PARALLEL RAY TRACING OF NON UNIFORM RATIONAL B-SPLINE SURFACES

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE

By

Geoffrey M. Cook

Department of Computer Science

October 1992

Contents

Abstract

UNIVERSITY OF MANCHESTER

ABSTRACT OF THESIS submitted by **Geoffrey M. Cook** for the Degree of Master of Science and entitled **Parallel Ray Tracing of Non Uniform Rational B-Spline Surfaces**

Month and Year of Submission: October 1992

ABSTRACT

Non-Uniform Rational B-Splines (NURBS) are commonly used to represent curves and surfaces because of their excellent design properties. This thesis investigates the implementation of a realistic imaging technique called ray tracing for NURBS surfaces. This centres around a study line/NURBS intersection methods. Several programs were written in conjunction with the thesis using different intersection techniques, the implementations are discussed and results compared. The thesis also investigates the use of parallelism to help speed up the ray tracing process.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Acknowledgements

I would like to thank for their help and support my supervisor Terry Hewitt, and the rest of the Manchester Computer Graphics Unit : Tony Arnold, Neil Gatenby, Andrew Grant, Steve Larkin, Fenqiang Lin, Mary McDerby, Martin Preston and David Yip.

Chapter 1

Introduction

Ray tracing is currently the most popular and successful technique for producing photo realistic images of three dimensional scenes. It is based on an idealized mathematical model of light ray propagation. The technique despite its simplicity can handle very complex effects such as hidden surface removal, reflection, refraction and shadowing.

Each object in a three dimensional computer scene is represented within the computer. This leads to a problem of what form to store them in. Obviously storing each surface point would be impractical, thus some sort of compressed form must be found. Commonly in ray tracing packages the primitive types sphere and polygon are used, the reason for which is the simple mathematics associated with these primitives. Unfortunately to attain a satisfactory representation of an object a large amount of these primitive objects normally has to be used, this leads to a large store requirement.

This project will investigate techniques of ray tracing mathematical shapes called

Non Uniform Rational B-Splines (NURBS). These provide a number of advantages over using the above mentioned primitives.

- They use relatively small amounts of memory.
- They are easy to design with.
- You can define a set continuity over the surface.
- You can represent conic sections exactly.

A general problem with the ray tracing procedure itself is that it takes a long time, several hours to produce a complicated picture is not unusual. The project will look into ways of speeding up the process. The main method of which will be the utilization of parallel processing.

Chapter two introduces mathematical shape representation, it ends with a definition of NURBS surfaces. The next chapter describes the theory of ray tracing, and gives some practical models. The fourth chapter describes various ways of ray tracing NURBS, and discusses their relative merits. The fifth describes how parallel processing can be used to speed up the ray tracing technique. Several programs were written as part of this thesis to demonstrate some of the techniques described, the way they were implemented and some of the results achieved are given in chapters six and seven. The last chapter concludes by summarizing the NURBS ray tracing methods discussed and gives some improvements that could be added to the NURBS ray tracing programs written.

Chapter 2

Mathematical Shape Representation

2.1 Parametric description

Any shape S in n dimensional space can be described in the following form

$$\mathbf{S}(\mathbf{V}) = (F_0(\mathbf{V}), F_1(\mathbf{V}), .., F_p(\mathbf{V}))$$

where

- $\mathbf{V}$ is a vector of parameters that can vary their value individually over a particular range. The number of components in $\mathbf{V}$ controls the dimension of the shape i.e., if one parameter a curve is described, if two a surface etc.
- $F_0, F_1, .., F_p$ are functions that return a value given $\mathbf{V}$. They compositely represent a position on the shape for a given $\mathbf{V}$. The number of functions controls the dimension the shape is in. If two functions are given the shape

can lie in two dimensional space, if three are given the shape can lie in three dimensional space etc.

This is called a parametric description. Note there can be many parametric forms representing one shape. For example a unit circle centered about the origin can be described as

$$(cos(\theta), sin(\theta)) \quad 0 \leq \theta \leq 2 * \pi$$

it can also be defined as

$$((1-t^2)/(1+t^2), 2t/(1+t^2)) \quad t\varepsilon\mathcal{R}$$

Which form we use to represent a particular shape depends on the application.

The rest of this chapter describes some general curve and surface parametric description forms commonly used in computer aided design and computer graphics.

2.2 General curve description

2.2.1 Polynomial curve

A simple way to represent a curve is via a polynomial. This can be achieved by setting the F functions to :

$$F_i(t) = a_{0,i} + a_{1,i}t + a_{2,i}t^2 + .. + a_{n,i}t^n \quad t_{min} \leq t \leq t_{max}$$

where

- n is an integer defining the degree of the polynomial
- $F_i(t), 0 \leq i \leq p$, are functions which return a ordinate.
- $a_{j,i}$, $0 \leq i \leq p$,$0 \leq j \leq n$, are constants.

If all the F functions are combined the curve can be represented as

$$\mathbf{C}(t) = \mathbf{a_0} + \mathbf{a_1}t + \mathbf{a_2}t^2 + .. + \mathbf{a_n}t^n \quad t_{min} \leq t \leq t_{max}$$

where

- $\mathbf{a_j}$, $0 \leq j \leq n$, is a vector with the components $a_{j,0}, a_{j,1}, .., a_{j,p}$.

This defines a curve via a polynomial of degree n.

There are several problems with this representation

1. Not intuitive in design

   Altering one of the control vectors, $\mathbf{a_j}$, does not produce an intuitive change in the shape of the curve.

2. Instability

   High order polynomials are often needed to represent complex curve segments this can cause instability in computational algorithms. e.g., drawing algorithms.

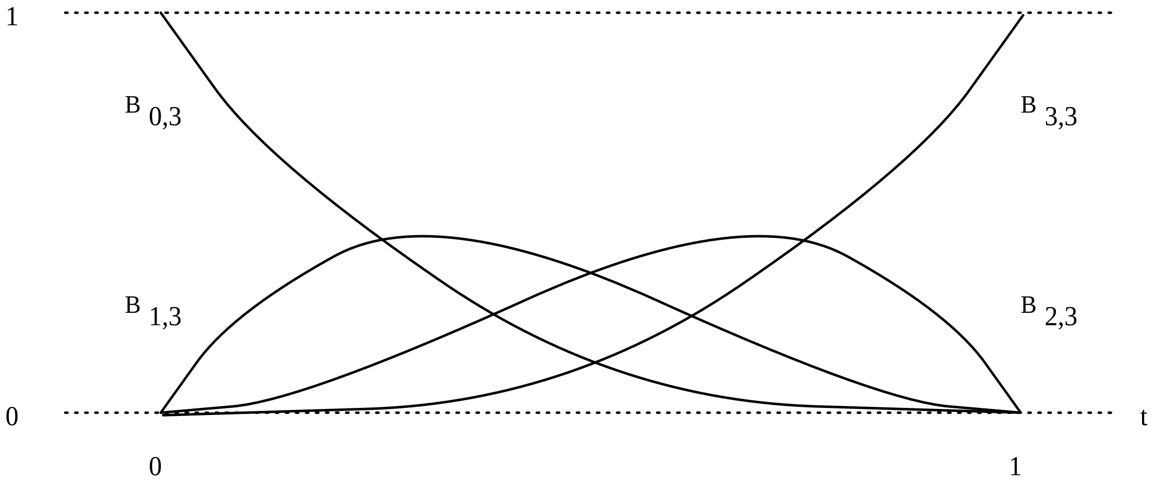

Figure 2.1: The cubic Bernstein polynomials

2.2.2 Bézier curve

Bernstein-Bézier curves attempt to make the designing process easier, An n'th degree Bézier curve is defined as follows.

$$\mathbf{C}(t) = \sum_{i=0}^{n} \mathbf{P_i} B_{i,n}(t)$$

where

- $\mathbf{P_i}$ are called the control points, the number of components of which relate to the dimension of the curve.
- $B_{i,n}(t) = C_i^n t^i (1-t)^{n-i}$ (Bernstein polynomials)

The Bernstein polynomials act as a set of merging functions to the control points. The cubic Bernstein polynomials are shown in fig 2.1.

The following facts about Bézier curves can be proved [11].

1. The curve always passes through the first and last control points

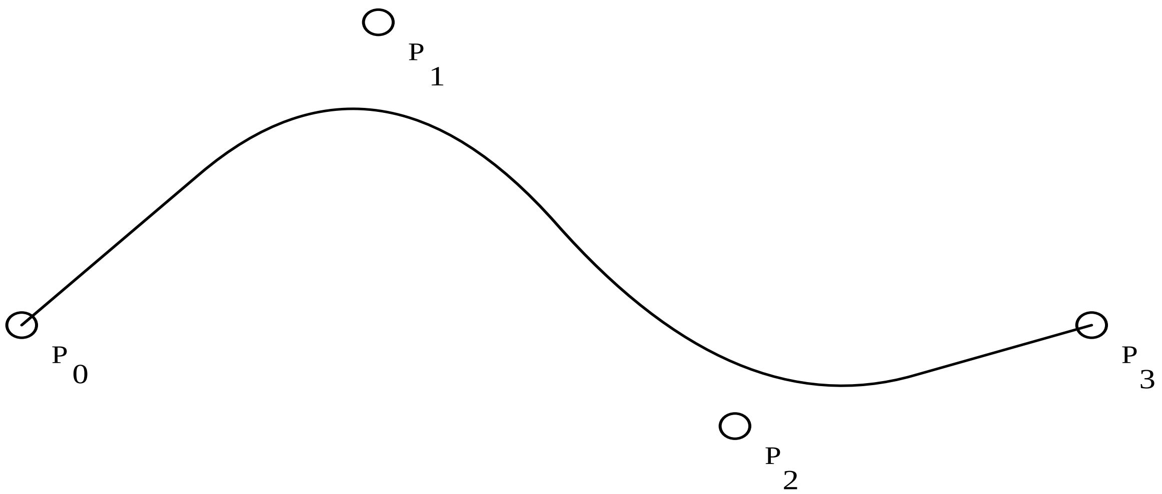

Figure 2.2: A cubic Bézier curve

2. The curve lies within the convex hull of the control points.

3. The gradient, at $t = 0$, to the curve is that of the line $\mathbf{P_1} - \mathbf{P_0}$.

4. The gradient, at $t = 1$, to the curve is that of the line $\mathbf{P_d} - \mathbf{P_{d-1}}$.

All these points, plus the fact that each control point is gradually given some control of the curve near the region it is defined, allows an intuitive curve definition from the control points. A cubic Bézier curve with its control points is shown in fig 2.2.

Some other important facts about Bézier curves :

- The Bernstein polynomials of degree d form a basis for the power polynomials of degree d. Hence easy conversion between the Bézier and polynomial formats is possible.

- Bézier curves are axis independent. i.e., A Euclidean transformation (a rotation followed by translation operation) of the curve can be done by performing the transformation on the control points. The same is also true for the scaling operation. These are very important properties in Computer Aided Design (CAD) applications. The polynomial representation does not have this property.

