VOLUME BASED TEXTURE MAPPING

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Burkan Saik
February, 1995

THESIS
T
385
.S25
1995

VOLUME BASED TEXTURE MAPPING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Gürkan Salk

February, 1995

Gürkan SALK

tarafından bağışlanmıştır

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Bülent Özgüç (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Cevdet Aykanat

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Asst. Prof. Faruk Polat

Approved for the Institute of Engineering and Science:

[illegible]

Prof. Mehmet Baray
Director of the Institute

T
385
.C25
1995

B005074

ABSTRACT

VOLUME BASED TEXTURE MAPPING

Gürkan Salk
M.S. in Computer Engineering and Information Science
Advisor: Prof. Bülent Özgüç
February, 1995

The most realistic and attractive computer generated images are usually those that contain a large amount of visual complexity and detail. Texturing is a widely used way of adding visual complexity and detail to computer generated images. Traditionally surface texturing was only used to simulate surface detail. In this thesis we generate textures defined throughout a region of three-dimensional space and map those textures together with their geometric definition onto complex objects. The textured object is rendered volume based with a backward mapping algorithm (ray tracing). Hence the texture affects the definition and the realism of the object. In rendering the scene, natural phenomena such as dispersion and absorption of light is also incorporated.

Keywords: 3-D Texture Mapping, Ray Tracing, Dispersion

ÖZET

HACİMLİ DOKU KAPLAMA YÖNTEMİ

Gürkan Salk
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Danışman: Prof. Bülent Özgüç
Şubat 1995

En gerçekci ve etkileyici bilgisayar çıktısı görüntüler, genellikle yeterince ayrıntılandırılmış ve görsel karmaşıklığa sahip olanlardır. Doku kaplama yöntemi bu görüntülere görsel karmaşıklık ve ayrıntı eklemenin en yaygın yöntemlerinden biridir. Eskiden doku kaplama sadece yüzey ayrıntısını artırmak için kullanılırdı. Bu tezde kaplanacak dokular üç boyutlu uzayda oluşturulup, geometrik tanımlarıyla birlikte karmaşık cisimler üzerine kaplanırlar. Doku kaplanmış cisim ışın izleme yöntemiyle tonlandırılır. Kullandığımız yöntemde doku cismin tanımını ve görünüşünü doğrudan etkilemektedir. Cisimlerin bulunduğu sahne bu yöntemle oluşturulurken ışığın soğurulması ve ışığın ayrışması gibi doğal olgular da dikkate alınmıştır.

ACKNOWLEDGMENTS

This thesis would not have been possible without the sympathetic aid given and the deep interest shown by my supervisor, Prof. Bülent Özgüç. His invaluable instruction in the field of computer graphics, especially in texture mapping, ray tracing and their combination has been my constant guidance and source of inspiration.

I would like to express my thanks to both members of my thesis committee, Assoc. Prof. Cevdet Aykanat and Asst. Prof. Faruk Polat for their valuable comments.

I cannot fully express my gratitude to Sibel Salman for her invaluble help and support.

I would like to express my deepest thanks to my family for making it possible.

Contents

1 Introduction **1**

1.1 Texturing and Volume Based Rendering in Literature 2

1.2 The Proposed Model 4

2 3D Texture Generation **6**

2.1 Methods For Generating Textures 6

2.1.1 Bombing 7

2.1.2 Fourier Synthesis 7

2.1.3 Projection Functions 8

2.2 Implementation 9

2.2.1 Rendering the image 9

2.2.2 Texture Generation 11

3 Light and Colour Calculations **15**

3.1 Direction Calculation for the Refracted Ray 15

3.2 Direction Calculation for the Reflected Ray 17

3.3 Intensity Calculation 17

3.3.1 The Intensity Calculation of Reflected and Refracted Rays 18

3.3.2 Absorption of Light 18

3.3.3 Intensity of a Light Source 20

3.4 Calculation of Colour 21

3.4.1 Primary Colours 22

3.4.2 Dispersion 22

3.5 Implementation 26

4 Ray Tracing **29**

4.1 Implementation 31

4.1.1 Implementation of Shading and Penumbras 32

4.1.2 Implementation of the Anti-aliasing Method 33

4.1.3 Calculating Intersections 35

4.1.4 The General Model 37

4.1.5 Results 40

5 Conclusion **41**

Appendix

A Sample Images **43**

List of Figures

2.1 A texture generated by bombing spheres into a cube 8

2.2 A wood texture generated by deformed projection 13

3.1 The solid angle ω defines an area s on the surface of a sphere . . 21

3.2 Wavelengths of primary colours 23

3.3 The decomposition of visible light into its component wavelength regions (colours) by a glass prism 24

3.4 Anomalous dispersion 25

3.5 The dispersion curve for the Borosilicate Crown material 28

4.1 Solid angle calculation in shading 32

4.2 Algorithm for adaptive sampling 34

4.3 An example to refinement selection 35

4.4 Examples of weighting windows 35

4.5 Traversal of a ray in two-dimensional grid 36

4.6 Voxel traversal algorithm 37

4.7 Ray Tracing algorithm 39

A.1 A texture generated with deformed projection 44

A.2 A marble 45

A.3 A wood textured object, where the wood texture is generated with deformed projection 46

A.4 A scene with two prisms, where the front one is texture mapped with bombing and is transparent, whereas the other one is non-transparent 47

A.5 A scene with two prisms, where the front one is texture mapped with bombing and is transparent, whereas the other one is non-transparent 48

A.6 The role of refractive index in dielectrics (animation) 49

A.7 The effectiveness of the adaptive supersampling anti-aliasing method 50

A.8 The effect of aliasing in point sampled ray tracing 51

Chapter 1

Introduction

Texturing is an efficient and low-cost technique for adding details and enhancing the optical complexity of computer-generated objects with the aim of achieving more realism. It is an efficient and an inexpensive method since the simulation of the details of an object is done without modeling the object explicitly. The texturing process consists of two steps: texture sampling and texture mapping. Texture sampling ([11], [17]) involves the calculation of the texture values at the location of the calculated texture coordinates. In this method the generation of the texture data and the geometry of the object are totally uncoupled. These two attributes are combined together by means of the texture mapping process. Texture mapping is the calculation of texture coordinates using the object coordinates at a particular location of interest, for example the location of the ray-object intersection during ray tracing. This texturing process, which we call *2D texturing*, works satisfactorily for texturing solid objects as well.

Solid texturing, which is a variation of 2D texturing, uses texture functions defined throughout a region of three-dimensional space ([25], [29]). Many kinds of non-homogeneous material, including wood and stone, may be more realistically rendered using solid texture functions. In solid texturing, the texture is specified as a spatial 3D pattern defining a unity block from which the body is sculptured. The main advantage of solid texturing is that it can be easily applied, by means of the mapping process, to complex surfaces which are difficult to texture using two-dimensional texture functions. Another advantage of solid texturing, which our proposed model in this thesis exploits, is that the texture in the object has its own geometry. This property is crucial for texturing transparent objects. Traditionally texture mapping (2-dimensional or 3-dimensional) was only used to give the surface of an object a colour value depending on the texture to be mapped. In this thesis, we handle the texture as objects which have their own refraction index, diffusion component, and other physical properties. However, with this method, texturing is not a low-cost and simple process anymore. Rendering a transparent object with non-transparent texture, such as a marble block, is not possible with the traditional 2D texturing. Instead, 3D texturing features are used for the realization of such objects and scenes. Another advantage of solid texturing is that it eliminates the aliasing problems that arise from the highly compressed surface coordinate system near the poles of a sphere or in regions of tight curvature on some parametric surfaces. Considering these facts, we used 3D textures in our model for representing objects.

1.1 Texturing and Volume Based Rendering in Literature

Texturing has been a critical development in the process of achieving realism in computer-synthesized images. Earlier work in computerized image generation lacked surface detail even if features such as shininess and transparency were incorporated. Later, with the modeling of complex surface variations, called texture, could computer images acquire more realism.

In 1974, Catmull [6] implemented the first system to use images of texture applied to surfaces to give the affect of actual texture. Basically, the system involved wrapping a 2D texture around an object. Blinn and Nevell [3] generalized Catmull's work and extended it to include environmental reflections. Then, Blinn [2] achieved the appearance of undulations on the surface as an improvement to earlier flat texture (such as the fake wood texturing found on many plastic desk tops) by a method called "bumb mapping".

In order to map texture onto a surface, texture coordinates must be calculated for each pixel representing a textured surface. The most straightforward application of texture mapping simply chooses the pixel from the texture image which lies closest to the computed texture coordinates. However, this method works well for only a certain class of textures and surfaces. When the texture is mapped onto a surface, it must be stretched and compressed in order to fit the shape of the surface and this may cause aliasing problems. Unless the texture image is very smooth, sharp details will become jagged and the texture will break up where it is highly compressed. This problem is discussed by Blinn [2] and later by Feibush et al [11]. In [11] an effective but very expensive solution is given. Later, many researchers have worked on eliminating this problem. Nevertheless, 2D texturing has another serious disadvantage: In 2D texturing the geometry of the texture is not available. This makes the use of 2D texture for volume rendering problemsome.

As an alternative to 2D texturing, Peachey [25] and Perlin [26] introduced the notion of "solid texturing" independently and simultaneously. Solid texturing uses texture functions defined throughout a region of 3D space. Peachey gave examples of solid texture functions based on Fourier synthesis, stochastic texture models, projections of 2D textures, and combinations of these functions. Solid texturing functions do not depend on the surface geometry and hence can be applied to complex surfaces which are difficult to texture using 2D texturing due to the aliasing effects. The other disadvantage of 2D texturing is also eliminated in 3D texturing since the texture has its own geometry.

The field of volume visualization can be traced back to the beginning of the 1970s. The first research was made in 3D medical imaging by Greenleaf, Tu and Wood [16]. Till now two principle approaches have been developed for volume rendering: backward mapping algorithms that map the image plane onto the data by shooting rays from pixels into the data space, and forward mapping algorithms that map the data onto the image plane. The forward mapping algorithms have been developed by reducing the volume array to only the boundaries between materials. Thus, the data image is divided into slices and is projected to the image space by combining the intensities of the portions of the slices, which correspond to a given pixel. There are several methods for the combination and calculation of the intensities on the image space. Forward mapping algorithms are developed by researchers such as Lorensen [23] and Westover [36]. The back mapping algorithms are methods where mainly ray tracing is used as the global illumination model (also called *depth cueing* in computer graphics literature). In these methods rays are traced through the data until they hit a surface and then an intensity which is inversely proportional to the distance to the eye is assigned to the corresponding pixel (Vannier [34]). Radiation transport equations have been used to simulate transmission of light through volumes (Kajiya [19]). The *low-albedo* or *single scattering* approximation has been applied to model reflectance functions from layered volumes (Blinn [4]). In all of these algorithms rays are traced in any direction through a volume array. Other algorithms for ray tracing volumes are described in (Fujimoto [12], Tuy [33] and Levoy [22]). The implemented algorithms are mainly used to abstract natural phenomena like clouds and volumes with a given density.

1.2 The Proposed Model

Our model is a backward mapping algorithm (*depth cueing*), mainly based on the effects of material on light, that is, how light is affected after it intersects with a medium, how dispersion, scattering and absorption effects occur. The model incorporates all of these natural effects to get more realistic pictures. The model is used to abstract solids (dielectrics) rather than volume densities. The structure of the dielectrics is assumed to be smooth, which is in fact rarely the case.

Another important point in achieving realistic images is the use of global illumination technique. Early local illumination techniques such as Gouraud or Phong shading are not adequate for the generation of realistic images. The popular and effective global illumination techniques are *ray tracing* and *radiosity*. Since our subject deals mainly with refraction, dispersion and scattering of light, we used ray tracing as the global illumination method, which is the only global illumination technique capable of handling refraction. This property comes from its ray oriented structure. The technique has both advantages and disadvantages as stated below.

Advantages:

- Ray tracing uses a global lighting model that calculates reflections, refractions and shadows.

- Ray tracing can handle a variety of geometric primitives.

Disadvantages:

- Ray tracing is often slow, since the intersection calculations are floating point intensive.
- Point sampling (Since ray tracing is a point sampling global illumination technique) the environment causes aliasing.

To overcome these disadvantages we used several techniques. To speed up the ray tracing we implemented a voxel based algorithm introduced by [1]. To overcome the aliasing effects we used an adaptive sampling technique introduced by [31]. These methods are explained in Chapter 4 in detail.

In Chapter 2, we describe 3D texture generation methods and their application to our model. Chapter 3 gives technical foundation for effects of materials (dielectrics) on light and colour. The application of these natural phenomena in our model is also discussed in this chapter. Chapter 4 gives the implementation of ray tracing as a global illumination rendering method in our model. The general structure of our proposed model is also described in this chapter. Finally, we conclude and give future research directions in Chapter 5.

Chapter 2

3D Texture Generation

The most realistic and attractive computer generated images are usually those that contain a large amount of visual complexity and detail. Surface texturing is an effective method of simulating surface detail at relatively low cost. Traditionally texture functions have been defined on the two-dimensional surface coordinate systems of individual surface patches. Alternatively, 3D textures, also called solid textures, are defined throughout a region of three-dimensional space. 3D texture generation is superior to the traditional 2D methods as it neatly circumvents the mapping problem. Since the texture value in the generated texture exists everywhere in the object domain, it is easy to map a point on the surface of the object, x_W, y_W, z_W, to a point on the texture, which is given by the identity mapping $T(x_W, y_W, z_W)$. Basically, 3D texture generating methods are *bombing*, *Fourier synthesis*, *orthogonal projection of two-dimensional textures* and *orthogonal projection of objects*, such as cylinders, that are deformed by means of twisting or bending functions. We concentrate on the texture generation methods in this chapter .

