranteed-Quality Triangular Meshes”, Technical Report TR 89-983, Department of Computer Science, Cornell University, 1989.

[COHE85] M. F. Cohen and D. P. Greenberg, “The Hemi-Cube: A Radiosity Solution for Complex Environments”, *SIGGRAPH ‘85 Proceedings, ACM Computer Graphics*, Vol. 19, No. 3, 1985, pp. 31-40.

[COHE88] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg, “A Progressive Refinement Approach to Fast Radiosity Image Generation”, *SIGGRAPH ‘88 Proceedings, ACM Computer Grahics*, Vol. 22, No. 4, August 1988, pp. 75-84.

[FLOR85] L. De Floriani, B. Falcidieno, and C. Pienovi, “Delaunay-based Representation of Surfaces Defined over Arbitrarily Shaped Domains”, *Computer Vision, Graphics and Image Processing*, Vol. 32, 1985, pp.127-140.

[FOLE90] J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Grahpics: Principles and Practice*, second edition, Addison-Wesley, 1990.

[FREY87] W. H. Frey, “Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes”, *International Jounal for Numerical Methods in Engineering*, Vol. 24, 1987, pp. 2183-2200.

[FUCH89] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, “Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memory”, *SIGGRAPH ‘89 Proceedings, ACM Computer Graphics*, Vol. 23, No. 3, July 1989, pp. 79 - 88.

[GORA84] C. M. Goral, K.E. Torrance, D.P. Greenberg, and B. Battaile, “Modeling the Interaction of Light Between Diffuse Surfaces”, *SIGGRAPH ‘84 Proceedings, ACM Computer Graphics*, Vol. 18, No. 3, August 1984, 213-222.

[GREE77] P. J. Green and R. Sibson, “Computing Dirichlet Tesselations in the Plane”, *The Computer Journal*, Vol. 21, No. 2, 1977, pp. 168-173.

[HANR91] P. Hanrahan, D. Salzman, and L. Aupperle, “A Rapid Hierarchical Radiosity Algorithm”, *SIGGRAPH ‘91 Proceedings, ACM Computer Graphics*, Vol. 25, No. 4, July 1991, pp. 197-205.

[HECK90] P. S. Heckbert, “Adaptive Radiosity Textures for Bidirectional Ray Tracing”, *ACM Computer Graphics*, Vol. 24, August 1990, pp. 145-154.

[HUGH87] T. J. R. Hughes, *The Finite Element Method*, Prentice-Hall, 1987.

[HOLE88] K. Ho-Le, “Finite Element Mesh Generation Methods: A Review and Classification”, *Computer Aided Design*, Vol. 20, No. 1, 1988, pp. 27-38.

[KELA86] A. Kela, R. Perucchie, and H. Voelcker, “Toward Automatic Finite Element Analysis”, *Computers in Mechanical Engineering*, Vol. 5, No. 1, July 1986, pp. 57-71.

[LIND83] D. A. Lindholm, “Automatic Triangular Mesh Generation on Surfaces of Polyhedra”, *IEEE Transactions on Magnetics*, Vol. MAG-19, No.6, November 1983.

[MEAG82] D. Meagher. “Geometric Modeling Using Octree Encoding”, *Computer Graphics and Image Processing*, Vol. 19, 1982, pp. 129-147.

[REQU82] A. A. G. Requicha, H.B. Voelcker, “Solid Modeling: A Historical Summary and Contemporary Assessment”, *IEEE Computer Graphics and Applications*, March 1982, pp. 9-25.

[REQU83] A. A. G. Requicha, H. B. Voelcker, “Solid Modeling: Current Status and Research Directions”, *IEEE Computer Graphics and Applications*, October 1983, pp. 25-37.

[RICH88] G. Richards, A. Karagas, P. Cooley, “Interpretation of Engineering Drawings as Solid models”, *Computer-Aided Engineering*, April 1988, pp. 67-78.

[SAME85] H. Samet, R. E. Webber, “Storing a Collection of Polygons Using Quadtrees”, *ACM Transactions on Graphics*, Vol.4, No.3, July 1985, pp. 182-222.

[SAME87] H. Samet, “Hierarchical Data Stuctures and Algorithms for Computer Graphics”, Technical Report CS-TR-1752, University of Maryland at College Park, January 1987.

[SAME90] H. Samet, *Application of Spatial Data Structures*, Addison Wesley, 1990.

[SCHU89] S. Schuirer, “Delaunay Triangulations and the Radiosity Approach”, *Eurographics ’89*, North Holland, 1989, pp. 345-353.

[SEUS53] Dr. Seuss, “The Sneetches”, *The Sneetches and Other Stories*, pp.2-25, Random House, 1953.

[SHEP83] M. S. Shepard and M. A. Yerry, “Approaching the Automatic Generation of Finite Element Meshes”, *Computers in Mechanical Engineering*, April 1983, pp. 49-56.

[SHEP88] M. S. Shepard, “Approaches to the Automatic Generation and Control of Finite Element Meshes”, *Applied Meachnics Reviews*, Vol. 41, No. 4, April 1988, 169-184.

[SIEG81] R. Siegel, *Thermal Radiation Heat Transfer*, Second Edition, McGraw-Hill, 1981.

[TOUS91] G. Toussaint, “Efficient Triangulation of Simple Polygons”, *Visual Computer*, Vol. 7, 1991, pp. 280-295.

[VIRT90] Virtus Corporation, *Virtus WalkThrough: User’s Manual*, Virtus Corporation, 117 Edinburgh South, Suite 204, Cary, NC 27511, USA.

[WALL91] J. Wallace, “Problems with Model Data”, *SIGGRAPH ‘91, Course Notes 11: Radiosity*, pp. VII.6-VII.13.

[YERR83a] M. A. Yerry and M. S. Shepard, “A Modified Quadtree Approach to Finite Element Mesh Genration”, *IEEE Computer Graphics and Applications*, January / February 1983, 39-46.

[YERR83b] M. Yerry and M. Shepard, “Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique”, *International Journal for Numerical Methods in Engineering*, Vol. 20, 1983, pp. 1965-1990.