- No line has more intersections with the curve than with the control polygon (a polygon formed by joining up the control points linearly). This is called the variation diminishing property.

- The control polygon represents a piecewise linear approximation to the curve. The accuracy depends on the length of the curve segment and the number of control points used to represent it.

There are still some problems with Bézier curves.

1. Instability

   As in the polynomial format high degree Bézier curves are still unstable in computational algorithms. A solution to this is to split the curve into segments and use a low degree Bézier curve for each segment. Techniques exist for matching derivatives at segment joins[4], these basically consist of putting constraints on the positioning of the control points. This leads to B-spline curves which are described in the next section.

2. No local control

   If one control point of a Bézier curve is moved the shape of the whole curve changes; this is often undesirable.

2.2.3 B-Spline curve

B-Spline curves are a generalization of Bézier curves.

Let $T = \{t_0, t_1, .., t_m\}$ be a non decreasing sequence of real numbers. The p'th degree B-spline curve is defined as follows :-

$$\mathbf{C}(t) = \sum_{i=0}^{n} \mathbf{P_i} N_{i,p}(t)$$

where the basis functions are defined [1]

$$\begin{aligned} N_{i,0}(t) &= \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \text{ and } t_i < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ N_{i,p}(t) &= \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t) \end{aligned}$$

T is called a knot vector, the t_i's being the knots. The most common form is the non-periodic B-Spline, this has the first $p+1$ knots equal to 0, and the last $p+1$ knots equal to 1. This assures the curve starts at the first control vector and ends at the last, and that the derivatives at the start and end of the curve are equal to $\mathbf{P_1} - \mathbf{P_0}$ and $\mathbf{P_n} - \mathbf{P_{n-1}}$ respectively. Only non-periodic B-Splines will be considered in this thesis. If the internal knots, i.e., $t_{p+1}, .., t_{m-p-1}$, are equally spaced the B-spline is called uniform. If there are no internal knots the B-Spline reduces to a Bézier curve.

The degree, the number of control vectors and the number of knots are related via the formulae $m = n + p + 1$.

[1] Note 0/0 sometimes arises in the evaluation of these functions, the result is defined as 0

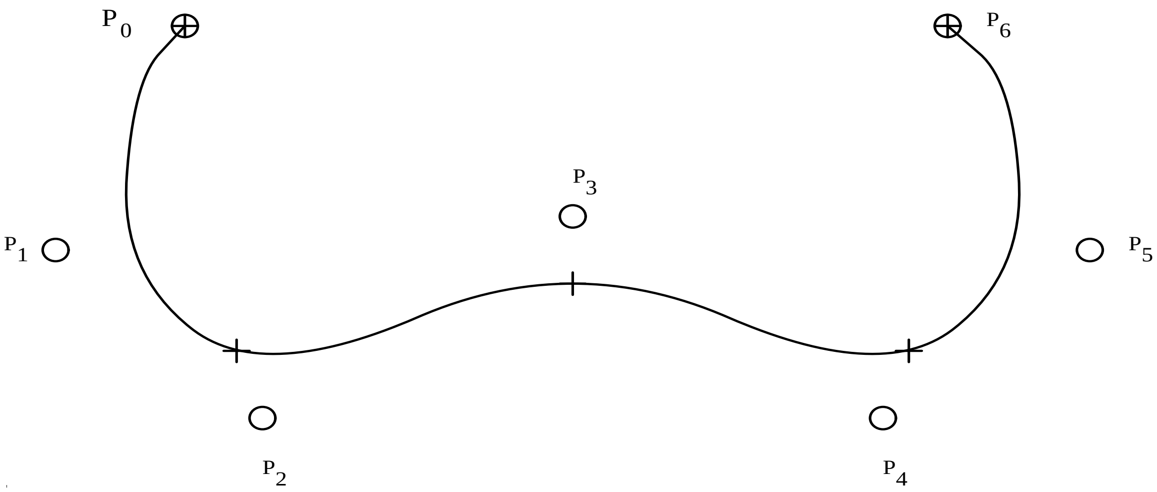

Figure 2.3: A cubic non-uniform B-Spline. The associated knot vector is $\{0,0,0,0,1/4,1/2,3/4,1,1,1,1\}$.

B-Splines have all the Bézier properties but solve the problems discussed in section 2.2.2.

1. Instability

   Since the degree of a B-Spline is independent of the number of control points, a low order can always be maintained. The only restraint on degree is the continuity of the curve at the knot values. If there are k duplicate knot values the curve is p-k differentiable at this point. Note deliberate discontinuities can be added at a point simply by repeating knot values.

2. Local control

   Clearly the shape of the basis functions, instead of being set as in Bézier curves, are controlled by the knot values. The knot values control

   (a) The range of influence each control vector has over the curve ($p+1$ knot spans), e.g., figure 2.3 shows an example of a non-uniform B-Spline curve with control points, and knot points added. Altering the first control point P_0 would only alter the curve section 0 to 1/4, since this is the range covered by the first p+2 knots. Altering the second would only alter the curve section 0 to 1/2 etc.

(b) The amount of influence each control point has over the curve in its region.

2.2.4 Rational curves

Consider a 4D perspective map

$$\mathbf{H}(wx, wy, wz, w) = \begin{cases} (\frac{wx}{w}, \frac{wy}{w}, \frac{wz}{w}) & \text{if } w \neq 0 \\ direction(wx, wy) & \text{if } w = 0 \end{cases}$$

This has the effect of projecting a 4D point, say $\mathbf{P^w}$, onto the plane w=1, along the line $\mathbf{P^w}$ makes with the origin . Note the 3D points produced by the map are independent of the w coordinate. The mapping $\mathbf{H}$ can be generalized to representing any dimensional point by an n+1 dimensional one. The n+1 dimensional point is called a homogeneous coordinate.

If we use homogeneous points to represent the control points in the Bézier and non-uniform B-Spline definitions above we form two new curve types.

1. Rational Bézier curves

$$\begin{aligned} \mathbf{C(t)} &= \mathbf{H}\{\mathbf{C^w}(t)\} = \mathbf{H}\{\sum_{i=0}^{p} B_{i,p}(t)\mathbf{P_i^w}\} \\ &= \frac{\sum_{i=0}^{p} B_{i,p}(t) w_i \mathbf{P_i}}{\sum_{i=0}^{p} B_{i,p}(t) w_i} \end{aligned}$$

2. Non Uniform Rational B-Spline (NURBS) curves

$$\begin{aligned} \mathbf{C}(t) &= \mathbf{H}\{\mathbf{C^w}(t)\} = \mathbf{H}\{\sum_{i=0}^{n} N_{i,p}(t)\mathbf{P_i^w}\} \\ &= \frac{\sum_{i=0}^{n} N_{i,p}(t) w_i \mathbf{P_i}}{\sum_{i=0}^{n} N_{i,p}(t) w_i} \end{aligned}$$

where

- p is the degree of the curve.
- $B_{i,p}$ are the Bézier basis functions.
- $N_{i,p}$ are the B-Spline basis functions.
- $\mathbf{P_i^w}$ represents a homogeneous control point.
- $\mathbf{P_i}$ represents a curve space control point.
- w_i is the last ordinate of the homogeneous point $\mathbf{P_i^w}$. It is called the weight of control point $\mathbf{P_i}$.

These curves have some interesting properties.

- All properties are kept from their non-rational counterparts.
- Closed under perspective transformation

  To perform a perspective transformation on the curve, the transformation only needs to be applied to its homogeneous control points.
- Extra local control

  Altering a weight associated with a control point has a direct effect on the curve near the control point. (See fig 2.4).

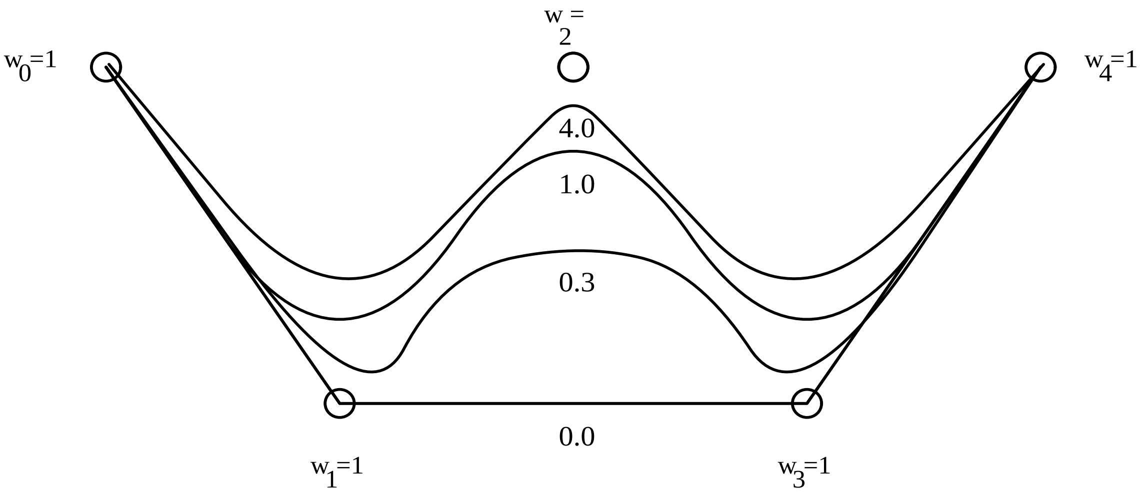

Figure 2.4: Quadratic rational B-Spline curves obtained by applying different values of w_2

- Conic sections

  Conic sections can now be represented exactly, which was impossible with the non-rational forms discussed above.

- Much of the theory and computational algorithms of the non-rational curves also apply. The exception being dimensional specific material. e.g., Bounding box calculations.

2.3 Surfaces

Surfaces can be described by using a tensor product of two curve descriptions.

e.g., A NURBS surface of degree (p,q) can be defined

$$\begin{aligned}\mathbf{S}(u,v) &= \mathbf{H}\{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)\mathbf{P^w_{i,j}}\} \\ &= \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P_{i,j}}}{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)w_{i,j}}\end{aligned}$$

where

- $N_{i,p}$ and $N_{j,q}$ are B spline basis functions each with their own knot vector.
- $\mathbf{P^w_{i,j}}$ represents a homogeneous control point.
- $\mathbf{P_{i,j}}$ is a surface space control point, they form the control net of the surface.
- $w_{i,j}$ is the weight of the control point $\mathbf{P_{i,j}}$.

The control points of a surface when linearly joined form a rectangular control net which gives a rough approximation to the surface.

The geometric properties of this surface definition are analogous to the curve counterpart, except there is no generalization known for the variation diminishing property.

For a more thorough introduction to curve and surface representations see reference [11] or [7], For a more detailed description of B-Splines see reference [4], [10] or [22].

Chapter 3

Ray Tracing

3.1 Introduction

Ray tracing is a technique that tries to produce photo realistic pictures of objects held as data within a computer. Effects such as shadows, reflection, refraction and transparency can all be generated using this method.