2.1 Methods For Generating Textures

In principle, solid texture functions can be evaluated in most of the ways which are popular for two-dimensional texture functions. Texture functions can be divided into digitized textures and synthetic textures. Digitized textures are more popular in two-dimensional textures, because it is relatively easy to digitize a photograph. Digitizing solid textures is less convenient since it involves the two-dimensional digitization of a large number of cross-sectional slices. On the contrary, synthetic textures are more flexible in the aspect that they can be designed to have certain desirable properties. For example, a synthetic texture can often be made smoothly periodic, so that it can be used to fill in infinite space without visible discontinuities. Based on the above discussion, we have used functions to generate only synthetic textures by the following techniques.

2.1.1 Bombing

Bombing is a random pattern generation process which has been successfully implemented in two-dimensional texturing by Schachter and Ahuja [30]. The main idea of bombing is randomly dropping bombs of various shapes, sizes and orientations onto the texture space. Utilizing this idea, we have bombed the three-dimensional texture space by cylinders, cubes and spheres. In Figure 2.1 there is a texture generated by bombing 365 spheres with radius of maximum 14 pixels (radii of spheres are randomly generated from a uniform distribution between 1 and 14 pixels) onto a 128x128x128 cube. The texture is generated by placing the spheres randomly into the cube.

2.1.2 Fourier Synthesis

Fourier synthesis can be used as a basis for representing various natural phenomena including water and terrain. Building a three-dimensional texture field using Fourier synthesis means generating parameters which specify the amplitude, frequency and phase of sinusoids. These parameters are then linearly combined to produce a function in which the underlying periodicities may be masked by a careful choice of the design parameters. Gardner [13] uses a three dimensional function G(X, Y, Z) to model the amorphous shape of tree and clouds, modulating the surface intensity and the transparency of the ellipsoids. We have also used this function. The parameter scheme by Gardner is:

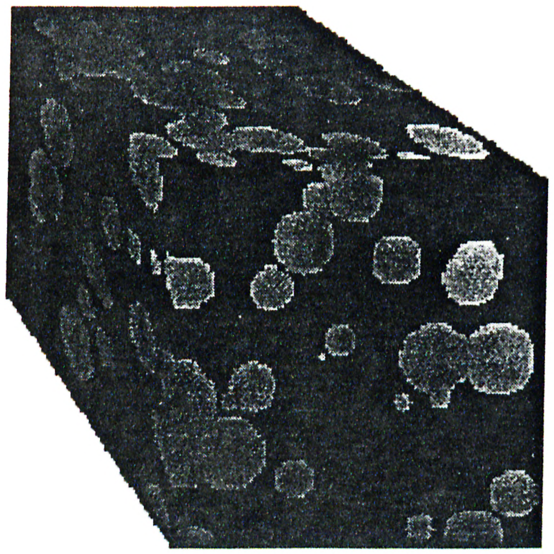

Figure 2.1. A texture generated by bombing spheres into a cube

$$\begin{aligned} G(X,Y,Z) = \ & \textstyle\sum_{i=1}^{n} C_i[cos(w_{x_i}X + \phi_{x_i}) + A_0] \\ & \times \textstyle\sum_{i=1}^{n} C_i[cos(w_{y_i}Y + \phi_{y_i}) + A_0] \\ & \times \textstyle\sum_{i=1}^{n} C_i[cos(w_{z_i}Z + \phi_{z_i}) + A_0] \end{aligned}$$

where n is a value between 4 and 7, and $C_{i+1} \approx 0.707C_i$. C_1 is chosen such that G(X,Y,Z) $\leq$ 1. The initial values of w specify the underlying or base frequencies such as the rolling of hills in terrain. ϕ_{x_i}, ϕ_{y_i} and ϕ_{z_i} are phase shifts into which a random component can be built. A_0 is the basic offset providing contrast control.

2.1.3 Projection Functions

Projection functions are a class of solid texture functions based on 2D textures which are projected through 3D space. For example, Peachey [26] and Gardner [14] have used orthogonal projection to approximate wood grain by applying a two-dimensional texture ρ(u,v) to a complex surface using the orthogonal projection function R:

$$R(X,Y,Z) = \rho(X,Y) \text{ for } X \text{ and } Y \in [0,1]$$
$$R(X,Y,Z) = 0, \text{ otherwise.}$$

Here, R simply projects the texture ρ along the Z axis. Each texture element of ρ generates a rectangular parallelepiped that extends infinitely in both directions on the Z axis of the 3D texture space. We have also used this method under the name "orthogonal projection".

Another projection function based method that we have used is what we call "deformed projection". In this method, an object is selected and is projected by a deformation function (e.g. a twisting function). A texture is generated by sweeping geometric objects embedded in each other along the Z-axis. It is easy to generate procedural textures such as wood texture using this method.

2.2 Implementation

The implementation of a 3D texture generator is made on the X-windows environment using C language. Operations of the user-interface can be grouped into four master groups. The first group of operations are general operations such as loading a file, saving a file, clearing the canvas, drawing the image and exiting from the program, which are implemented via buttons. The second group of operations consist of rendering the image such as shading, light vector position, and projection. The third group of operations are operations for changing the properties of the cube in which the texture is generated. These properties are colour and rotation. The fourth group of operations are those which are used to generate the texture in the cube.

2.2.1 Rendering the image

To render the generated 3D textures we used local illumination techniques such as constant, Gouraud and Phong shading. Shading is a difficult concept in this program, since the colour is set in a dynamic way due to the limited colour palette. The number of usable entries of the colour palette is limited by approximately 240. We divided the colour palette into 3 parts, where each part contains intensities of the colours red, blue and green respectively. The interpolation of the colours is made dynamically. If a maximum red intensity of 150 is selected, then the 80 locations reserved for red are interpolated in such a way that the 80th entry has red intensity of 150 and the other entries contain intensity values uniformly distributed between 0 and 150. Better results are achieved using this type of interpolation.

All shading methods are implemented in such a way that only the polygons of the objects have to be added to an edge_list and then the whole object is rendered. First all edges of the objects are added to an edge_list (the objects are represented as wire frames) holding the x and z values of each corresponding y value and calculating these incrementally. The edge_list structures for the implemented three shading algorithms are different since the data needed for Phong and Gouraud shade are different.

Features such as the light source position and the orientation of the cube can be changed using the mouse. Hence, the interaction of the user and the program is done mainly with the mouse. Projection used in the program is perspective projection.

2.2.1.1 Constant Shading

Constant shading is used on objects having plane surfaces, that can be realistically shaded using constant surface intensities. The constant shading model produces a constant surface intensity, provided that the point source and the view reference point are sufficiently far from the surface. Suppose that N is the surface normal, L is the direction of the light vector and V is the viewing direction. Thus, when the point source is far from the surface, there is no change in the direction to the source ($N \cdot L$ is constant). Similarly, the direction to a distant viewing point will not change over a surface, so $V \cdot R$ is constant. On objects with complex surfaces, constant shading is an inefficient method. As the generated texture is to be mapped on more complex objects than a cube, and furthermore, as we want to implement bumb textures as well, the need for shading methods different than constant shading arises.

2.2.1.2 Gouraud Shading

This intensity interpolation scheme, developed by Gouraud [15], removes intensity discontinuities between adjacent planes of a surface representation by linearly varying the intensity over each plane so that intensity values match at the plane boundaries. In this method, intensity values along each scan line passing through a surface are interpolated from the intensities at the intersection points with the surface. At each intersection point of an edge of the surface and a scan line we find an intensity and the intensity on the rest of the surface is calculated using the intensity on the edges. In this method, first surface normals must be approximated at each vertex of the polygon. This is accomplished by averaging the surface normals of each polygon containing the vertex point. These vertex normal vectors are then used to generate the vertex intensity values.

2.2.1.3 Phong Shading

In the Gouraud shading method, we have calculated only the intensities on each vertex and then interpolated the intensities along the edges and the scan lines. In the Phong shading method, all the normal vectors on the scan line are interpolated and then the intensity for each point is calculated.

2.2.2 Texture Generation

In the program, all of the texture generation methods introduced at the beginning of the chapter are implemented.

2.2.2.1 Implementation of Bombing

The bombing method is implemented as described in Section 2.1.1., where the user is able to select the type of object to be bombed into the texture cube. The possible objects are sphere, cylinder, and cube with given radius and/or height. The placement of the objects into the texture cube is made randomly.

2.2.2.2 Implementation of Deformed Projection

This 3-dimensional texture generation method is used to define the texture throughout the 3D space procedurally. It is well suited for abstracting natural textures, such as wood. The main idea is to sweep embedded 2D geometrical objects along the Z-axis (e.g. circle). But sweeping without deforming the object results in a too smooth texture, which is not the case in real textures, for example wood. Thus, we added deforming functions, where the objects can be deformed while sweeping. The deformation functions implemented are tapering and twisting.

Tapering is easily developed from scaling. We implemented a method, where we choose a tapering axis (Z-axis) and differentially scale the other two components. Thus, to taper an object along its Z-axis:

$$X = rx,\ Y = ry,\ Z = z$$

where $r = f(z)$ is a linear tapering profile function, (x, y, z) is a vertex in an undeformed solid and (X, Y, Z) is the deformed vertex. In our implementation, we chose a linear function for r, where $r = a_i * z$. In this function, a_i is a variable which is randomly generated from a uniform distribution between -0.06 and 0.06. To prevent overlapping of objects, we choose the difference between the radii or side lengths of objects to be greater than $2 \cdot 0.06 \cdot Max\_Z$, where Max_Z is the maximum value until which the objects are sweeped on the Z-axis. Note that the value 0.06 pixels is not a constant and other numbers can be tried to obtain better results. The radius/side length of each object is found with respect to this formula: $R_i = i * C$, where i shows the object number and C is the radius/side length difference. Thus $C > 2 \cdot 0.06 \cdot Max\_Z$. Finally, the tapering formula for each object is defined as follows:

$$\begin{aligned} X_1 &= f_1(z) \cdot x + x \\ Y_1 &= f_1(z) \cdot y + y \\ Z_1 &= z \end{aligned}$$

Resulting in the following recursive formula:

$$X_i = f_i(z) \cdot x_i + f_{i-1}(z) \cdot x_i + x_i$$

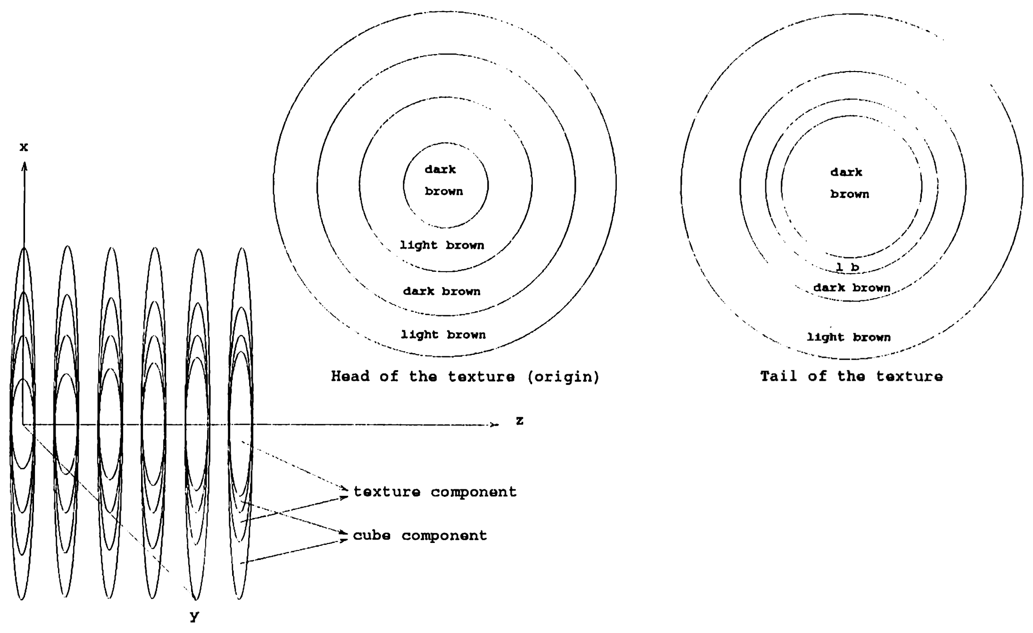

Figure 2.2. A wood texture generated by deformed projection

$$\begin{aligned} Y_i &= f_i(z) \cdot y_i + f_{i-1}(z) \cdot y_i + y_i \\ Z_i &= z \end{aligned}$$

where $1 < i < No\_of\_objects + 1$.

An example is given in Figure 2.2. There are 4 objects (circles) sweeped along the Z-axis while being deformed with the following data:

$$\begin{aligned} f_1(z) &= 0.1 \cdot z \\ f_2(z) &= -0.1 \cdot z \\ f_3(z) &= -0.05 \cdot z \\ f_4(z) &= 0.05 \cdot z \end{aligned} \tag{1}$$

Radii of the circles are 2, 4, 6, and 8 respectively. As can be seen in Figure 2.2, this method is appropriate for abstracting wood textures and other procedural 3D-textures.

Twisting is developed as a differential rotation. To twist an object about its Z-axis we apply :

$$\begin{aligned} X &= x \cdot cos\theta - ysin\theta \\ Y &= x \cdot sin\theta - ycos\theta \\ Z &= z \end{aligned}$$

Applying twisting deformations on circles does not make much sense, so it is prefered to use objects like rectangles for twisting. In the implementation we used circles and rectangles for generating textures with deformed projection. Again to prevent the overlapping of the objects, we must assure a sufficient clearance between adjacent objects. Thus, the treshhold of space, which guarantees no overlapping, is equal to the diagonal of the rectangle. If object i is a x by y rectangle, object i will not intersect with object $i+1$ after a twisting function, if object $i+1$ has both edges longer than $\sqrt{x^2+y^2}$. We set object lengths which satisfy this criterion.

Using the union of two deformation functions is also possible. In this way, different 3D textures can be generated by combining various tapering and twisting functions.

2.2.2.3 Implementation of Fourier Synthesis and Orthogonal Projection

These two 3D texture generation methods are implemented exactly the way described in Sections 2.1.2. and 2.1.3. However, these two techniques are not suitable for volume based rendering not only because it is difficult to distinguish between the texture and the material area, but also the number of distinct textures in a generated texture may be unknown. Thus, the textures generated via these methods are mapped on solid objects and used only to give colour to the surfaces of the objects as in classical texture mapping.

Chapter 3

Light and Colour Calculations

In order to represent solids in a more realistic way, in addition to using synthetic texture, we represent semi or full transparent objects with semi, non or full transparent textures in them. For example an object which is textured by bombing can be a semi-transparent object, where the bomb texture is non transparent. This operation thus includes colour calculations (assuming that the colours of the texture and the object are different) and refraction computations. In order to handle those calculations easily and to get high quality images we will use the ray tracing method as the rendering method. Hence, we can get more realistic pictures in representing objects like marbles with solid textures and balls made of semi-transparent marble and so on. In ray tracing the most important topics for generating high quality images are the calculation of the reflection and refraction vectors, the colour value of the object at each pixel and the calculation of the intersection point of the ray with the object.

3.1 Direction Calculation for the Refracted Ray

The refraction operation determines the direction of the refracted ray and the colour value at the intersection point. The inputs to this operation are: (1) the direction of the surface normal; (2) two refractive indices, one on each side of the refracting surface; and (3) the direction of the incident ray. Here for calculating the direction of the refracted ray we use Snell's law. Accordingly the direction of the refraction vector will be determined as follows : Let N be the direction vector for the incident ray and let N^* be the direction vector for the refracted ray. Moreover, let $\overline{N}$ be the normal to the refracting surface, and let i_1 be the angle of incidence (angle between the incident ray and the surface normal) and i_2 be the angle of refraction. Finally let μ be the ratio of the refractive indices on opposite sides of the refracting surface. (That is, if n is the index of refraction on the incident-ray side of the refracting surface and n^* is that on the refracted-ray side, then $\mu = n/n^*$). The vector form of Snell's law is:

$$N^* \times \overline{N} = \mu(N \times \overline{N}) \tag{1}$$

and the scalar version is

$$sin(i_2) = \mu sin(i_1) \tag{2}$$

where i_2 and i_1 are the angles of incidence and refraction, respectively. Thus, we get

$$(N^* - \mu N) \times \overline{N} = 0 \tag{3}$$

which means that the vectors $(N^* - \mu N)$ and $\overline{N}$ are parallel. Therefore, we can find a scalar quantity γ such that $(N^* - \mu N) = \gamma \overline{N}$, which gives us the formula for the direction vector of the refracted ray,

$$N^* = \mu N + \gamma \overline{N} \tag{4}$$

If we can determine γ, we can easily find the direction vector, since the other variables in the formula are known. Reminding that N^*, N and $\overline{N}$ are unit vectors, if we get the square, the equation turns to :

$$1 = \mu^2 + \gamma^2 + 2\mu\gamma(N \cdot \overline{N}), \tag{5}$$

Hence, the solution for γ is

$$\gamma = -\mu(N \cdot \overline{N}) \pm \{1 - \mu^2[1 - (N \cdot \overline{N})^2]\}^{1/2} \tag{6}$$

The plus sign should be used between the two terms, because if the incident ray intersects with a perpendicular surface where the refraction indices are the same, then $N = \overline{N}$ and $\mu = 1$. Therefore, $N \cdot \overline{N} = 1$ and $\gamma = -\mu \pm 1$. Since from equation (Eq-4) γ should be zero (direction vector of the incident ray and the refracted ray should be the same) and $\mu = 1$, the plus sign should be used.

So, the expression for γ is:

$$\begin{aligned}\gamma &= -\mu(N\cdot\overline{N}) + \{1-\mu^2[1-(N\cdot\overline{N})^2]\}^{1/2} \\ &= -\mu(N\cdot\overline{N}) + \{1-\mu^2(N\times\overline{N})^2\}^{1/2} \\ &= -\mu(N\cdot\overline{N}) + \{1-\mu^2(N^*\times\overline{N})^2\}^{1/2} \\ &= -\mu cos(i_1) + \{1-(N^*\times\overline{N})^2\}^{1/2} \\ &= -\mu cos(i_1) + cos(i_2)\end{aligned} \tag{7}$$

We conclude that to find the direction of a refracting ray we need only apply Eq. (3) and Eq. (6).

3.2 Direction Calculation for the Reflected Ray

Reflection can be seen as a special case of refraction. If we take the refraction index as $\eta^* = -\eta$ the same equations as in the refraction operation can be used. So from equation 6, $\gamma = 2cos(i_1)$ and from equation 3, the direction of the reflected ray is :

$$N^* = -N + 2cos(i_1)\overline{N} \tag{8}$$

Having outlined the methodologies used in finding the directions of the reflected and refracted rays, the next step is to explain the calculation of the intensities at the intersection points and the factors that effect the intensities of the refracted and reflected rays.

3.3 Intensity Calculation

The factors that effect the intensity of a ray and consequently the intensity and colour values of the object at each pixel are the intensity of the incidence ray and the absorption of the object. The calculation of those intensities (reminding that the used method is ray tracing) is explained in the following sections. In those calculations the index of refraction of the object and the texture is a necessary constant, which should be known.

3.3.1 The Intensity Calculation of Reflected and Refracted Rays

The intensity value of the reflected ray can be calculated by the following formula :

$$\frac{I_{reflected}}{I_{incident}} = \frac{\eta_2 - \eta_1}{\eta_2 + \eta_1} \Longrightarrow I_{reflected} = \frac{\eta_2 - \eta_1}{\eta_2 + \eta_1} * I_{incident} \tag{9}$$

Where η_2 is the index of refraction of the denser medium and η_1 is the index of refraction of the less dense medium. For example a ray intersecting with a glass (index of refraction of glass is 1.5 and index of refraction of air is 1) with an angle of incidence say of 15^o will reflect 4% of the incident light and refract the remaining intensity. It can be seen from the formula that the amount of light reflected and refracted depends on the refraction index of the medium, which in turn results in transparency. Thus materials with large indices of refraction turn out to be more opaque.

Another aspect in the reflected light calculation is the so called *total internal reflection*. This happens when a refracted ray enters a transparent object and strikes the surface at angles greater than a particular angle (angle with the surface normal) called the *critical angle of incidence*, i_c. This value again depends on the index of refraction of the transparent object. In this case the ray is reflected fully back and does not escape the medium. The angle of incidence can be calculated using the following formula:

$$sini_c = \frac{\eta_1}{\eta_2} \tag{10}$$

where η_1 is the refractive index of the less dense medium and η_2 is the refractive index of the more dense medium. Note that the phrase 'does not escape the medium' is valid for only convex objects. In our implementation i_c is checked at each intersection and the ray is reflected fully only when it makes an angle greater than i_c with the normal.

3.3.2 Absorption of Light

Another aspect, which effects the intensity of a pixel on the object is the absorption of the intensity of the refracted ray. What is meant by the absorption is the loss of energy, while the ray is being transferred in a transparent object. This loss is negligible when the ray is just reflected and not refracted, since the loss of energy while the ray is traveling in air is very small. But in case of refraction through objects the loss is not that small due to the disturbing influence of molecules in close proximity to each other. If the absorption for a given thickness or concentration is known for the transparent object it is easy to generalize this for other thicknesses and concentrations. The calculations can be made according to two laws, Lambert's law or Beer's law, which in fact state the same result in different contexts.

Lambert's law deals with the relationship of the absorbing medium to the absorption of radiant energy. It states that the fraction of light, which is absorbed is independent of the intensity of the incident light. The main idea is given with the extremely constrained example. It applies to light incident onto the surface at $i = 0^o$, to monochromatic light, and to pure, homogeneous materials (to avoid complications). If the material is made up of n layers, each with a thickness of d, the fraction of energy absorbed by each layer is the same and is denoted by A. Hence, the transmittance through each layer is (1 - A). The light intensity at the end of the object, which is made up of n layers is:

$$\begin{aligned} I &= I_0 * (1 - (A + A(1-A) + A(1-A)^2 + ... + A(1-A)^{n-1})) \\ &= I_0 * (1-A)^n \end{aligned}$$

Notice that the intensity of the light is decreasing exponentially with an increase in the thickness of the medium. Therefore, Lambert's law is mathematically expressed as :

$$I = I_0 e^{-\alpha d} \tag{11}$$

where I is the intensity of the transmitted light, I_0 is the intensity of the incident light, d is the thickness of the medium and α is the absorption coefficient of the medium, which is called the *extinction coefficient* when log_{10} is used instead of ln.

Beer's law states that the absorption of light is directly proportional to the number of molecules in the absorbing substance through which the light passes. The mathematical expression of Beer's law is:

$$log \frac{I}{I_0} = -Acd \tag{12}$$

$$\begin{aligned}\frac{I}{I_0} &= 10^{-Acd}\\ I &= I_0 10^{-Acd}\end{aligned} \tag{13}$$

It can be seen that Beer's law is the same as Lambert's law in mathematical sense. The difference is that Beer has decomposed Lambert's *extinction coefficient* into the coefficients A and c, where A is the extinction coefficient and c is the concentration of the absorbing material.

As a result, Lambert's law can be used to calculate the intensity of the transmitted light by taking appropriate values for the extinction coefficient according to the object on which light is incident.

3.3.3 Intensity of a Light Source

For many years the standard against which intensity was measured was the candle. One candle power represented the luminous intensity of a flame of a certain make of candle. Now the standardized international unit of intensity is the *candela* (cd). The intensity of the light from any source in a particular direction is expressed by a number of candela.

In order to define the intensity of light in space away from the source, it will be necessary to deal with solid geometry and define the *solid angle*. If the light source is envisioned as a point in space, we can imagine a sphere of illumination around it. Since it is important to measure the intensity on the surface of the imaginary sphere, we begin with the definition of the solid angle. Consider a sphere of radius r and a solid angle ω. The part of the sphere s that is enclosed by the conical boundary surface of the solid angle is proportional to the solid angle subtended by s. This is pictorialized in Figure 3.1. When the size of the portion of the spherical surface s is equal to r^2, then the solid angle equals to one steradian (sr). Thus more formally:

$$\omega = \frac{s}{r^2} \tag{14}$$

Thus there are 4π steradians about a point in a complete sphere (area of a sphere $= 4\pi r^2$). Now lets define an other concept, the *luminous flux*. This is

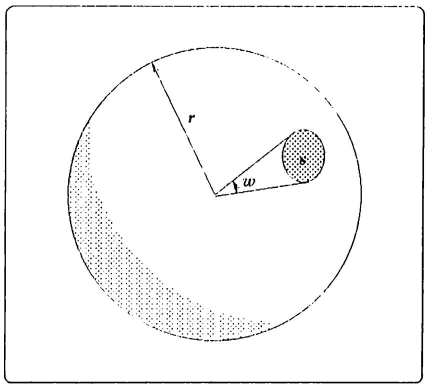

Figure 3.1. The solid angle ω defines an area s on the surface of a sphere

the visible energy from a light source evaluated on the basis of the impression of light which it induces in the eye. The distribution of flux varies directly with the solid angle ω and the luminous intensity I according to:

$$\Phi = I\omega \longrightarrow I = \frac{\Phi}{\omega} \tag{15}$$

Since the flux is distributed homogeneously on the light source, the intensity of the light source at various surface patches varies directly with the solid angle. The flux for a flashbulb gives 1.2×10^6 lm in all directions. So we will be holding the flux at that constant and calculate the intensity of the surface patch with respect to the solid angle.

3.4 Calculation of Colour

The most important mechanism for the production of colour by materials is the selective removal of certain wavelengths of light from the spectrum by absorption. Different than the concept explained in the previous section, this absorption is the absorption of the different wavelengths of frequencies. Colours, as perceived by humans as colours, are in fact lights of different wavelengths and white light is a set containing all the colours that a human can perceive. The pigments on an object act as a filter which only reflect the colours of the object and absorb the other colours. For example, if a white light falls on a red object, only the wavelengths that we perceive as red are reflected, whereas the other wavelengths are absorbed.

However, there are other mechanisms which produce colour in a different way. These are dispersion, interference and Rayleigh scattering. These mechanisms are explained in the following sections.