Consider the simple box camera shown in figure 3.1. A picture of the scene can be generated by placing film inside the box, and letting light enter through the small hole on the opposite side. The colour at any point on the film depends on the frequency and intensity of the light ray hitting it at that point. The light ray was originally generated by a light source, and maybe after being reflected or refracted by an object or objects ended up going through the pin hole and onto the film.

In ray tracing the film is replaced by a computer screen and moved in front of the

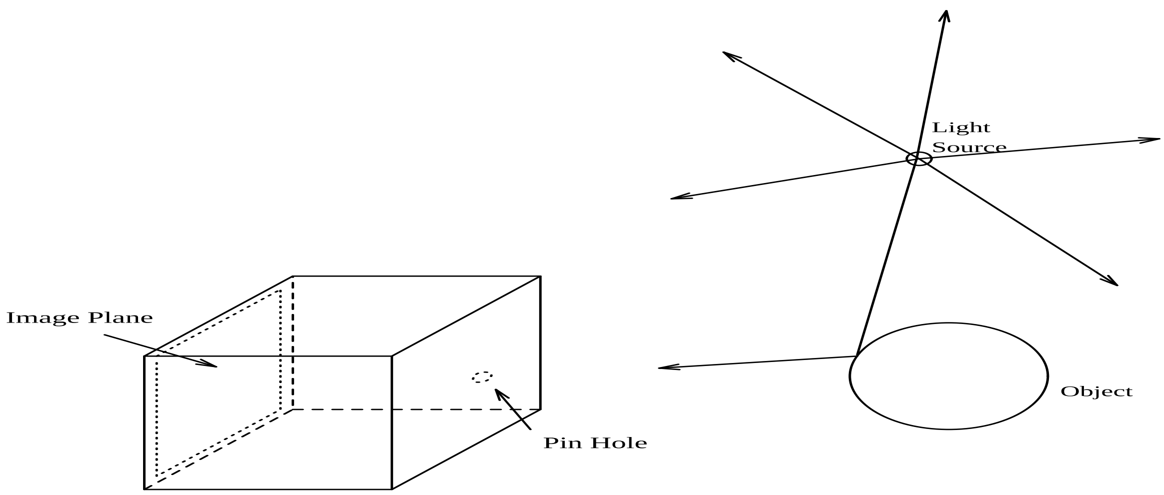

Figure 3.1: Simple box camera

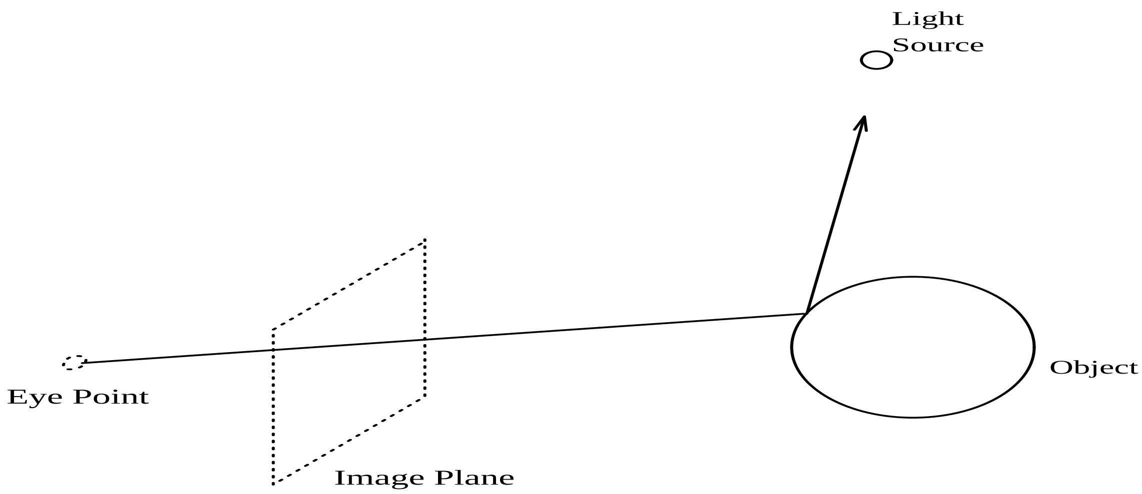

Figure 3.2: Ray tracing model

pinhole (now called the eye point). Light rays are followed from the eye point, through the screen and into the scene, in which ultimately they will be traced back to a light source (see figure 3.2). Rays are not followed from a light source as an infinite amount are produced at each, keeping track of all of them to see which would contribute to the computer scene is impractical

The problem is how to represent light and its interactions with the objects of the scene.

3.2 Light representation

A light ray can be perceived as being made up of small particles called photons, each having its own frequency and energy value, the overall colour of a ray seen by the eye is the average of the frequency and energy values carried by these photons.

Holding all the frequency and energy components for each ray would be impractical, so only energy values for three frequencies are normally kept, that of red, green and blue. This may seem quite restrictive but in fact a large subset of colours perceivable by the human eye can be made up using different weightings of these frequencies. Computer graphics monitors can also directly take this information and produce an appropriate colour on the screen.

A light ray in ray tracing can thus be represented by three different rays, having the frequencies of red, green and blue.

3.3 Light interaction models

3.3.1 Theoretical models

What actually happens when a light ray hits the surface of an object? The overall result is made up of four effects or transport modes : specular reflection, diffuse reflection, specular transmission and diffuse transmission. Idealized views of these effects are used as a basis for most ray tracing illumination models. These idealized views are :

1. Perfect specular reflection

   This occurs on hard shiny surfaces, e.g., mirrors. The ray is totally reflected by the top of the surface, much like a ball bouncing off a hard floor. Given the angle of the incoming light ray, and the normal at the point where the ray hits the surface, the direction of the reflected ray can be calculated by the following formula [16].

   $$\mathbf{R} = \mathbf{I} - 2 * (\mathbf{N} \cdot \mathbf{I})\mathbf{N}$$

   where

   - **I** is the incident ray
   - **N** is the surface normal
   - **R** is the reflected ray.

2. Perfect diffuse reflection

   Occurs with rough surfaces. When a ray hits the surface it is absorbed by it. This gives the object more energy which it will turn into heat or re-radiate as light. These re-radiated light rays have no particular direction thus they are modelled as going in all directions with equal amplitude. The amplitude is proportional to the angle between the incoming light ray, and the normal on the surface at the point of intersection. This idealized model is known as the Lambertain reflection model.

3. Perfect specular transmission

   Specular transmission occurs on hard transparent objects.

   Light that travels though a surface of a transparent object is called transmitted light. The direction of a transmitted ray given an incident ray can be worked out by an equation known as Snells law. This relates the angle the incident ray makes with the normal of the surface, with the angle the transmitted ray makes with the normal.

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{\eta_2}{\eta_1}$$

where

- θ_1 is the angle between the incident light ray and the surface normal
- θ_2 is the angle between the transmitted light ray and the surface normal
- η_1 is the index of refraction of the medium which the incident light is travelling in
- η_2 is the index of refraction of the medium which the transmission light is travelling in.

The index of refraction of a medium describes the speed of a ray of light in that medium compared to that of its speed in a vacuum. It depends on the frequency of the light. This is why a prism produces a spectrum of light.

The complete formula for the direction of a transmitted ray is shown below, it is derived in [16].

$$\mathbf{T} = \eta_{12}\mathbf{I} + (\eta_{12}cos(\theta_1) - \sqrt{1 + \eta_{12}^2(cos(\theta_1)^2 - 1)})\mathbf{N}$$

where

- η_{12} is the ratio η_1/η_2
- $\mathbf{I}$, $\mathbf{T}$, and $\mathbf{N}$ are the incident,transmitted and surface normal vectors respectively.

If the square root in the formula has an imaginary solution then a phenomena called total internal reflection has occurred. This happens when a light ray attempts to go from a dense medium to a less dense one at a shallow angle. The effect on the ray is that it only bounces off the intersection rather than going through it.

4. Perfect diffuse transmission

   When a ray hits a rough surface of a transparent object diffuse transmission is produced, this is very similar to diffuse reflection and the same cosine rule applies. Translucent plastic is a good example of an almost perfect diffuse transmission material.

3.3.2 Practical ray tracing models

An illumination model, in ray tracing, finds the frequency and intensity of an incident ray on a surface. This is sent from the eye point and through the image plane as discussed earlier. Light from two sources determine the frequency and intensity of the ray.

1. Specific light

   Light that has come directly from light sources.

2. Global light

   Light that has originally come from light sources, but has been reflected or refracted off other objects.

Both these can be separated into four parts: reflected diffuse, reflected specular, transmitted diffuse and transmitted specular light.

This information is often approximated in ray tracing illumination models to make the calculations faster. Some of the commonly used models are described below.

Phong's model

Phong's model [21] was the first model used in ray tracing. It was developed initially for use in local shading algorithms, i.e., algorithms that calculate the colour at a point on a surface directly. No global light information is available to these algorithms thus they use a constant term. Transmitted light is ignored in the model.

$$\mathbf{I} = \underbrace{k_d \sum_{j=1}^{l} (\mathbf{N}.\mathbf{L})(Object\ colour)}_{diffuse\ from\ light\ source} + \underbrace{k_s \sum_{j=1}^{l} (\mathbf{R}.\mathbf{V})^n \mathbf{I_j}}_{specular\ from\ light\ source} + \underbrace{\mathbf{I_a}}_{global\ term}$$

The variables are defined in table 3.1. The following points should be noted about the model

- The diffuse reflection coefficient and the specular coefficient should always add to 1, thus guaranteeing no energy generation. i.e., $k_d + k_s = 1$.
- The lambertain model is used for the diffuse term.
  Note $\mathbf{N} \cdot \mathbf{L} = cos(\theta_{\mathbf{NL}})$
- The specular term takes into account that not all the rays will be reflected in exactly the same direction as in the perfect case. The term given takes

into account that the microfacets on the surface will not all be in the same direction as the normal, although a majority will be. The exponent n controls the amount of microfacets pointing in the normal direction. Hence it controls the degree to which the surface matches the perfect specular model discussed above.

Whitted's model

Whitted [28] improves on phong's model by using a slightly more accurate light source specular term, and by including terms for global reflection and refraction. Note light source specific transmission is still ignored. His formula is :

$$
\begin{aligned}
I &= \underbrace{k_d \sum_{j=1}^{l} (\mathbf{N}.\mathbf{L})(object\, colour)}_{diffuse\ from\ light\ source} + \underbrace{k_s \sum_{j=1}^{l} (\mathbf{N}.\mathbf{H})^n \mathbf{I_j}}_{specular\ from\ light\ source} + \underbrace{\mathbf{I_a}}_{global\ diffuse} \\
&+ \underbrace{k_s \mathbf{I_r}}_{global\ specular} + \underbrace{k_t \mathbf{I_t}}_{global\ transmitted}
\end{aligned}
$$

The new light source specular term compares the perfect normal for greatest reflection to the incident ray, **H** ,against the actual normal. **H** can be derived to the following [16]

$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|}$$

The global reflection and transmission terms only look in the direction that will make the most contribution to the present viewer direction, i.e., uses the theoretical reflexive and transmission directions discussed in the perfect models. Looking in all other directions would have to be done by sending more rays, and hence would slow down a ray tracing process considerably. However Whitted does try and make the model more accurate by adding a perturbation to the normal, and following rays whos directions have been calculated from several settings of the normal. The higher the variance of the perturbation the rougher the surface would appear.