3.4.1 Primary Colours

Experiments by the English physician Thomas Young show that virtually all colours can be produced from a set of three lights, whose colours are found at widely separated regions of the spectrum. These three colours are red, green and blue. This means that the combination of all of those colours produce white and it is possible to produce all the visible colours by using different combinations of intensities from these primary colours. For example, it is possible to obtain the colour yellow by using same amount of red and green and no blue colour. Of course it is possible to use another set of primary colours, but we have selected these for simplicity. The wavelengths of these colours as measured by a spectrophotometer are as in Figure 3.2, which is taken from Williamson and Cummins [38].

We use these colours with the measurements given in Figure 3.2 as our set of primary colours, since we can obtain all the visible colours by adding or subtracting different combinations.

3.4.2 Dispersion

The separation of white light into colours or equivalently wavelengths by a medium is called dispersion. This fact was experimentally shown in 1762 by Newton, who sent a white light beam to a prism, which decomposed into a spectrum consisting of a large number of colours. The explanation for this well-known fact is that light of all wavelengths travel at different velocities in a transparent object. Since the red light has the (longest wavelength) greatest velocity, it has the least dispersion and the violet light has the least velocity, hence the greatest dispersion. Thus, the colours are divided sequentially between red and violet. This is called the normal dispersion, which is illustrated in Figure 3.3 taken from Brill [5].

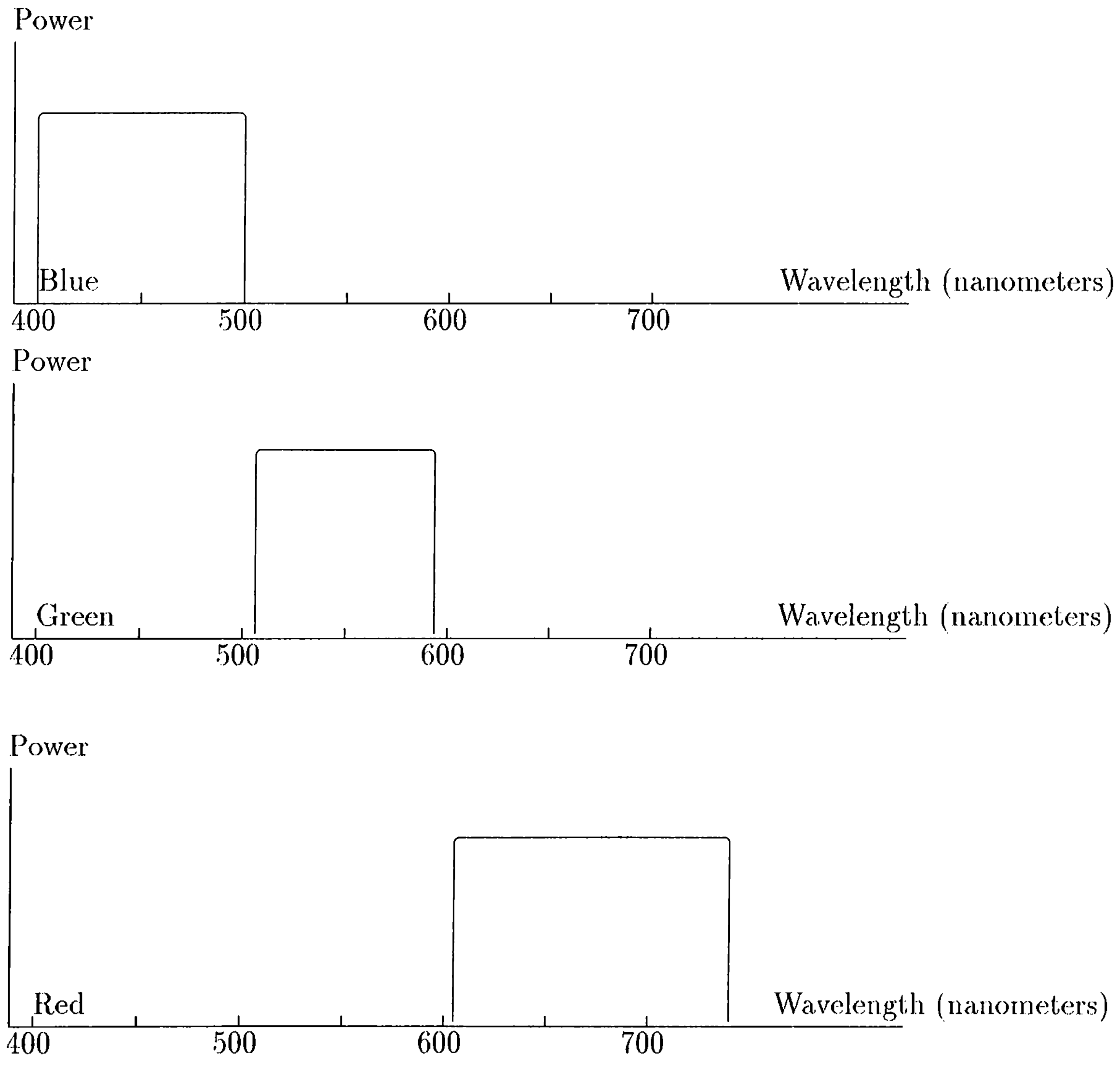

Figure 3.2. Wavelengths of primary colours

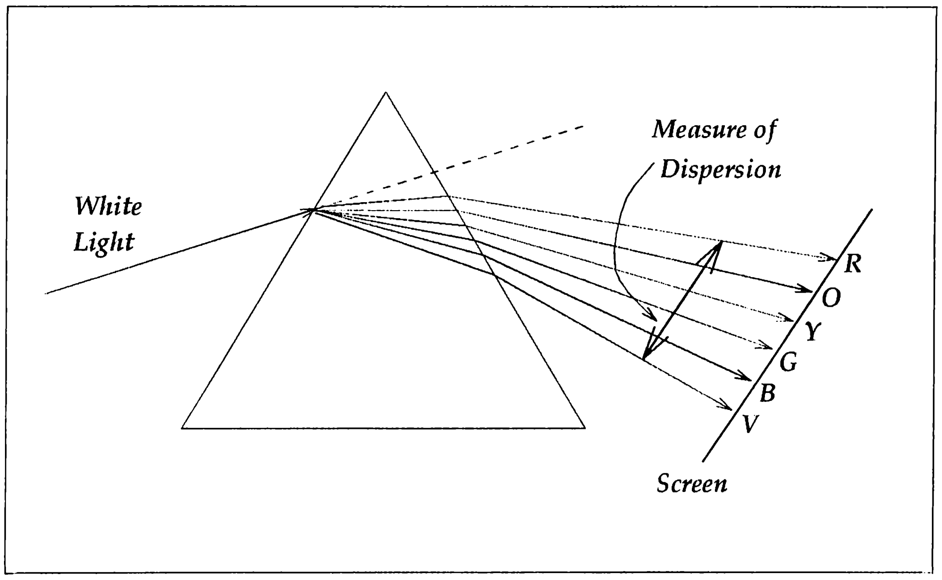

Figure 3.3. The decomposition of visible light into its component wavelength regions (colours) by a glass prism

Since the velocity of lights of different wavelengths differ in a medium, the refractive index of the medium according to these different wavelengths changes. The calculation of the refractive index can be stated as follows:

$$\eta = \frac{V_{light}}{V_{wave}} = \frac{c}{v} = \frac{c}{\lambda \cdot v_t} \tag{16}$$

Here v_t is the *temporal frequency*, which is the number of waves per unit time, λ is the wavelength and c is the velocity of light in air, which is approximately $3 \cdot 10^8$ m/s. From this formula the refractive index of a medium for each colour can be calculated, since the wavelengths are given in the previous section, the only unknown is the *temporal frequency*, which differs according to the mediums nature.

Dispersion of visible light varies with wavelength approximately as $1/\lambda^3$ (Brill [5]). The additional λ^2 comes from the *temporary frequency component*. For this reason the shorter wavelengths show the greatest dispersion ($1/\lambda^3$ is larger for smaller values of λ) and also a much greater rate of change of dispersion for small changes in λ ($1/\lambda^3$ is nonlinear in λ) than do longer wavelengths.

Anomalous dispersion is the dispersion of visible light in coloured media.

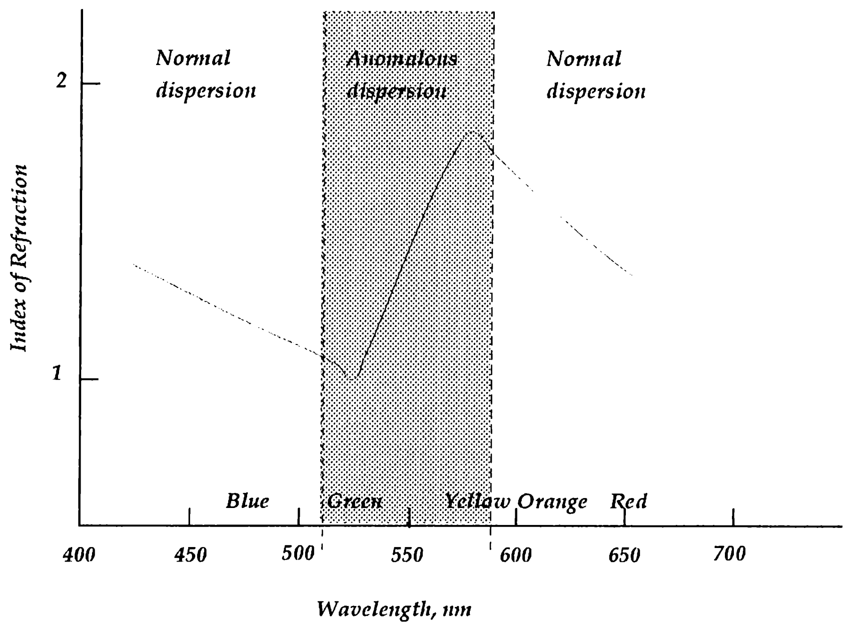

Figure 3.4. Anomalous dispersion

The difference of the anomalous dispersion is that in coloured media, as we have noticed earlier, some wavelengths are absorbed. This causes an absorption band, the absorption band is the interval of wavelengths, which were absorbed by the object (here yellow-green). The effect of the absorption band is that η decreases at the beginning of the absorption band and increases rapidly to the end. Suppose a transparent object having colour implement yellow-green (that is yellow-green wavelengths are absorbed). On the short-wavelength side (the violet through blue colours) η decreases in the normal way, that is the colours disperse in a decreasing manner (since the wavelengths are increasing). However, the decrease of η becomes more rapid as the absorption band is approached. On the long-wavelength side of the absorption band, η takes a larger value than on the short-wavelength side. So the colours on the long-wavelength side may be dispersed more than the ones on the short-wavelength side. Figure 3.4, taken from Brill [5], shows the anomalous dispersion for the given example.

In other words, the anomalous dispersion can have the effect of dispersing the long-wavelength at a greater amount than the short-wavelength. So a coloured media can produce a colour progression of blue-indigo-violet-red-orange-yellow and with green not appearing because it is absorbed. Since the effect of the anomalous band varies with the media that is transmitting the ray, the jump of η in the absorption band is media dependent.

We implement this phenomena by taking a constant jump in the absorption band (according to solid cyanin, where the change is given in [5]).

3.5 Implementation

As mentioned earlier, we used ray tracing as the global illumination technique, since it is suitable due to its ray oriented structure. The implementation details of this technique are left to the next chapter. In this section we explain how the new features such as absorption of light and dispersion are implemented using ray tracing.

Absorption of light is implemented using the fact that the light transmitted through a medium looses a fraction of its intensity. This phenomena is represented by equation 11. Here, the only unknown is the so called *extinction coefficient* α, which is a constant for homogeneous media and is given as an input for each object to be rendered. The value of this coefficient should be positive (approximately 0 for air) and approximately 0.0353 for a medium absorbing 10% of the transmitted intensity. Having found the intersection points of a ray at the biggest depth level, we calculate the intensity at each intersection point by adding the ambient light intensity, the local intensity of the light sources and the intensity due to specular reflection and then subtracting the fraction of the absorbed intensity, since we know the length of the transmission path (the distance from one intersection to the next).

The implementation of dispersion is not as obvious as that of absorption. As ray tracing is a technique in which the rays reaching the eye due to a light source are traced backwards, separated light (as a result of dispersion, the light is separated into different wavelengths) cannot be combined (Rays are shot one by one). Thus, another method must be used to generate this effect. We implemented a new method for abstracting dispersion. In this method the scene is rendered for each RGB colour, which makes up 2 extra rays for each pixel. That is, when we ray trace the scene for the red colour we assume that the light sources only produce red light. For example, for the blue light we first calculate the refraction index of each object. Since the refraction index entered initially for each object is assumed to be measured with light of wavelength 670nm (in measuring refractive indices occasionally the lithium line with wavelength 670nm is used), the new refraction index will be (e.g. a medium made of glass, where the temporal frequencies for red, green and blue colour are $2.86 \cdot 10^{14} Hz$, $3.61 \cdot 10^{14} Hz$ and $4.92 \cdot 10^{14} Hz$ respectively):

$$\eta_{new} = \frac{\eta_{old} \cdot \lambda_{lit} \cdot v_{t_{lit}}}{\lambda_{red} \cdot v_{t_{red}}} = 1.525 \tag{17}$$

From this formula we calculate the new refraction index of each object in the scene and ray trace the scene for each colour, RGB. Thus by ray tracing the scene with respect to each colour we even obtain effects like translucency without extra computation.