Only a constant global diffuse term is included because computing a more accurate term would involve rays being sent out in all directions from the intersection point. This is not practical.

The values of the reflected ray $\mathbf{I_r}$, and the transmitted, $\mathbf{I_t}$, are calculated by following the ray as with the incident ray. If they intersect objects then the illumination model is used again at this point to work out the value, if not they just attain the background colour. This process is normally stopped after a certain depth, or after the contribution of reflected and refracted rays is negligible to the present calculation.

Hall's model

The following improvements were added to Whitted's model by Hall [18].

- The angle of reflected or refracted light on a surface is actually dependent on the frequency and angle of incidence. Hall's model includes a fresnel term to cope with this.
- A term is added to represent transmitted specularity from direct light.
- Corrections are made for loss of intensity due to the material through which the rays travel.

The resulting model is expressed below

$$
\begin{aligned}
\mathbf{I} \;=\;& \underbrace{k_d \sum_{j=1}^{l} (\mathbf{N}.\mathbf{L}) R_d \mathbf{I_j}}_{diffuse\ from\ light\ source} + \underbrace{k_s \sum_{j=1}^{l} (\mathbf{N}.\mathbf{H})^n R_f \mathbf{I_j}}_{specular\ from\ light\ source} + \underbrace{k_t \sum_{j=1}^{l} (\mathbf{N}.\mathbf{H}')^n T_f \mathbf{I_j}}_{transmitted\ from\ light\ source} \\
&+ \underbrace{\mathbf{I_a} R_d}_{global\ diffuse} + \underbrace{k_s R_f \mathbf{I_r} F_r(d_r)}_{global\ specular} + \underbrace{k_t T_f \mathbf{I_t} F_t(d_t)}_{global\ transmitted}
\end{aligned}
$$

$\mathbf{H}'$ is equivalent to $\mathbf{H}$, it is set to the direction that would give maximal refraction. It can be derived [27] to be

$$\mathbf{H}' = \frac{-\mathbf{L} - \eta \mathbf{V}}{\eta - 1}$$

Note none of these models include a term for transmitted diffuse rays from direct light this is very similar to reflected diffusion and can easily be added [16].

The models are given in diagrammatic form in figure 3.3. The diagrams on the left of this figure show an example of a specific light distribution produced by each model. These diagrams assume that the incident ray hit the surface at the normal vector's origin, and that the light source is at approximately forty five degrees to the normal. You can see in all these models that the greatest contribution is made to an incident ray coming in at the perfect specular reflection angle to the light source direction, i.e., generating specular reflection.

The diagrams on the right show the global illumination used for each model. Note a constant is always used as an approximation for the global diffuse term. Global specular light is approximated in Whitted's and Hall's models by spawning new rays in the perfect specular reflection and refraction directions.

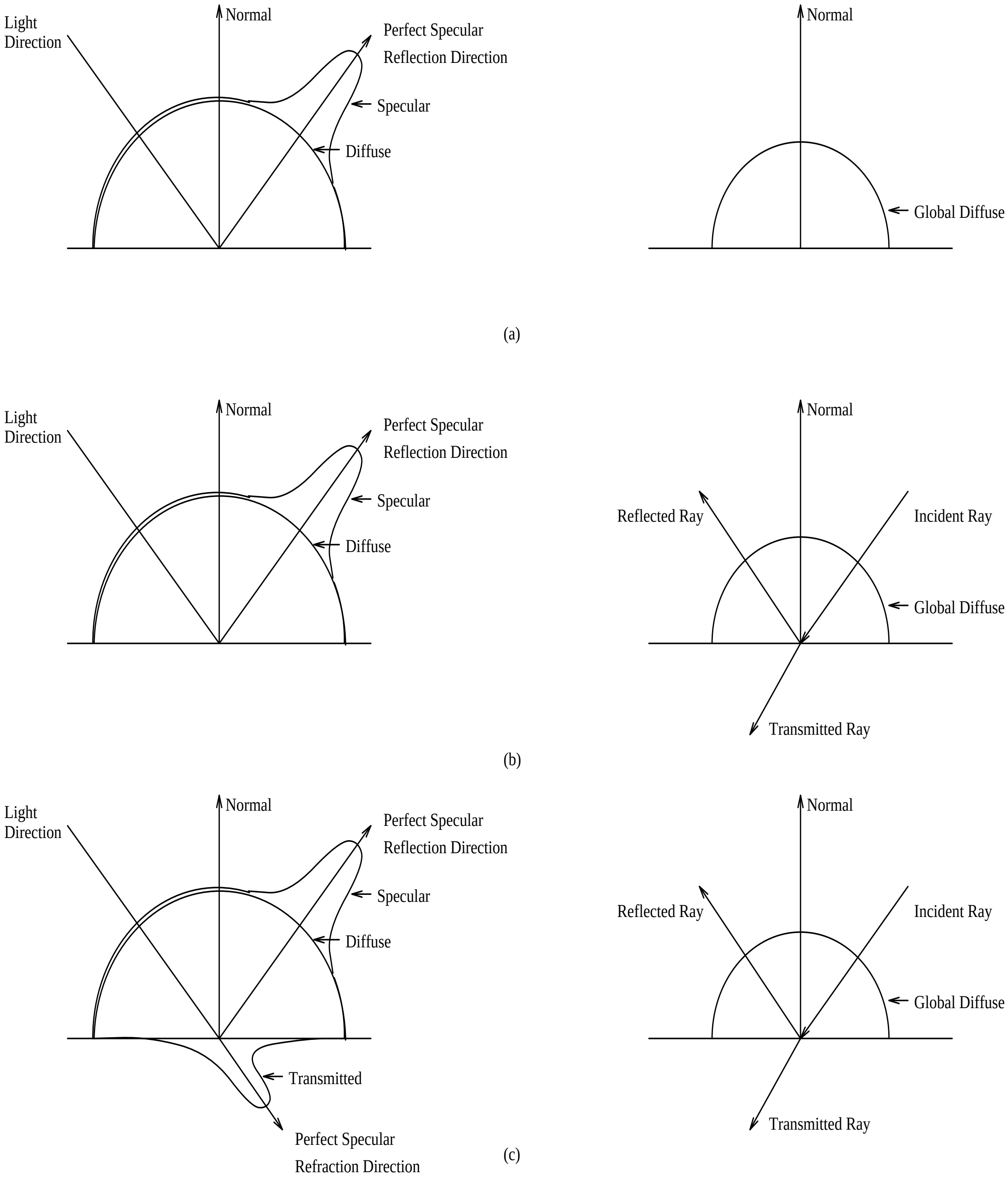

Figure 3.3: The phong (a), Whitted(b), Hall(c) illumination models, with specific light source models on the left and global illumination models on the right.

d_r	= distance reflected ray travels
d_t	= distance refracted ray travelled
F_r	= transmittance per unit length of material of reflected ray
F_t	= transmittance per unit length of material of refracted ray
$\mathbf{H}$	= unit mirror direction for reflected rays
$\mathbf{H'}$	= unit mirror direction for refracted rays
$\mathbf{I}$	= intensity of incident ray
$\mathbf{I_a}$	= intensity of global ambient illumination
$\mathbf{I_j}$	= intensity of jth light source
$\mathbf{I_r}$	= intensity of reflected ray
$\mathbf{I_t}$	= intensity of transmitted ray
j	= light source index
k_d	= material diffuse reflection coefficient
k_s	= material specular reflection coefficient
k_t	= material transmission coefficient
l	= number of light sources
$\mathbf{L}$	= unit light source vector
n	= exponent representing glossiness
$\mathbf{N}$	= unit surface vector normal
$\mathbf{R}$	= unit reflection vector
R_f	= Fresnel reflectance curve of material on wavelength basis
R_d	= diffuse reflectance curve of material on wavelength basis
T_f	= Fresnel transmissivity curve of material on wavelength basis
$\mathbf{V}$	= unit incident ray direction

Table 3.1: Definitions of terms in illumination models

3.3.3 Implementation model

The model I chose to implement was Whitted's model with the addition of a term for calculating the contribution of transmitted diffuse rays from direct light.

3.4 Anti-aliasing

The most obvious way of constructing a picture is to send one ray per pixel but this leads to several problems.

1. The objects on screen appear with jagged edges.

2. Small objects can be totally missed if they lie between sample points.

These are both instances of aliasing. Aliasing effects occur when some kind of continuous signal is represented by a set number of sample points from the signal. Several techniques are used to reduce these effects in ray tracing.

3.4.1 Super-sampling

The easiest and most obvious way is just to take more sample points. This could be done by sending more than one ray per pixel, and then averaging the result. The question is where should the rays be sent through the pixel to achieve a fair representation of what is contained within its boundaries? There are several possibilities :

- Uniform distribution [16]

  The rays are evenly spaced throughout the pixel.

- Stochastic distribution [16]

  The rays are spaced in a random distribution throughout the pixel. This simulates some noise on the image, the human eye finds this less irritating than regular sample points.

- Adaptive [16] [28]

  A set amount of rays, distributed throughout the pixel, are initially sent. If the value of these differ enough more rays are sent. Note this will represent sharp colour gradients, i.e., edges, very well.

3.4.2 Post-filtering

The final image produced by a ray tracer can be passed through an image filter to try and improve the picture quality, e.g., a gaussian filter can be used to reduce localized sharp gradients, smoothing jagged edges. Note if a low sample rate is used to generate the picture post-filtering can cause features to be lost or badly represented.

3.5 Speed up techniques

Ray/object intersection calculations can take up to 95% of the total time to ray trace complex scenes. This is partly due to the fact that each ray has to be checked with every object in the scene for intersection. The number of intersection calculations for a particular ray can be drastically reduced by splitting the scene into parts, and only performing intersection calculations for the objects that lie in the parts the ray will travel through. Two methods are commonly used :

- Voxel method [15] [16] [12]

  The voxel method splits the whole scene into boxes called voxels. Each box contains a list of objects that are contained within it. The number of objects within a voxel can be kept below a threshold by simply dividing an offending voxel into smaller ones. To find the minimum intersection point a ray is followed from its start point, i.e., the eye point, through the voxel network. As the ray goes through a voxel it is tested for intersection with the objects contained within it, the minimum distance intersection found is the minimum intersection for the whole scene. If no intersection is found the procedure is repeated for the next voxel in the rays path.

- Bounding box method [16]

  In this technique each object or group of objects is bounded by a box, called a bounding box. Initially only bounding boxes are checked with a ray for intersection. If the ray intersects any bounding boxes, then the objects within these are then checked for intersection. Note

  - For complex scenes there can be several layers of bounding boxes. i.e., bounding boxes containing other bounding boxes.
  - Boxes are used because there exist fast line/box intersection routines. Other more complex bounding volumes are sometimes used giving closer fits to the objects they bound. This can only be justified however if the time taken to work out the intersection of the ray with the bounding volume is significantly faster than that of the time taken to calculate the intersection of the ray with the object or objects contained within it.

Chapter 4

Ray Tracing NURBS

4.1 Introduction

Each of the lighting models discussed in the last chapter need to be provided with a line/surface intersection point and the normal vector at this intersection point. The primitive objects used in most ray tracers are spheres and polygons, finding the intersection and normal for these surfaces is simple and will be discussed in the implementation section of this thesis. Finding these values for a NURBS surface is a non trivial problem, this chapter will describe several techniques.

4.2 Tessellation

One way of rendering a complicated surface is to approximate it via lots of smaller surfaces which have simpler intersection procedures. The approximated surface is called a tessellation. Triangles, because of their simplicity, are often used for the tessellation procedure. Methods for shading a triangular tessellation and processes to actually create a tessellation from a NURBS surface are now discussed.

4.2.1 Shading methods

Two main shading methods are used in local shading models to make planar polygons appear as curved surfaces. These can equally as well be used in global illumination models. They both make use of normals defined at polygon corners. These can be the exact normal of the surface at this point, or an averaging of the joining polygon normals.

- Gouraud Shading [17]

  This takes a weighted average of the colours at the corner points, the weights depend on where on the polygon is being coloured. The colours at polygon corners can be worked out using an illumination model e.g., phong's. To make this method more efficient the colour at a corner point once calculated should be saved. This value can then be reused if a ray hits a polygon adjacent to the point. Gouraud shading suffers from the following :

  - Mach band effect

    Colours over polygon borders tend not to be smooth, the human visualization system tends to pick this up as bright bands.

  - Bad specular highlight representation

    Specular highlights tend not to be represented that accurately. The shape of the highlight also changes depending on the orientation of the underlying polygon mesh.

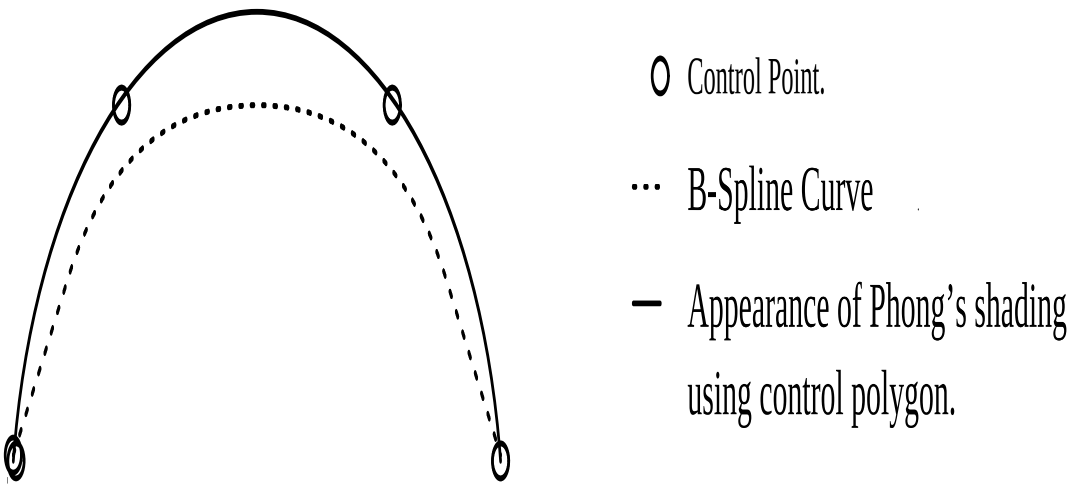

Figure 4.1: Appearance of Phong's shading using control points as interpolation points

These effects are reduced, but not cured, if the polygon sizes are reduced.

- Phong shading [21]

  This takes a weighted average of the normals at the corner points of the polygon to produce a new normal. This can be used in an illumination model ,such as phong's, to calculate a colour. Phong's shading is slower than Gouraud's although it reduces its problems drastically, and produces a far more accurate picture for the same size polygons.

The rest of the section will discuss ways of tessellating a NURBS surface into triangles.

4.2.2 NURBS tessellation methods

Using control polygon

Probably the most obvious and easiest technique is to use the control mesh of a NURBS patch as an approximation to the surface. Unfortunately this does not always give an accurate representation (See figure 4.1).

The accuracy can be improved upon by refining the surface, i.e., adding more control points. This can be done via algorithms such as the Oslo and Boehm algorithms [6] [8]. A more accurate result for the same amount of sample points can be achieved however by taking the sample points on the surface

These can be joined to make a tessellation of the original surface. The difficulty of this is at what place to take the sample points from.

Static tessellation

An obvious way of taking points on a surface is to evaluate it at uniform intervals of the surface parameters. A version of this was implemented by the author as a first method for rendering NURBS surfaces. The implemented version evaluates the surface at a given number of points between each knot span. One number is used as a divisor for all the knot spans in the u direction, and another is used similarly for the v direction. These numbers are set by the user. Each knot span was split because this was computationally simpler to do than splitting the whole knot vector span equally. There are several problems with this method

- If either of the split numbers are too large a feature might be missed, or badly represented (see figure 4.2).
- If either set to small a needless amount of splitting has been done to represent the surface.

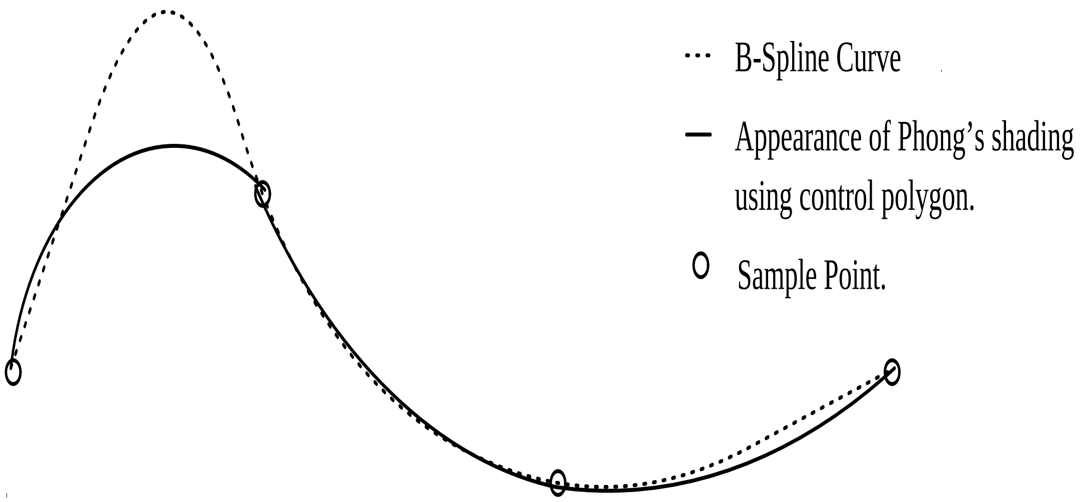

Figure 4.2: An example of the split number being set to large

- Even if the split numbers are correctly set, if the surface is badly parameterized certain parts of the surface may still be over sampled. This is because uniform steps in the parameter space would produce widely varying steps along the surface.

Dynamic tessellation

This section will look at several ways of automatically setting the step size between sample points. Before this we have to consider criteria for calculating the size of interval needed between samples.

1. Polygon mesh deviation

   This places a threshold on the maximum distance a planar polygon in the tessellation can be away from the exact surface. If set too large the polygon approximation might miss features of the surface. The deviation clearly relies on the viewing transformations being used to display the surface.

2. Shading model specific

   This is a criteria based on the shading model being used

   (a) Gouraud's

       A threshold should be set on the size of a polygon in the tessellation. This should depend on light source placements. i.e., need a lot of small polygons near light sources so that specular reflection can be modelled accurately.

   (b) Phong's

       A threshold should be set on the change of curvature (the first derivative) allowed in a segment of surface to be modelled by a polygon in the tessellation. This is because the closer the curvature is to constant in the original surface patch the more accurate phong's shading will be over the polygon representing it.

Uniform step intervals

Abi-Ezzi and Shirman [1] describe a method where one step size is calculated automatically and used throughout a section of a surface. Their method is given set values for the criteria 1 and 2a above. It then calculates derivative bounds for the whole surface segment involved, i.e., between knot spans in our case. Using these, and taking into account the viewing transformations and lighting information they calculate the number of splits over the segment in each direction that are needed for an accurate picture. Note for a different lighting arrangement or viewing transformation the derivative bounds do not have to be recalculated. Their method does not work too well if the surface is badly parameterized. They suggest that rational surfaces are badly parameterized when their weights widely vary, and thus repeatedly subdivide the surface if this occurs.

Non uniform step intervals

In this method each individual step size is automatically calculated. Lien, Shantz and Vaughan [20] describe a rendering procedure that performs automatic step size calculations in this way using adaptive forward differencing. They describe a process where the step size along the u and v directions are progressively changed depending on if the previous step size was too large or small for some criteria. They do this by continually halving or doubling his step transformation. The method they describe uses the criteria of stepping one pixel at a time in viewing space, this could easily be changed however to the criteria discussed above.

This method has two main advantages over methods that use uniform intervals

1. Less approximating triangles are needed for the same accuracy.

2. Badly parameterized surfaces cause no problem.

Unfortunately calculating the tessellation by this method would probably take a lot longer because the interval criteria used would have to be checked for after each differencing step.

Using dynamic tessellation for ray tracing

Both the techniques described above would need information about viewing transformations, e.g., the magnification used on a surface. If Gourard's shading is being used it also requires the lighting information for the scene. In ray tracing this information is hard to calculate. This comes from the fact that objects can be reflected and refracted from surfaces which may lead to effects such as magnification of surfaces, and production of more light sources.

The only alternative apart from a complete pre ray trace to find out where objects are reflected and refracted is to make the user enter the criteria values for each object in the scene. These values would be a lot more intuitive than the original split numbers in our static tessellation approach, but would probably still take a few attempts to get right for a picture.

For these reasons I decided not to continue along this path, my objective being not to have the user use a trial and error method if at all possible to get a good image out. The next techniques try to solve the problem by finding the exact intersection of the ray with the surface.

4.3 Exact intersection methods

4.3.1 Intersection techniques

The problem is to find the first intersection point between a ray and a NURBS surface patch. This problem cannot be solved directly so an iteration technique must to be used. Only two iteration techniques seem to have been used to solve this and similar problems.

- Solving a set of simultaneous equations via the Newton Raphson method (Sweeney & Bartels [25] and Yang [29]).
- Subdivision (Schneider [24] and Veen [26]).

The first to be looked at will be the Newton Raphson method. Sweeney/Bartell and Yang both use the method on uniform B-Spines but the basic process could be extended to NURBS.