To decrease the aliasing effects and discontinuities on the coloured surface, the user may choose to increase the number of rays per pixel which are sent for each colour. Since we use one ray for each colour, we get an average wavelength for each colour (for example 675nm for red). Theoretically, we have to ray trace the scene with each visible wavelength but this would be too costly. The best result with feasible number of rays is obtained if we linearly interpolate the refractive index rather than the wavelength. The variation of the refractive index according to wavelength is nonlinear (approximately $1/\lambda^3$). It makes more sense to shoot a ray with respect to the refractive index, as it is the refractive index that determines the angle of refraction. For example, for glass the refractive index varies between 1.5 and 1.53 (see Figure 3.5).

Thus ray tracing at $\Delta\eta = 0.05$ will give a reasonably good result. This will result in 3 rays for blue, 2 rays for green and 1 ray for the red colour. Finally we will calculate the intensity and colour of each pixel, by simply averaging the calculated intensities for each colour. So using this method we can generate colour in a scene, where there is no colour (in a scene like Figure 3.3).

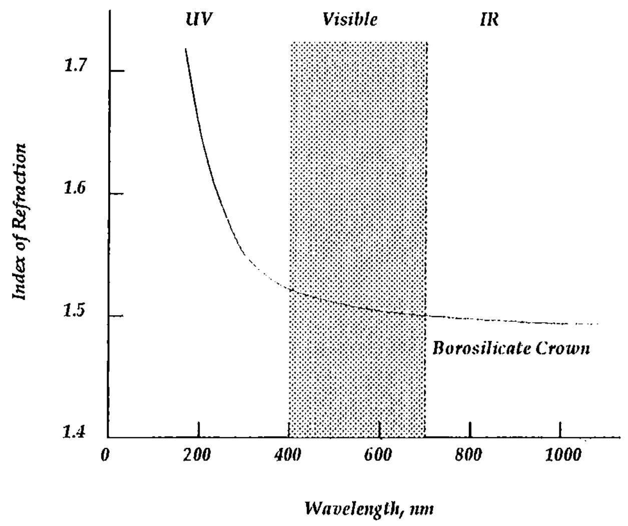

Figure 3.5. The dispersion curve for the Borosilicate Crown material

Chapter 4

Ray Tracing

Ray tracing is a point sampling method in which a picture is generated by tracing rays backwardly from the eye into the scene, recursively exploring reflected and transmitted directions and tracing rays toward point light sources to simulate shading [37]. Ray tracing is one of the most elegant techniques in computer graphics. Many phenomena that are difficult or impossible to model with other techniques are manageable with ray tracing, including shadows, reflections, and refraction of light. However, there are some disadvantages of ray tracing, as mentioned earlier, that have to be overcome:

- Ray tracing is often slow.
- Ray tracing is prone to aliasing artifacts.

Ray tracing is often slow since the intersection calculations are floating point intensive. The most costly part of the ray tracing method is the calculation of the intersections. There are mainly two general strategies for decreasing the number of intersections: hierarchical bounding volumes and space partitioning. In the first approach complicated objects are enveloped with simpler bounding volumes. Intersection tests are done first with bounding volumes (simpler tests) and then the intersection with the real object is done only if it is necessary. Many methods that take advantage of this technique have been implemented ([37], [28], [35], [20]). In the second approach the data space is partitioned into regions or voxels, where each voxel contains a list of objects that are in that voxel. Thus, when a ray enters a voxel, the intersection test is made with only those objects in the partitioned region. Again, several methods exploiting this technique have been proposed to speed up ray tracing ([14], [12]). We used a fast voxel traversal algorithm (space partitioning algorithm) introduced by Amanatides and Woo [1].

Ray tracing is prone to aliasing artifacts as it is a point sampling method. According to Shannon's theorem, for a given sampling rate, every signal which has frequencies beyond the Nyquist limit will alias. Yet, it is too expensive to sample at a rate where the aliasing will be sufficiently small (several hundreds per pixel). Thus, powerful sampling strategies have to be found to reduce aliasing while maintaining low cost.

A recent and most widely used anti-aliasing method is *supersampling*. We used a supersampling based anti-aliasing method which we explain in detail in the implementation section.

In order to evaluate the power of anti-aliasing methods we need a framework. This framework has been constructed by several authors by bringing out some characteristics of an optimal anti-aliasing method [31]. We state these characteristics and evaluate our method in terms of these characteristics below.

Adaptivity: One way to reduce the number of rays is to increment the rays per pixel until the current error in a pixel falls below a predefined treshhold, otherwise the sample number will be default (e.g. 1 per pixel). In our method this will be done until a refinement criterion for one pixel is fulfilled. The criterion for refinement can be based on statistics (e.g. variance in [21], confidence in [27]), signal theory (e.g. signal to noise rate [9]), or some characteristics of the human eye as in our method, i.e. contrast).

Irregularity: Cook [7] showed that irregular sampling achieves better results than regular one, since it replaces coherent aliasing patterns by incoherent broad-band noise that is much less objectionable for human eye. The irregular sampling techniques introduced are poisson sampling [7], jittering [7] and N-Roots sampling [32]. Our method does not have the irregularity property.

Complete Stratification: The main idea of stratification is that when N samples are to be taken in an interval L, complete stratification consists of taking exactly one sample in each stratum (interval length L/N). In our method we fulfill this condition by dividing the pixel into equally sized subpixels.

Importance Sampling: When a weighting function is used in sampling the signal , it is more efficient to sample the signal with a non-uniform density. This notion was introduced by Shirley [32], and he obtained such a sampling by transforming the samples by the inverse of the distribution function associated with the weighting function. We did not incorporate this notion in our method.

Uncorrelation: In uncorrelated sampling the idea is to create a bijection between the strata of a dimension and those of another [18]. The important point is that the bijection must be different for neighbouring dimensions and for neighbouring pixels. This property is useful for methods which are made of many dimensions like the distributed ray tracing method by Cook [8]. As we do not have that many dimensions, we did not adopt uncorrelated sampling.

Fast Reconstruction: Reconstruction is a convolution carried out after the sampling has been done. When sampling is not made uniformly the reconstruction becomes more complex and needs more expensive filters [24]. In our method it is not that expensive since the sampling is done with uniform density (pixels are divided into equal sizes of order 2^N).

As a summary, our implemented model contains the properties of adaptivity, complete stratification and fast reconstruction.

4.1 Implementation

In this thesis we have implemented a ray tracing method which makes use of features of the Distributed Ray Tracing method introduced by Cook [8] and improved by Shirley [32]. It is a simplified method in that we did not implement features like depth of field and motion blurr, which is left as a future work.

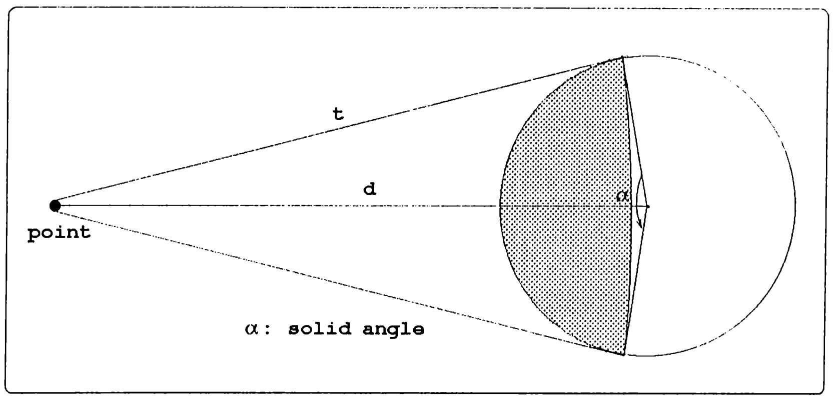

Figure 4.1. Solid angle calculation in shading

In our model the translucency effect, which is handled in [8] by distributing the secondary rays of the reflected and refracted rays with respect to the solid angle, is obtained by the effect of dispersion. Since the refractive index varies for each colour, as explained in Chapter 3, the distribution of the rays for different wavelengths will be achieved automatically.

4.1.1 Implementation of Shading and Penumbras

We implemented the shading concept like in the classic ray tracing method, that is a so called shadow feeler (a ray) is sent from each light source to the intersection point with one difference being that we used extended light sources instead of point light sources. As described in Chapter 3, the intensity of light on a surface depends on the visible parts of the light source. The light sources are abstracted by spheres, assuming that the flux of light is the same in all directions. Hence, the flux of the light is divided into pixels on the surface of the sphere. The solid angle with respect to the intersection point is calculated (see Figure 4.1) and the part of the light source visible to the point is found.

As we have shown in Chapter 3 the effective intensity of the light source on the point is the total flux/solid angle. Hence the total flux is held as a constant (in our implementation 1.2×10^6 lm), the intensity varies with the visible part of the light source. To intersect the shadow feelers we use the same intersection algorithm that we use for the normal rays.

The calculation of the intensity is done in the following way. The distance between the center of the sphere and the point is known (d in Figure 4.1). The length of the tangent to the sphere can be calculated (t in Figure 4.1) so each pixel on the surface of the light source with a distance smaller than t will shoot a ray to the intersection point. If the intersection point is visible, a fraction of intensity will be added to the point's local intensity. Thus, the proportion of lit sample points in a region of the surface is equal to the proportion of that light's intensity that is visible in that region. Note that the light sources will be stored with the location of the pixels on its surface. This methodology will abstract the specular reflection in a more realistic way.

4.1.2 Implementation of the Anti-aliasing Method

Although *distributed ray tracing* is not prone to aliasing effects as much as classic ray tracing, still a need for anti-aliasing arises to generate more realistic pictures. We implemented an adaptive supersampling method, which satisfies the adaptivity, complete stratification and fast reconstruction properties. The main idea of this method is originated by Schlick [31]. Since the smallest feature that can be displayed on the screen is the pixel, the highest frequency of the signal (in image synthesis, signal is only virtual and is not available under its continuous form) should be the pixel frequency. To filter the signal, Shannon's theorem states that four samples per pixel will be sufficient to reconstruct it. Therefore, the main idea is to subdivide into four subpixels and to compute a mean value of the light reaching each sub-pixel. The value of the pixel becomes more precise as the refinement progresses.

The anti-aliasing procedure begins after the intensities for each pixel are calculated (for one colour). The algorithm used is given in Figure 4.2:

```
/* input : a Pixel array of the scene containing intensities of initial evaluation */
/* output : Anti-aliased pixel intensities of the whole scene */
Void $Anti\_alias$
for $j = 2$ to $X\_Max\_Scene - 1$ do
    for $j = 2$ to $Y\_Max\_Scene - 1$ do
        /* The intensity of the center pixel is compared with the intensities
            of the surrounding 8 pixels (8-connectivity) and the two pixels
            not satisfying the condition below are selected*/
        while $\frac{abs(Pixel\_array[i,j]) - Compared\_pixel.colour\_r)}{Pixel\_array[i,j].colour\_r + Compared\_pixel.colour\_r} > Treshhold$
            if $Pixel\_array[i,j].ref\_level > Compared\_pixel.ref\_level$
                $Pixel\_array[Compared\_pixel].ref\_level++$
                $Refine(Pixel\_array[Compared\_pixel])$
                $Reconstruct(Pixel\_array[Compared\_pixel])$
            else
                $Pixel\_array[i,j].ref\_level++$
                $Refine(Pixel\_array[i,j])$
                $Reconstruct(Pixel\_array[i,j])$
endFunction
```

Figure 4.2. Algorithm for adaptive sampling

This algorithm is executed for each wavelength, that is for example for the three colours RGB. The refinement procedure subdivides the entered pixel, according to the refinement level to 4, 16 or 64 and finds the direction of the ray, which will be sent through each new subpixel and calls the ray tracing procedure to find the intensity values for these new subpixels. After the values for these pixels are calculated, the subpixels are reconstructed to form the new intensity of the pixel. If the pixel is again not in the treshhold limit, the refinement level is increased by one and sampling is done by dividing into more subpixels. We used a refinement upper bound to reduce the cost, the bound is given by dividing the pixel at most into 256 (16x16) subpixels. An example of the refinement selection and the illustration of how the pixels are stored are given in Figure 4.3. In this example, the illumination of the center pixel and that of the subpixel on the left of the center pixel are compared and the contrast is found to be out of the treshhold value and the left subpixel is selected, since the refinement level is smaller.

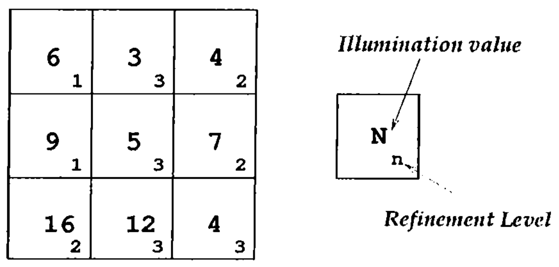

Figure 4.3. An example to refinement selection

The reconstruction function combines the calculated intensity values in a fast and weighted manner using weighting windows. The advantage of the method compared to other methods is that the final samples are evenly spaced, so the reconstruction is only a convolution with a weighting window. Examples of such weighting windows of various sizes are given in Figure 4.4.

$$\frac{1}{4}\begin{bmatrix}1&1\\1&1\end{bmatrix}$$

$$\frac{1}{64}\begin{bmatrix}2&3&3&2\\3&8&8&3\\3&8&8&3\\2&3&3&2\end{bmatrix} \qquad \frac{1}{256}\begin{bmatrix}3&4&5&5&4&3\\4&7&8&8&7&4\\5&8&20&20&8&5\\5&8&20&20&8&5\\4&7&8&8&7&4\\3&4&5&5&4&3\end{bmatrix}$$

Figure 4.4. Examples of weighting windows

4.1.3 Calculating Intersections

In our implementation we used a space partitioning ray tracing algorithm. The algorithm implemented is introduced by Amanatides and Woo [1] and is a well known fast voxel traversal algorithm. The reason for using this algorithm is the high speed and the simplicity of the algorithm. The path from one voxel to its neighbour requires only two floating point comparisons and one floating point addition.

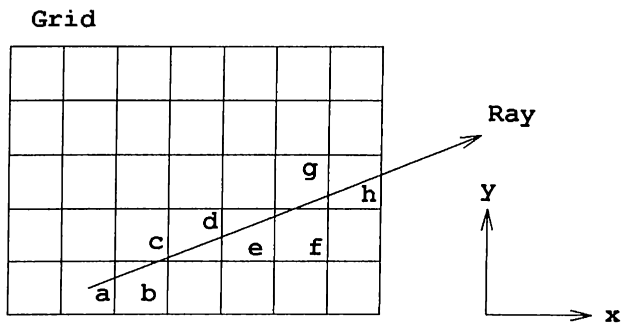

Figure 4.5. Traversal of a ray in two-dimensional grid

The traversal of a ray in two dimensions is shown in Figure 4.5. The 3D version of the algorithm is very similar in implementation, so we explain the algorithm on the two dimensional case. In Figure 4.5 the shown ray must visit the voxels a, b, c, d, e, f, g in that order. The algorithm has two main operations:

1) the determination of the voxel in which the ray origin is found and

2) the determination of the value, where the ray crosses the first vertical voxel boundary.

In the first operation, if the ray origin is outside the grid, the point where the ray enters the grid is found. The variables StartX and StartY show the starting voxel and the variables StepX and StepY are either 1 or -1 (showing whether X and Y are incremented or decremented, while crossing voxels). In the second operation a similar procedure is done in the y direction and the results are stored in the variables BoundX and BoundY. The minimum of these values indicates how much the ray can travel in the current voxel. Finally the two variables DeltaX and DeltaY indicate the amount of movement along the ray in the X and Y directions. The algorithm implemented is given in Figure 4.6.

```
/* input : Direction of the ray, grid size, objects in each voxel */
/* output : The intersection of the ray with an object or no intersection */
Void Traverse
while (intersect == NIL)
    if (BoundX < BoundY)
        if BoundX < BoundZ
            X += StepX
            if X is outside grid then return(NIL)
            BoundX += DeltaX
        else
            Z += StepZ
            if Z is outside grid then return(NIL)
            BoundZ += DeltaZ
    else
        if BoundY < BoundZ
            Y += StepY
            if Y is outside grid then return(NIL)
            BoundY += DeltaY
        else
            Z += StepZ
            if Z is outside grid then return(NIL)
            BoundZ += DeltaZ
    intersect = ObjectList[X][Y][Z]
return(list)
endFunction
```

Figure 4.6. Voxel traversal algorithm

4.1.4 The General Model

The model is made up mainly of two parts. The first part is the "Texture Generator," which generates 3D textures according to the techniques explained in Chapter 2. The second part is the "Ray Tracer" having the features explained in the previous sections. After the texture is generated, the data of the texture and the object on which the texture will be mapped is given as an input to the "Ray Tracer." The input data for the object and the texture are their index of refraction, the extinction coefficient and the ambient intensity. Note that the given object can be any complex object, represented by triangular patches. Such objects have been generated using the object modeler by Erkan [10]. In this model objects are created by rotating a curve about an axis of rotation. Thus, complex objects which are difficult to formulate can be generated and abstracted. After the object is created and triangulated, the 3D texture is mapped on the object and the scene is rendered with the ray tracing algorithm given in Figure 4.7.

The colour information of the object is determined via the ambient coefficient, which is given as input. The other coefficients used to determine the pixel intensity are calculated with respect to the refraction index and the extinction coefficient, which show the nature of the textured object and the nature of the texture. The specular part of the intensity calculation is rather a heuristic abstraction, since one of the two coefficients which determine the angle of specularity is taken as a constant in the program and may not reflect the actual value. Other natural phenomena like dispersion, absorption of light and opacity are implemented as explained previously in Chapter 3. Thus, after the image is rendered the RGB values of each pixel are written to an RGB file. Note that the final intensities are calculated taking the average of the evaluated intensities for the different refraction indexes as a result of the dispersion effect.

```
/* input : Direction of the ray, intersection point & object no, Refraction
index and ext. coef. of the environment from which the ray comes from, depth
and intersection point*/
/* output : Intensity of the corresponding pixel from which the ray is shot */
Void Ray_Trace
if (depth > Max_depth)
    reflected_colour = transmitted_colour = black
else Calc_Direction(Direc_ray, Object[objectno].normal,
    prev_eta, Object[objectno].eta, &reflected_dir, &transmitted_dir);
    k_reflected = $\frac{(abs(prev\_eta-Object[objectno].eta))}{(prev\_eta+Object[objectno].eta)} \cdot Object[objectno].k\_global$;
    k_transmitted = Object[objectno]k_global - k_reflected;
    if (k_reflected > 0)
        Traverse(reflected_dir, prev_eta, prev_alp, depth++,inter,&length,
        &reflected_colour);
    else reflected_colour = 0;
    if (k_transmitted > 0)
        Traverse(transmitted_dir, Object[objectno].eta, Object[objectno].alp,
        depth++, inter,&length,&transmitted_colour);
    else transmitted_colour = 0;
k_specular = $\frac{(abs(prev\_eta-Object[objectno].eta))}{(prev\_eta+Object[objectno].eta)}$;
Intersect_light(inter, &light_direction, &dist, k_specular);
Calculate the specular intensity and the diffuse intensity
pixel_colour = Object[objectno].k_global·(transmitted_colour
    - EXP(2·$Object[objectno].\alpha$·length·transmitted_colour + reflected_colour
    - EXP(2· prev_alp·length·reflected_colour);
    + Object[objectno].k_local·(diffuse_colour + ambient_colour +
                    specular_colour);
endFunction
```

Figure 4.7. Ray Tracing algorithm

4.1.5 Results

To get more insight to the efficiency of the Rendering algorithm, we extracted some results using the scene in Figure A.4. The rendering of the figure is done in a 400x400 image resolution, and with a grid size of 20x20x20. The results are given in the following table (time is measured as seconds):

Anti-alias treshhold	Depth of Ray Tracing	Rendering Time	Anti-alias Time	Effect of dispersion	Total Time
1.0	6	42.03	0	N/A	81.03
0.8	6	43.90	0	N/A	82.95
0.6	6	42.36	0.1	N/A	83.1
0.4	6	42.68	5.0	N/A	89.17
0.2	6	42.67	18.87	N/A	102.1
0.1	6	42.89	27.19	N/A	112.5
0.05	6	42.64	44.9	N/A	128.54
0.02	6	42.71	93.96	N/A	174.31
0.6	7	48.20	0.12	N/A	88.1
0.6	8	54.05	0.14	N/A	93.19
0.6	9	62.46	0.30	N/A	101.96
0.610	10	74.99	0.34	N/A	115.13
0.02	6	42.67	95.23	430.248	614.148

Chapter 5

Conclusion

In this thesis, we proposed a method for volume rendering texture mapped dielectrics using the global illumination technique ray tracing. In abstracting dielectrics we implemented the natural phenomena of effects of materials on light such as dispersion, anomalous dispersion, absorption of light and opacity. In the model these effects are implemented in such a way that the user is required only to enter natural information about the objects in the scene (refraction index and extinction coefficient). Properties of objects, such as opacity, are implemented in an inherent way, which are abstracted in many models using several coefficients. By incorporating these natural phenomena to our model, we generated more realistic images using a backward mapped volume rendering algorithm.

The developed method is capable of texture mapping any 3D texture (represented by triangles) on any complex object (represented by triangles). The objects to be texture mapped can be modelled as sweep objects. 3D textures are generated via a "Texture Generator", which generates 3D textures according to bombing, deformed projection, Fourier synthesis and 2D texture projection methods.

The rendering method is implemented by adding features like anti-aliasing and speeding up the ray trace algorithm using a space partitioning algorithm. An adaptive supersampling anti-aliasing algorithm which satisfies the properties of complete stratification, adaptivity and fast reconstruction is implemented. The algorithm is constructed by shooting more rays from one pixel, if the need arises (If the pixels' intensity value is greater than a predefined treshhold). The need for anti-aliasing arises, since the algorithm deals with objects represented by triangular patches. The space partitioning algorithm increases the speed of the algorithm, by space partitioning the data space and as a result, decrementing the number of intersection calculations between objects and rays. Another important feature of the rendering algorithm is that the scene is rendered using extended light sources. With this method natural phenomena like penumbra are implemented as well.

The implementation of dispersion and the feature of texturing objects using fully defined geometric textures causes texturing to be no more a low-cost, simple process. The effect of dispersion results in at least 2 more rays (one for each basic colour, RGB) shot from each pixel, which triples the execution time. In addition, the anti-aliasing algorithm increases the execution time also, since it is a supersampling technique which when necessary can shoot 36 rays from one pixel. Although these factors effect the execution time of the implementation, they are necessary for obtaining more realistic pictures. The current version of the implementation is efficient, such that even with the existence of these factors the execution time results are feasible. This efficiency is achieved mainly by the use of the acceleration algorithm introduced by Armanatides and Woo [1]. But the algorithm can be improved if parallelization is implemented. The structure of the algorithm is very suitable for parallelization, since the dispersion effect is calculated for each different wavelength in an independent manner. Thus, these calculations can be done on different processors of a parallel machine.

As future work new 3D texture generation methods can be developed, which enable the modelling of the generated textures, so that volume rendering can be possible. Another point is the natural phenomena of Rayleigh scattering, which is not implemented and can be added to the available system.

Appendix A

Sample Images

In this appendix, some sample images generated by the explained model, are given. The sample images have a final 400x400 resolution, where the resolution from which the scene is generated depends on the treshhold value of the adaptive anti-aliasing method. The images have been produced on IRIS Indigo[1]. The grid subdivision is 20 in all figures.

[1]IRIS Indigo is a registered trademark of Silicon Graphics,Inc.

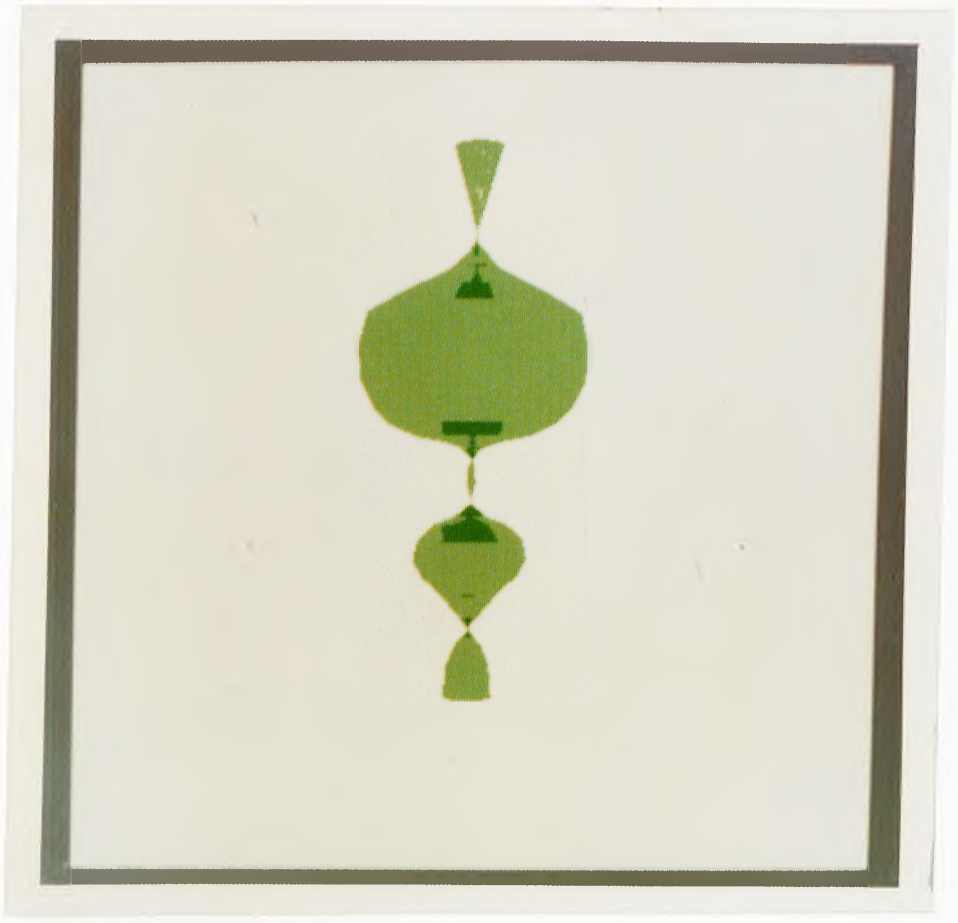

Figure A.1. A texture generated with deformed projection

This is a texture generated with deformed projection (with one object, a circle). The texture is approximated by triangular patches (2500 patches), where the triangles are calculated from NURBS approximations. In the next figure this texture is mapped onto a sphere.