4.3.2 Newton Raphson

Newton Raphson method

The Newton Raphson method in one variable can solve an equation of the form.

$$f(x) = 0$$

It involves taking an initial guess at the root, x_0, and putting it in to the following equation to get a closer approximation to the root.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

This process is repeated until the root is found within a certain tolerance. The method has quadratic convergence, i.e., it converges quickly to the region of the solution and gets slower as it gets nearer and nearer the solution. There are several problems however :

- It may not converge.
- It may converge to an undesirable solution.

It can be shown [14] that if $|f(x)f''(x)/f'(x)^2| < 1$ over an interval, then for any root and starting point in the interval the method will converge. Working this out in practice can be very difficult, so methods generally ignore this and just look for a close starting point to the root, and hope the iteration will converge.

The Newton Raphson method can be generalized to two variable problems. It takes equations of the form

$$\begin{aligned} f(u,v) &= 0 \\ g(u,v) &= 0 \end{aligned}$$

and has the iteration step

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - J^{-1} \begin{pmatrix} f(u_n, v_n) \\ g(u_n, v_n) \end{pmatrix}$$

where

- J is the Jacobian matrix defined

$$\begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{pmatrix} \begin{matrix} u = u_n \\ v = v_n \end{matrix}$$

  thus

$$J^{-1} = \frac{1}{det(J)} \begin{pmatrix} \frac{\partial g}{\partial v} & -\frac{\partial f}{\partial v} \\ -\frac{\partial g}{\partial u} & \frac{\partial f}{\partial u} \end{pmatrix} \begin{matrix} u = u_n \\ v = v_n \end{matrix}$$

Using Newton Raphson for the intersection problem

We now derive f and g functions for our intersection problem

The light ray can be defined as

$$(O_x, O_y, O_z) + \lambda(D_x, D_y, D_z)$$

where

- **O** and **D** are the origin and direction of the line respectively.

A plane has the definition

$$(N_x, N_y, N_z) \cdot (x, y, z) = D$$

where

- **N** is the unit normal of the plane
- (x, y, z) is a point on the plane
- **D** is the perpendicular distance from the plane to the origin.

Two planes perpendicular to each other can represent a line. Choose two planes perpendicular to the x and y axis.

If a plane is perpendicular to the x axis and contains the line above we have

$$(0, N_y, N_z).(O_x, O_y, O_z) = D \quad (4.1)$$

$$(0, N_y, N_z).(D_x, D_y, D_z) = 0 \quad (4.2)$$

Equation 4.1 assures that the origin of the line lies on the plane, while 4.2 assures that the direction of the line and normal of the plane are perpendicular.

Solving we get the plane

$$(0, D_z, -D_y) \cdot (x, y, z) = D_z O_y - D_y O_z \quad (4.3)$$

solving similarly for a plane parallel to the y axis we get

$$(D_z, 0, -D_x) \cdot (x, y, z) = D_z O_x - D_x O_z \tag{4.4}$$

If the NURBS surface equation is substituted into equations 4.3 and 4.4 we get two loci. They are curves that are on the NURBS surface but also lie within the planes. Thus solving the intersection of these loci finds the intersection between the line and the surface. The equations for the loci are :

$$f(u,v) = (0, D_z, -D_y) \cdot \mathbf{S}(u,v) - D_1 = 0 \tag{4.5}$$

$$g(u,v) = (D_z, 0, -D_x) \cdot \mathbf{S}(u,v) - D_2 = 0 \tag{4.6}$$

Clearly we have the correct form in equations 4.5 and 4.6 to use the two variable Newton Raphson method. What we need is the partial derivatives of f and g with respect to u and v, and start values u_0 and v_0.

Taking the derivative of f we have

$$\begin{aligned}\frac{\partial f}{\partial u} &= (0, D_z, -D_y)\frac{\partial \mathbf{S}(u,v)}{\partial u} \\ \frac{\partial f}{\partial v} &= (0, D_z, -D_y)\frac{\partial \mathbf{S}(u,v)}{\partial v}\end{aligned}$$

similarly for g.

we need to find the partial derivatives $\frac{\partial \mathbf{S}(u,v)}{\partial u}$ and $\frac{\partial \mathbf{S}(u,v)}{\partial v}$

For a NURBS surface we have

$$\frac{\partial \mathbf{S}(u,v)}{\partial u} = \frac{\partial}{\partial u}\mathbf{H}\{\mathbf{S^w}(u,v)\}$$

$$
\begin{aligned}
&= \frac{\partial}{\partial u}\mathbf{H}\{(wx(u,v), wy(u,v), wz(u,v), w(u,v))\} \\
&= \frac{\partial}{\partial u}\mathbf{H}\{(\mathbf{P^w}(u,v), w(u,v))\} \\
&= \frac{\partial}{\partial u}\frac{\mathbf{P^w}(u,v)}{w(u,v)} \\
&= \frac{\mathbf{P^w}'(u,v)w(u,v) - \mathbf{P^w}(u,v)w'(u,v)}{w^2(u,v)}
\end{aligned}
$$

where

- $\mathbf{H}$ is the homogeneous mapping, discussed in chapter 2
- $\mathbf{S^w}(u,v)$ is a homogeneous NURBS surface
- $wx(u,v)$, $wy(u,v)$, $wz(u,v)$, $w(u,v)$ are the components of the homogeneous surface
- $\mathbf{P^w}$ is a mapping grouping the first three homogeneous surface components.

We need the component derivatives of the homogeneous curve.

Taking the x component

$$
\begin{aligned}
\frac{\partial}{\partial u}wx(u,v) &= \frac{\partial}{\partial u}(\sum_{i=0}^{n}\sum_{j=0}^{m} wx_{i,j}N_{i,p}(u)N_{j,q}(v)) \\
&= \sum_{i=0}^{n}\sum_{j=0}^{m} wx_{i,j}N_{j,q}(v)\frac{\partial}{\partial u}N_{i,p}(u)
\end{aligned}
$$

where

$$
\begin{aligned}
\frac{\partial}{\partial u}N_{i,0}(u) &= 0 \\
\frac{\partial}{\partial u}N_{i,p}(u) &= \frac{N_{i,p-1}(u)}{u_{i+p}-u_i} + \frac{(u-u_i)N'_{i,p-1}(u)}{u_{i+p}-u_i} \\
&- \frac{N_{i+1,p+1}(u)}{u_{i+p+1}-u_{i+1}} + \frac{(u_{i+p+1}-u)N'_{i+1,p-1}(u)}{u_{i+p+1}-u_{i+1}}
\end{aligned}
$$

Similarly for the other homogeneous components.

$\frac{\partial \mathbf{S}(u,v)}{\partial v}$ can be computed similarly.

This method will converge if a close enough starting point is given to the root. Once the solution is found the normal at the intersection point can be calculated easily by computing $\frac{\partial \mathbf{S}(u,v)}{\partial u} \times \frac{\partial \mathbf{S}(u,v)}{\partial v}$, which would be a side effect of the last iteration of the Newton Raphson method.

The two methods described below both create a bounding box tree structure at a preprocessing stage. The root bounding box contains the whole surface. Its descendant bounding boxes contain less and less of the surface. When a ray hits the root bounding box, it is tested against its descendants etc. If a ray hits a leaf bounding box the Newton Raphson algorithm is invoked with the initial parameter values that are stored within that leaf.

The way they differ is in the construction of the bounding boxes and the initial parameters used.

- Yang's Method [29]

  This method splits the surface in object space, each non leaf bounding box being divided into eight parts. An approximation of the average values of the surface inside a leaf bounding box are used as starting points for that box. The algorithm is :

  ```
  Perform incremental steps over the surface to form
  an array of points on the surface (the u and v
  increments are given by the user).
  ```

```
   Define a root bounding box just large enough to contain
   all of the surface points.
L1 Split box evenly into eight new boxes.
   For each of the eight boxes.
       Are any surface points in box?
       Yes:Reached pre-defined depth level?
           Yes:Average surface points in box and use
               this as a u0, v0 starting value for
               the leaf box.
           No :Goto line L1.
       No :Set box empty.
```

- Sweeney and Bartel Method [25]

This method keeps splitting the surface in parameter space until sections containing individual knot values are reached. These are then used as starting values for leaf bounding boxes based around the nearest control point. The algorithm is :

```
   Refine surface to a set depth.
   Assign total u and v ranges to root box.
L1 Split box in four.  Associate each with a quarter of
   the parents u,v ranges.
   i.e., ((u_min,(u_max - u_min)/2),(v_min,(v_max - v_min)/2)),
         ((u_max - u_min)/2,u_max),(v_min,(v_max - v_min)/2)), etc ..
   For each of the four boxes.
       Is there only one pair of knots in the ranges
```

```
        associated with the box (call $u_r, v_s$).
        Yes:Create bounding box around $U_{r-2,s-2}$ and use $u_r$, $v_s$
            as newton starting points.
        No :Goto line $L_1$.
    As recurse back up the tree make parent bounding box contain all
    its immediate descendant bounding boxes.
```

Note for the Sweeney and Bartel algorithm above

- The algorithm works for uniform cubic B-spline surface patches.
- The algorithm given by Sweeney/Bartel does not explain the case when one knot is in the range of u and more than one knot is in the range v, and visa versa. I suspect you just split the appropriate u or v range.
- The nearest knot pair to a control point $\mathbf{P_{r,s}}$, of a uniform cubic Bézier, are u_{r+2} and v_{s+2}, this can be seen from cubic basis functions. This seems a good approximation of the nearest point on the surface to a control point. Although the scheme breaks down slightly near edges of the surface. e.g., if the algorithm converges to knot values 0, v_s, which u knot do we take bearing in mind $u_0 = u_1 = u_2 = u_3 = 0$. A way around this is to take the control points $\mathbf{P_{0,s}}$ and $\mathbf{P_{1,s}}$ and put bounding boxes around both. Zero is a good starting point for $\mathbf{P_{0,s}}$, and $u_4/2$ would be a good starting point for $\mathbf{P_{1,s}}$ thus these values can be used.
- The bounding box around a vertex is defined in terms of immediate vertices around it, and a user set overlap parameter. This controls the

amount of overlap between adjacent leaf bounding boxes. The more curvy the surface is the higher the overlap that has to be set.

To generalize the Sweeney/Bartel algorithm an approximate starting point has to be found for any degree'd non uniform surface. For uniform surfaces the low ordered basis functions seem to suggest that:

1. All uniform odd degree surfaces can use the same starting points as the algorithm above.
2. All uniform even degree surfaces can use mid points between knots as their starting points.

A different method of finding starting points has to be used for non uniform surfaces.

The maximum subdivision level acts much like the number of set splits in static tessellation (although it would normally be set much lower). Set too low the Newton Raphson algorithm might not converge, or converge to the wrong answer leaving badly coloured pixels in the final image. Set to high and needless amount of storage space is used. Thus the maximum subdivision level has to be set at a high enough value to force enough Newton Raphson iterations to converge, to the correct value, to produce a satisfactory picture. This means the user still has to set parameters for each NURBS surface, as in the tessellation techniques.

4.3.3 Subdivision

The basic process of subdivision is

```
L1 Does the line intersect the surface.
   yes:Has sufficient accuracy been attained.
       yes:Return solution.
       no :Split surface into sections.
           For each section.
               Goto line L1.
   no :Ignore surface.
```

The main advantage of this process is that it will always converge, the disadvantage being that it is quite slow to do so. Veen and Schneider both have produced procedures for the subdivision of NURBS.

- Schneider [24]

  Schneider checks for line/surface intersection by using the control mesh of the surface. Since the control mesh forms a convex hull of the surface, any line that intersects the surface will intersect the control mesh. The method checks a line for intersection with the control mesh of the surface, if there is an intersection the control mesh is further refined (subdivided) into four sections, each attaining a closer approximity to the surface than the original mesh, the intersection test is repeated for each. When the control mesh reaches a set accuracy the intersection with it is taken as a solution.

  A method called the Oslo algorithm [8] (with a few alterations so that it can be implemented efficiently on VLSI circuits) is used to subdivide a surface.

One limitation with the search procedure needed with subdivision is that a large amount of surfaces have to be kept in memory at once, i.e., branches that have yet to be expanded. Schneider's searching scheme, called subdivision in stages, stores a path that can be used to regenerate the surface rather than storing the surface itself. This greatly reduces the speed, but it decreases the storage requirements.

- Veen [26]

  Veen notes that the Oslo algorithm for subdividing NURBS is quite a slow process since it has to take into account knot values. He converts the NURBS surface into rational Bézier surfaces. Although this increases the number of control points by a factor of sixteen, it greatly increases the speed of the subdivision process.

  He also checks for the ray/surface intersection by checking the intersection with the control mesh, but he uses a novel technique to do it. He first translates the line and the patch such that the line has its start point at the origin. Then rotates the line with the patch such that the line lies on the x axis. Now all that has to be checked for is if the surface patches control mesh intersects the x axis. This is obviously true if

  $$(\exists y_{i,j} = 0 \vee (\exists y_{i,j} < 0 \wedge \exists y_{i,j} > 0)) \wedge (\exists z_{i,j} = 0 \vee (\exists z_{i,j} < 0 \wedge \exists z_{i,j} > 0))$$

  where

  - $y_{i,j}$ and $z_{i,j}$ are the y and z ordinates of the 3-D control point $\mathbf{P_{i,j}}$ of the transformed rational Bézier surface.

  Veen gives no search procedure.

Subdivision techniques have a great advantage over the others already discussed in that they always converge. Also if the search procedure is correct to the smallest distanced solution. All that is needed is a convergence criteria. This would be independent of a NURBS shape, i.e., no parameters are needed from the user as in the other methods discussed. For this reason the author decided to implement a subdivision technique based on Veen's, which is described in chapter 5.

Chapter 5

Parallelism

5.1 Meiko computing surface

The implementation of parallel processing is clearly dependent on the machine used. The machine available for implementation is called the Meiko Computing Surface. A brief summary of the hardware of the machine and the software available is given below.

5.1.1 Hardware

The machine is made up of modular boards and an interconnection network to communicate between these boards.

Each board can be one of varying types, the ones presently available on the University of Manchester computing surface are listed below along with their function.

- MK060 : This carries four transputers each with up to 8Mbytes. Six boards are installed on the Manchester machine each transputer having 4Mbytes of memory.
- MK042 : This carries four transputers each with 3Mbytes. Four boards are installed on the Manchester machine.
- MK086 : This carries two intel i860 chips with four transputers to perform the communications between the i860's and the interconnection network. One of these boards is installed on the Manchester machine.
- MK052 : This is a display board. It has one transputer with 4Mbytes ram and a frame store of 2Mbytes. 256 Colours are available for each rgb gun at a resolution of 768 by 576 pixels. One is installed in the Manchester machine and is connected directly to a 24 bit rgb monitor.
- MK083 : This contains a sparc chip. This is used as a host processor for the system. One is installed in the Manchester machine.

All communications between different boards need to use the network, thus will be slower than communication between processors on the same board.

5.1.2 Software

CS(Communicating Sequential) Tools is a set of libraries that allow a programmer working in a standard language access to functionality of the underlying hardware. It can be divided into two parts.

- Communication

  It provides extensions to languages such as C and Fortran so they can communicate with other programs on different processors. It also can load a run time execution module onto a processor that allows a program on that processor to access I/O and operating system calls by automatically setting up communication channels to the host board. This will only be loaded if the program requests such system calls.

- System Configuration

  CSTools provides two methods of loading applications onto the processor network.

  1. PAR files

     These provides an automatic procedure for placing processes on processors. It uses a user script (called a Par file) that describes the network topology.

  2. CSBuild files

     These are user written programs that specifically specify what processes go on what processors, and the connection links between them.

5.2 Parallel ray tracing strategies

From the information given above it is clear that the Meiko's memory is distributed about its network. Three main strategies exist for implementing ray tracing on these types of machines.

- Processing without dataflow [3]

  All the object data is duplicated on each processor. This allows processes placed on different processors to run independently of each other. However the complexity of the scene is limited to that of which can fit on the processor with the least memory.

- Processing with ray dataflow [3]

  The object data is spatially distributed around the network. Each processor holds a region of the object space, ideally adjacent regions will be held on adjacent processors. A ray is initially sent to the processor that contains the region of object space that the ray first intersects with. The resultant rays to come out of this region will then be sent to the processors that contain the next region they will enter, this process then continues until there are no more resultant rays or a certain depth is reached.

- Processing with object dataflow [3]

  This method randomly distributes the object store around the processors. When a processor needs an object, bounding box or voxel it sends a message to the relevant processor, and it will be sent. Note for this to work information about how the data is distributed has to be stored somewhere, preferably on each processor.

The processing without dataflow was chosen to be implemented by the author, this was for the following reasons:-

- Very complex scenes can fit on the processors available. This is for two reasons.

  - The memory available on each transputer in the network is quite large, i.e., at least 2MBytes.
  - The NURBS representation used in the ray tracer program to represent surfaces require far less memory than triangular mesh representations used in normal ray tracer implementations.

- This is by far the fastest technique, because no communication is needed between the ray tracing processors.

5.3 Communication considerations

Given the method that was decided upon two things have to be considered

1. The communication protocol.

   How the processes are going to communicate with each other.

2. The communication network.

   How to physically connect the processors.

5.3.1 Communication protocol

A master slave type protocol seemed most appropriate for communication. In the implementation the master process sends each slave process a part of the image screen to ray trace. When a slave finishes it sends the colour data back to the master which then sends the next part of the image screen to be ray traced back, this process continues until the whole image screen has been ray traced. The

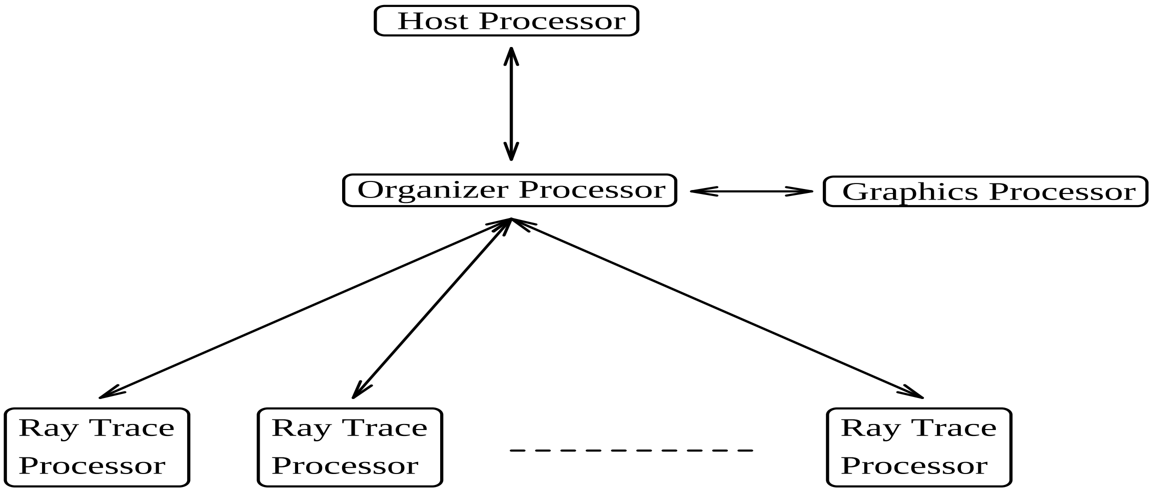

Figure 5.1: Virtual connection network used for implementation

virtual connection network used by the author is shown in figure 5.1.

The idealized case in parallisation is linear speed up, for example a picture produced on sixteen ray trace processors would be sixteen times faster than if one were used. There are two things that could stop linear speed up in the system implemented.

1. Communication overhead

   As more ray trace modules are added to the network, the physical time needed to send some data to a processor can increase. This is normally due to limited band width of the communication network, i.e., the physical connections in networks are limited, thus when these are used up connections have to be shared, thus increasing communication time.

2. Uneven loading on processors

   Different parts of the image screen will take varying amounts of time to ray trace, this is directly related to the complexity of the part sent. This can lead to processors doing different amounts of work, and thus there will not be a linear speed up.

The communication network is dependent on the machine and the hardware network used. It will be discussed in the next subsection.

Uneven loading can be reduced by splitting the image screen into small parts. The parts should be as simple as possible so that little communication is needed to send a part to a ray tracer module. A very simple method is to split the image screen into scan lines each scan line could easily be represented by its y coordinate. An alternative method [23] is to divide the image plane into equally sized small squares. This is done so that adjacent pixels are normally available to a processor, and thus can be used for anti-aliasing. The edge pixels would have to be dealt with specially. In the implementation the scan line method is used, anti-aliasing is produced by sending more rays per pixel, thus adjacent pixels are not needed.

5.3.2 Communication network

The idea when designing a physical network is to reduce communication time. Ransen [23] and Barton [5] both have produced parallel ray tracing schemes for a transputer network. Ransen uses a ring scheme to join all the slave processors up, while Barton uses long lines of transputers. Both these networks rely on using the transputer chip capability of allowing a through router program to run simultaneously to another program on the chip. This allows information to be passed through a ray tracing slave without stopping the ray tracing process.

The implementation used the CSTools library to ease communication programming. The virtual network shown in fig 5.1 was described via a CSBuild program which allowed CSTools programs to implement an appropriate physical network. This network will probably not be as efficient as if one was manually set up but it eases the programming significantly.

Chapter 6

Implementation

6.1 World model

As explained in the introduction to chapter 2, rays are followed from the eye point through the image plane and into object space. To generate a colour value for each pixel ray is sent through it. This is achieved by dividing the image plane into segments, each of which will correspond to a pixel on screen, then a ray is sent through each segment. More than one ray will be sent through a segment if antialising is being used. This is explained in the last section of this chapter.