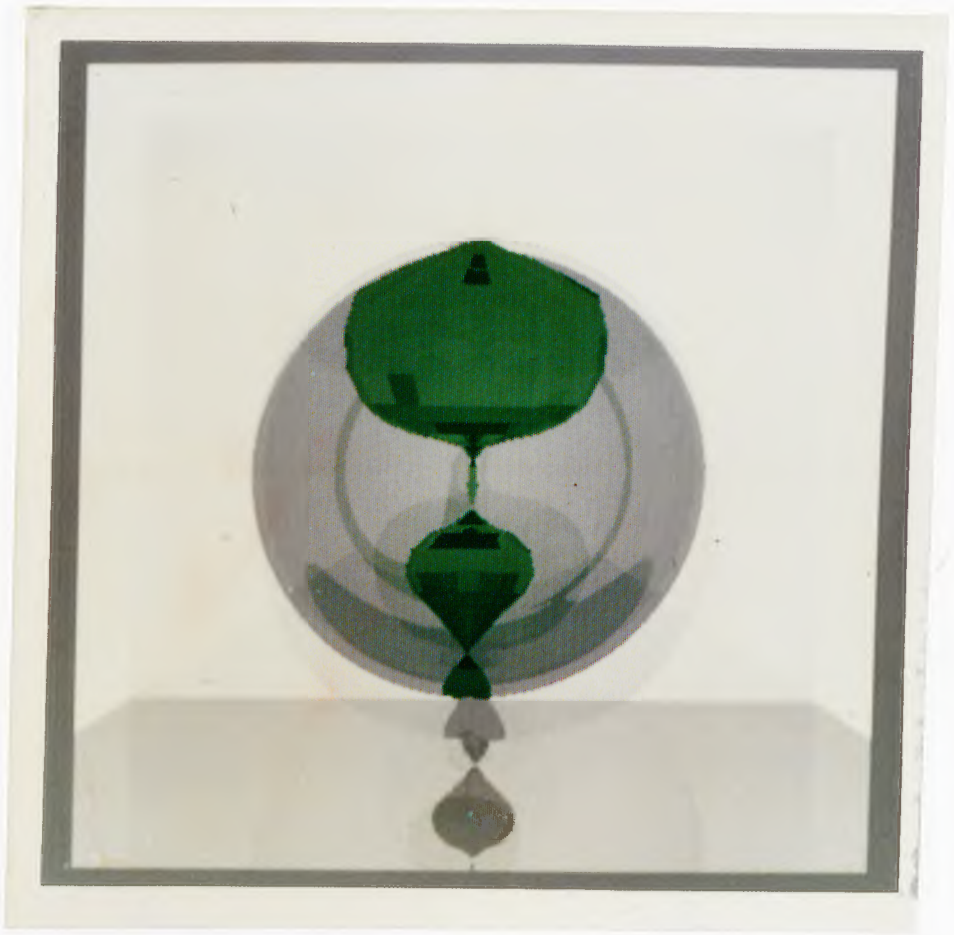

Figure A.2. A marble

In this figure the previous texture is mapped, and the scene is volume based rendered. The scene is made up of 4 objects, 2 make up the background (4 total triangle patches), one is the mapped sphere and the fourth is the texture (2500 patches). The refraction index of the sphere is 1.501.

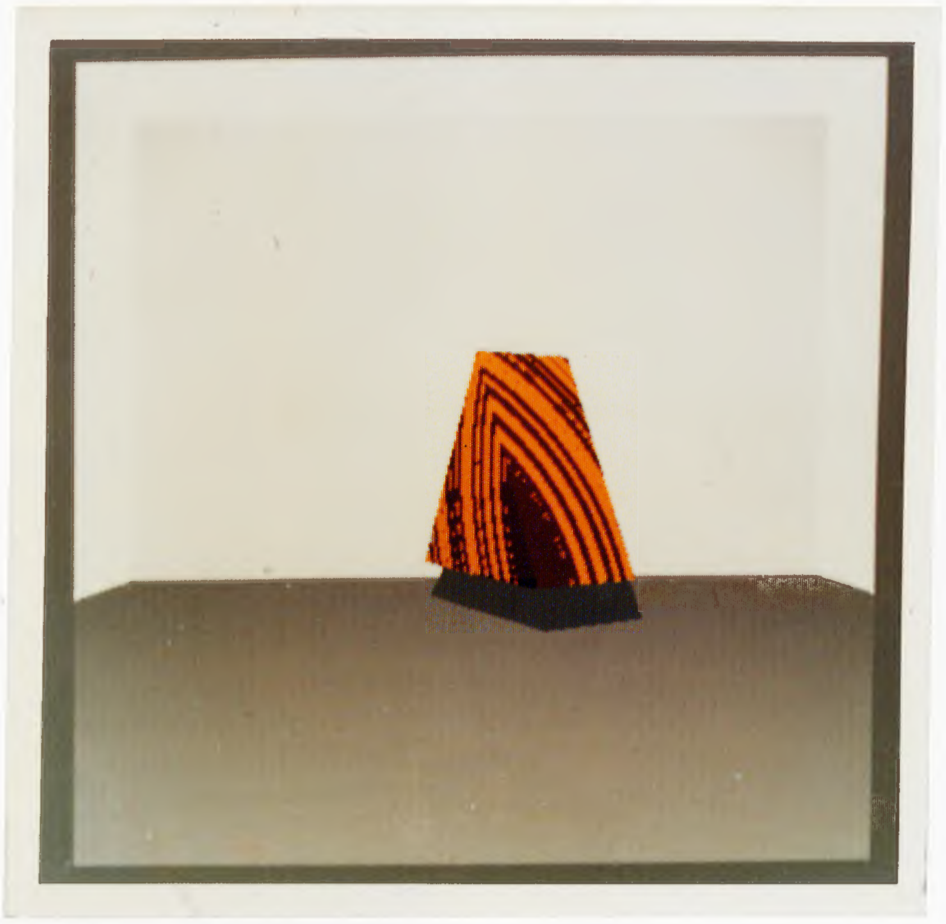

Figure A.3. A wood textured object, where the wood texture is generated with deformed projection

The wood texture is generated using deformed projection with 15 embedded circles. The texture is made off 2500 triangular patches.

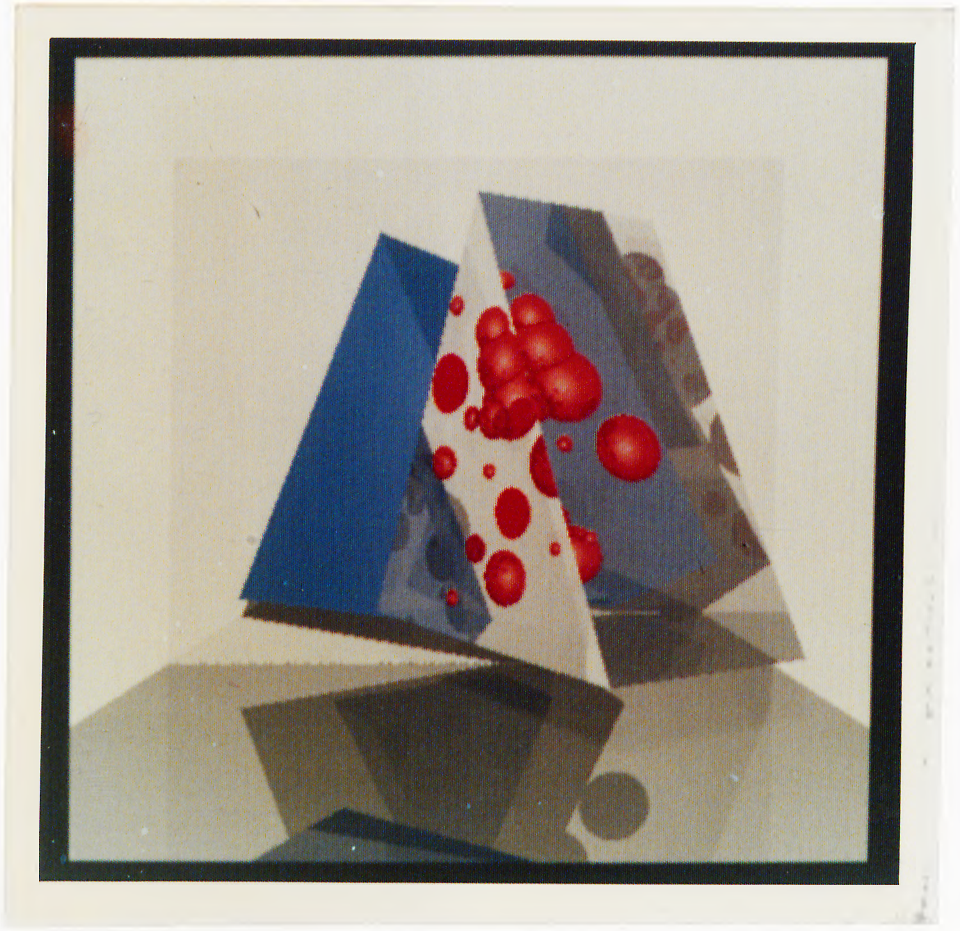

Figure A.4. A scene with two prisms, where the front one is texture mapped with bombing and is transparent, whereas the other one is non-transparent

The scene consists of 34 objects (2 background, 2 prisms, 30 bombed spheres) and is rendered with anti-aliasing treshhold 0.1 and there is no dispersion effect. The refraction index of the transparent prism is 1.501 and of that of the solid one 3.227.

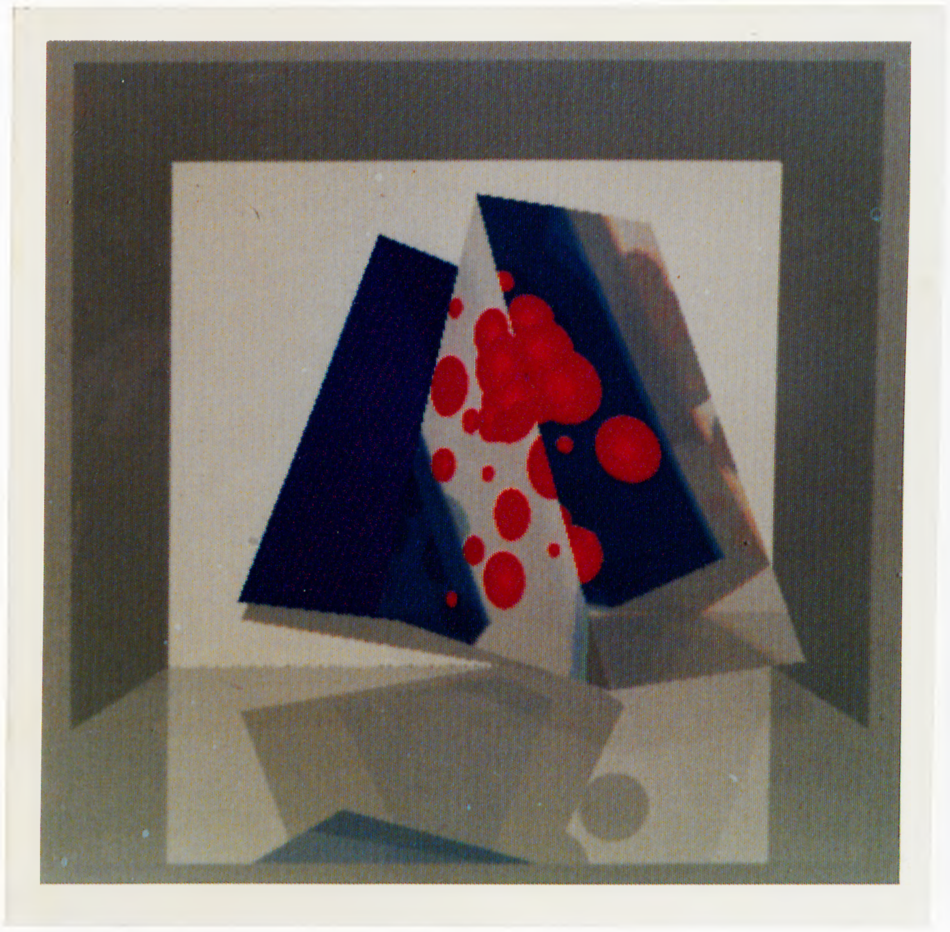

Figure A.5. A scene with two prisms, where the front one is texture mapped with bombing and is transparent, whereas the other one is non-transparent

The same scene as in the previous figure, but with the dispersion effect. Here the light source have less intensity to observe the dispersion effect better.

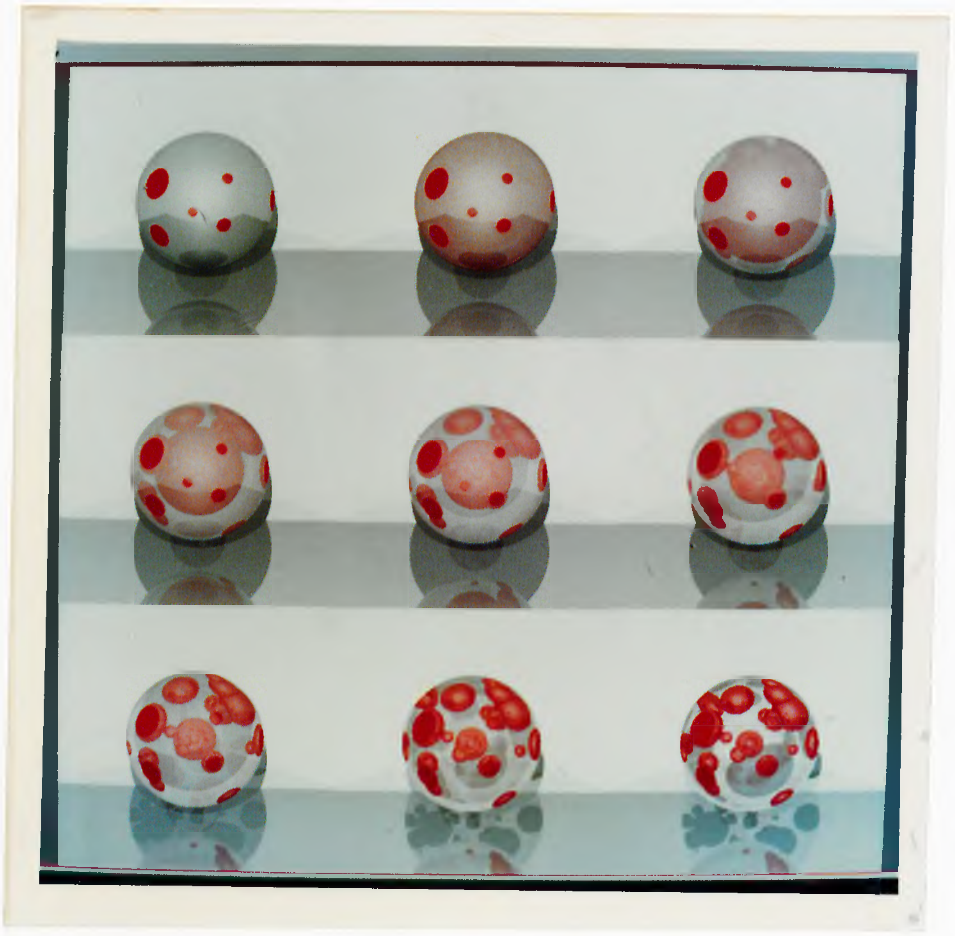

Figure A.6. The role of refractive index in dielectrics (animation)

In this scene there is a bombed textured sphere (with spheres) whose refractive index is decreasing. In the first frame it is solid ($\eta = 3.7$), then in the following frames the refractive index decreases uniformly till 1.501 (refractive index of glass), so the transparency increases and the opacity of the texture increases. In the scene the dispersion effect is not incorporated.

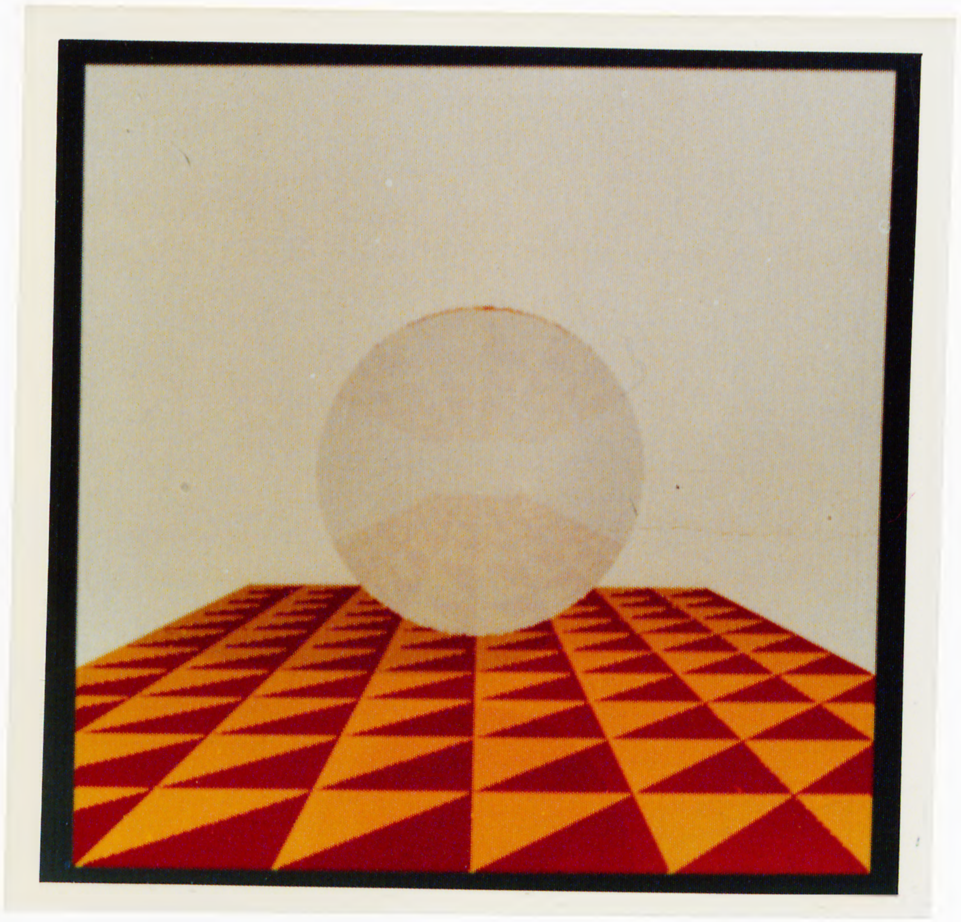

Figure A.7. The effectiveness of the adaptive supersampling anti-aliasing method

In this scene there are 3 objects (total 224 triangular patches). In rendering this scene the anti-aliasing treshhold was given as 0.02 (that is 5 pixel intensity difference is erroneous). The execution time of the algorithm to generate this figure is 52 minutes.

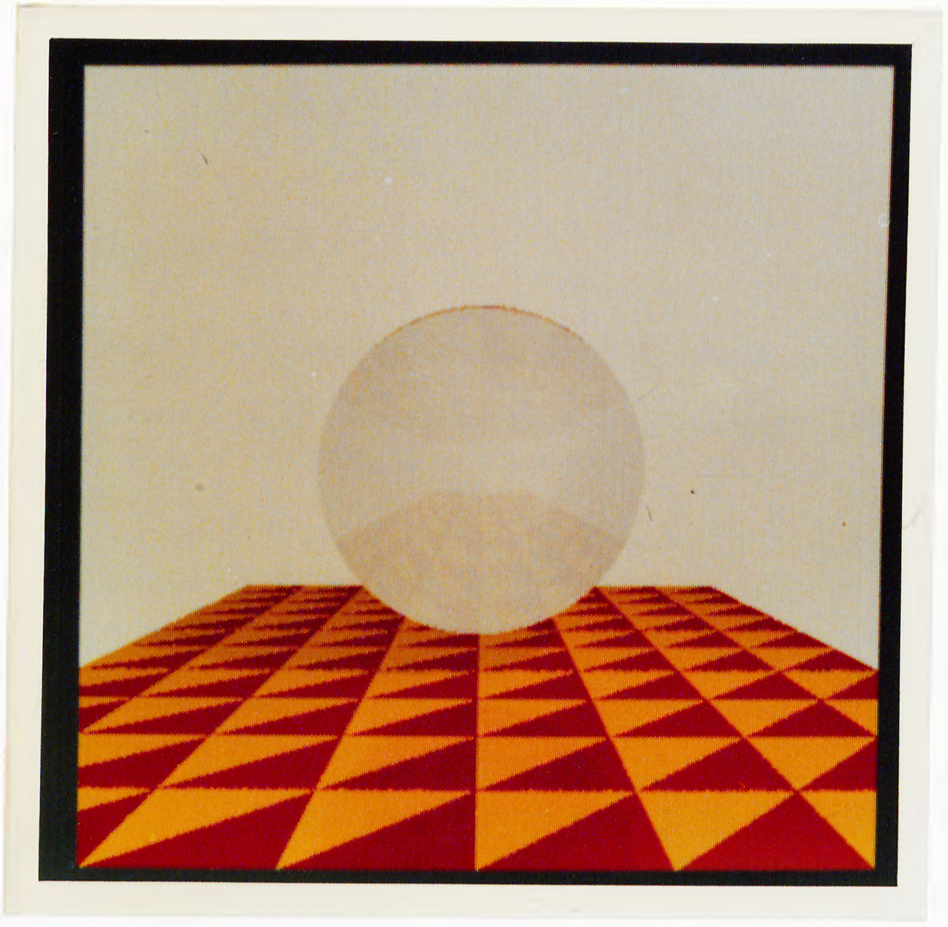

Figure A.8. The effect of aliasing in point sampled ray tracing

The same figure without anti-aliasing. Again there is no dispersion. The execution time of the algorithm to generate this figure is 17 minutes.

Bibliography

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. *EUROGRAPHICS'87, G. Marechal (Editor)*, pages 3 – 10, 1987.

[2] J. Blinn. *Computer Display of Curved Surfaces.* PhD thesis, University of Utah, Dept. of Computer Science, 1978.

[3] J. Blinn and M. Nevell. Texture and reflection on computer generated images. *Communications of the ACM*, 19(10):47 – 58, 1976.

[4] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics (Proc. SIGGRAPH 82)*, 16(3):21 – 29, 1982.

[5] T. B. Brill. *Light: Its Interaction with Art and Antiquities.* Plenum Press, 1980. New York.

[6] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, Dept. of Computer Science, 1974.

[7] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51 – 72, 1986.

[8] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):137 – 143, 1984.

[9] M. A. Dippe and E. H. Wold. Antialiasing through stochastic sampling. *Computer Graphics (Proc. SIGGRAPH 85)*, 19(3):69 – 78, 1985.

[10] S. B. Erkan and B. Özgüç. Object oriented motion abstraction. *The Journal of Visualization and Animation*, 5:1 – 17, 1994.

[11] E. A. Feibush, M. Levoy, and R. L. Cook. Synthetic texturing using digital filters. *Computer Graphics (Proc. SIGGRAPH 80)*, 14(3):294 - 301, 1980.

[12] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 5(2):16 - 26, 1986.

[13] G. Y. Gardner. Simulation of natural scenes using textured quadratic surfaces. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):11 - 20, 1984.

[14] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15 - 22, October 1984.

[15] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623 - 628, June 1971.

[16] J. F. Greenleaf, T. S. Tu, and E. H. Wood. Computer generated 3-d oscilloscopic images and associated techniques for display and study of the spatial distribution of pulmonary blood flow. *IEEE Trans. Nucl. Sci.*, NS-17:353 - 359, 1970.

[17] P. S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):321 - 332, 1986.

[18] J. T. Kajiya. The rendering equation. *Computer Graphics (Proc. SIGGRAPH 86)*, 20(3):143 - 150, 1986.

[19] J. T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):165 - 174, 1984.

[20] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *Computer Graphics (Proc. SIGGRAPH 86)*, 20(4):269 - 278, August 1986.

[21] M. E. Lee, R. A. Redner, and S. P. Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics (Proc. SIGGRAPH 85)*, 19(3):61 - 67, 1985.

[22] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29 - 37, 1988.

[23] E. W. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):163 - 169, July 1987.

[24] D. P. Mitchell. Generating antialiased images at low sampling densities. *Computer Graphics (Proc. SIGGRAPH 87)*, pages 65 – 72, 1987.

[25] D. R. Peachey. Solid texturing of complex surfaces. *Computer Graphics (Proc. of SIGGRAPH 85)*, 19(3):279 – 286, 1985.

[26] K. Perlin. An image synthesizer. *Computer Graphics (Proc. of SIGGRAPH 85)*, 19(3):287 – 296, 1985.

[27] W. Purgathofer. A statistical method for adaptive stochastic sampling. *Proc. of Eurographics 86*, pages 145 – 152, 1986.

[28] S. M. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex. *Computer Graphics (Proc. SIGGRAPH 80)*, 14(3):343 – 349, 1980.

[29] G. Sakas and B. Kernke. Texture shaping: A method for modeling arbitrarily shaped volume objects in texture space. In P. Brunet and F.W. Jansen, editors, *Photorealistic Rendering in Computer Graphics, Proceedings of the Second Eurographics Workshop on Rendering*, pages 207 – 218. Springer-Verlag, 1994.

[30] B. J. Schachter and N. Ahuja. Random pattern generation processes. *Computer Graphics Image Processing*, 10:95 – 114, 1979.

[31] C. Schlick. An adaptive sampling technique for multidimensional integration by ray-tracing. In P. Brunet and F.W. Jansen, editors, *Photorealistic Rendering in Computer Graphics, Proceedings of the Second Eurographics Workshop on Rendering*, pages 21 – 29. Springer-Verlag, 1994.

[32] P. S. Shirley. Physically based lighting calculations for computer graphics. Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1990.

[33] K. H. Tuy and L. T. Tuy. Direct 2d display of 3d objects. *IEEE Computer Graphics and Applications*, 4(10):29 – 34, 1984.

[34] M. W. Vannier, J. L. Marsh, and J. O. Warren. 3d computer graphics for craniofacial surgical planning and evaluation. *Computer Graphics (Proc. SIGGRAPH 83)*, 17(3):263 – 273, 1983.

[35] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52 - 69, January 1986.

[36] L. Westover. Interactive volume rendering. *Proc. of the Chapel Hill Workshop on Volume Visualization*, pages 144 - 153, May 1989.

[37] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23:343 - 349, 1980.

[38] S. J. Williamson and H. Z. Cummins. *Light and Color in Nature and Art.* John Wiley and Sons Inc., 1983.