Different projections of the objects being displayed can be obtained by varying the eye point and image plane positions.

- A Parallel projection is obtained when the light rays through the image plane are parallel to one another. This is achieved by keeping the distance between the eye point and the image plane very large, i.e., approximating infinity.

- An Oblique projection is obtained when the light rays through the image plane are not parallel to one another.

6.2 Program Structure

The program can be split in three parts, the loader, organizer and ray tracer described in the next three sections.

6.2.1 The loader program

This runs on the host processor. It loads the organizer and ray tracer programs onto the transputer network and then starts them.

6.2.2 The organizer program

This program controls all the ray tracing process by coordinating all the ray tracer programs. It also sends all the ray tracing information to the graphics board to be displayed.

The outline of the program in pseudo code is :

```
Load object and viewing data from disk.
Send object and viewing data to each ray tracer program.
Send each ray tracer program an image line.
Repeat while all lines of the picture have not been received.
    Receive scan line from a ray tracer program.
```

```
    If all scan lines have not been sent.
       Send next scan line to ray tracer program.
    Output scan line received to graphics board.
Terminate all ray tracer programs.
```

Note :

- All communications are performed synchronously, this is because no advantage in the algorithm is gained by using asynchronous communications.
- The screen is split into scan lines. Each scan line is sent to a ray tracer program. Passing the scan line requires a very small amount of communication, i.e., just the y coordinate needs to be sent.

6.2.3 The ray tracer program

This program performs the ray tracing. It is duplicated and run on different processors to speed up the ray tracing process.

The main outline of the ray tracer program is described below in pseudo code:

```
Receive viewing and object information from organizer program.
Repeat.
    Receive scan line number from organizer program.
    For each ray in scan line.
        Calculate and save ray colour.
```

```
    Send scan line colour information to organizer program.


Calculate ray colour.

    If intersection with object

        Find minimum intersection point and surface normal at point.

        Find direct light contribution for incident light ray.

        Calculate ray colour of reflexive ray and add this contrib-
        ution to the incident ray.

        Calculate ray colour of refractive ray and add this contrib-
        ution to the incident ray.

    else

        Give incident ray background colour.
```

A variation of Hall's illumination model is used to calculate the contributions made by direct, reflected and refracted light. This is discussed in chapter 3 so will not be dealt with any further.

6.3 Object type Implementation

Two routines need to be written for each object type.

1. An intersection routine

   This calculates the intersection between a light ray and the object. If more than one intersection is found the one nearest the origin of the light ray should be taken.

2. A normal routine

   This calculates the normal at the point were the ray/object intersection occurs. The normal should point away from the centre of the object by convention.

Three object types have been implemented for the ray tracer: spheres, polygons and NURBS surfaces, their implementations are discussed in the following sections.

6.3.1 Sphere object type

The equation of a sphere can be written as

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 = R^2 \tag{6.1}$$

where

- (x, y, z) is a point on the sphere
- (C_x, C_y, C_z) is the centre of the sphere
- R is the radius.

and an equation of a line, i.e., the light ray, can be written as

$$(x, y, z) = (O_x, O_y, O_z) + \lambda(D_x, D_y, D_z) \tag{6.2}$$

where

- (x, y, z) is a point on the line
- (O_x, O_y, O_z) is the start of the line
- (D_X, D_y, D_z) is a normalized vector of the direction of the line
- λ is a real. At any point on the line the value of λ represents the distance from the origin of the line to that point.

To simplify calculations we can move the sphere to be centred on the origin and compensate by moving the starting point of the line relatively.

i.e.,

$$(NO_x, NO_y, NO_z) = (O_x, O_y, O_z) - (C_x, C_y, C_z)$$

where

- (NO_x, NO_y, NO_z) is the new origin of the line.

Now substituting the line equation 6.2 into equation 6.1 we get the quadratic

$$A\lambda^2 + B\lambda + C = 0$$

where

$$\begin{aligned} A &= D_x^2 + D_y^2 + D_z^2 \\ B &= 2 * (NO_x * D_x + NO_y * D_y + NO_z * D_z) \\ C &= NO_x^2 + NO_y^2 + NO_z^2 - R^2 \end{aligned}$$

Since the direction of the light ray is normalized $A = 1$. Defining B as $2 * D$ we have the solution

$$\lambda = -D \pm \sqrt{D^2 - C}$$

Note we want the minimum positive solution of this equation, since this will be the nearest to the start of the ray. A negative result indicates an intersection behind the start of the ray and thus should be ignored. The point of intersection can easily be found by substituting λ into the equation 6.2.

The normal at this intersection point (P_x, P_y, P_z) is simply

$$\frac{(P_x, P_y, P_z) - (C_x, C_y, C_z)}{R}$$

6.3.2 Polygon object type

There are two types of polygon surface defined, a triangular and a parallelogram type. These can be defined by using the same equation but with different restrictions on the parameters. The definition of the polygon types is :

$$\mathbf{S}(\alpha, \beta) = \mathbf{P} + \alpha\mathbf{A} + \beta\mathbf{B} \tag{6.3}$$

where

- **P** is the starting point of the polygon
- **A** and **B** are the vectors of the sides of the polygon emanating from **P**
- α, β are real numbers

  If $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$ then the equation defines a planar parallelogram

  If $\alpha, \beta \geq 0$ and $\alpha + \beta \leq 1$ then the equation defines a triangle.

Now consider the intersection problem, (Solution taken from N Gatenby's MSc thesis [13]).

We have the equation of the ray line, $\mathbf{O} + \lambda\mathbf{D}$. After translating this line to the origin and the polygon relatively, and equating we get the following

$$\lambda\mathbf{D} = \mathbf{P_N} + \alpha\mathbf{A} + \beta\mathbf{B} \tag{6.4}$$

where

- $\mathbf{P_N} = \mathbf{P} - \mathbf{O}$.

Solving this equation will give us the intersection. Define the unit normal $\mathbf{n}$ to the polygon as $\frac{\mathbf{A}\times\mathbf{B}}{||\mathbf{A}\times\mathbf{B}||}$. Taking the scalar product on both sides of (6.4) with the normal, noting $\mathbf{A}\cdot\mathbf{n} = 0$ and $\mathbf{B}\cdot\mathbf{n} = 0$ because perpendicular gives:

$$\lambda(\mathbf{D}\cdot\mathbf{n}) = \mathbf{P_N}\cdot\mathbf{n}$$

thus

$$\lambda = \frac{(\mathbf{P_N}\cdot\mathbf{n})}{(\mathbf{D}\cdot\mathbf{n})}$$

with this we can work out, from the line equation 6.2, the point of intersection between the line and the plane containing the polygon. To find out if this lies within the ranges of are defined polygon types we need to find α and β values for the intersection point.

Define $\mathbf{e}$ as $\mathbf{e} = \lambda\mathbf{D} - \mathbf{P_N}$ rewriting using equation 6.4 gives

$$\mathbf{e} = \alpha\mathbf{A} + \beta\mathbf{B}$$

Multiplying both sides by the term $\begin{pmatrix} \mathbf{A} \\ \mathbf{B} \end{pmatrix}$ we get

$$\begin{pmatrix} \mathbf{A}\cdot\mathbf{e} \\ \mathbf{B}\cdot\mathbf{e} \end{pmatrix} = C \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

where

- $$C = \begin{pmatrix} ||\mathbf{A}||^2 & \mathbf{A}\cdot\mathbf{B} \\ \\ \mathbf{B}\cdot\mathbf{A} & ||\mathbf{B}||^2 \end{pmatrix}$$

Note

$$\begin{aligned} Det(C) &= ||\mathbf{A}||^2||\mathbf{B}||^2 - (\mathbf{A}.\mathbf{B})^2 \\ &= ||\mathbf{A}\times\mathbf{B}||^2 \\ &= ||\mathbf{A}||^2||\mathbf{B}||^2\sin^2\theta \end{aligned}$$

Thus C is invertible ($Det(C) > 0$) if the vectors $\mathbf{A}$ and $\mathbf{B}$ are non zero and have different directions. Both these criteria are met if a polygon is defined properly, thus α and β can be found by multiplying both sides be C^{-1}. This gives

$$\begin{pmatrix} \alpha \\ \\ \beta \end{pmatrix} = \frac{1}{||\mathbf{A}\times\mathbf{B}||^2} \begin{pmatrix} ||\mathbf{B}||^2 & -\mathbf{A}\cdot\mathbf{B} \\ \\ -\mathbf{A}\cdot\mathbf{B} & ||\mathbf{A}||^2 \end{pmatrix} \begin{pmatrix} \mathbf{A}\cdot\mathbf{e} \\ \\ \mathbf{B}\cdot\mathbf{e} \end{pmatrix}$$

6.3.3 NURBS object type

Two separate methods were implemented to find a NURBS surface intersection, tessellation and subdivision. The tessellation technique splits a NURBS surface into triangles and uses phongs shading algorithm to shade the triangles smoothly. The subdivision technique uses an iteration procedure to converge to the ray/surface intersection point.

Tessellation method

Triangular surface construction

The method used involves taking sample points on the surface after set distances of the u and v variables. These are joined to create a square mesh which represent the surface. Each of these non planar squares can be divided in two to create planar triangles.

We need a way to find x,y,z values on the surface given the parameters u,v. This could simply be done by evaluating the NURBS equation for each u,v point we want. There is a quicker way however by which all points needed on the surface can be calculated all at once, rather than individually.

Consider a NURBS segment of degree n which contains the knots

$$t_i = t_{i+1} = ... = t_{i+n-1}$$

and

$$t_{i+n} = t_{i+n+1} = ... = t_{2n+i-1}$$

where

- i is an integer.

It can be proved (Bohm [7]) that

1. $N_{i,n}(t) = 0 \; \forall i > n, \; t_i \leq t \leq t_{2n+i-1}$.

2. $N_{i,n}(t)$ correspond to the Beinstein Polynomials of degree n

   $\forall 0 \leq i \leq n, \; t_i \leq t \leq t_{2n+i-1}$.

i.e., the NURBS segment between $t_i \leq t \leq t_{2n+i-1}$ can be modelled exactly with a Bézier curve. The control points being $\mathbf{P_i}, \mathbf{P_{i+1}}, .., \mathbf{P_{i+n}}$.
Note $\mathbf{P_i}$ and $\mathbf{P_{i+n}}$ will actually lie on the curve because they are the end Bézier control points.
The Oslo algorithm allows knot values to be added to a NURBS surface. It does this by increasing the number of control points in the NURBS representation. Using this algorithm, and the fact above, points on the surface can be found. The program implemented uses the following steps to find points on a NURBS surface of degree n,m.

- Duplicates all u direction internal knots such that there are n copies of each internal knot.

- Duplicates all v direction internal knots such that there are m copies of each internal knot.

- Split each u direction knot span by the constant `x_split`. At each new interval place n duplicated knots.

- Split each v direction knot span by the constant `y_split`. At each new interval place m duplicated knots.
- Perform Oslo algorithm (Generates rational Bézier patches).
- Read off control vectors that lie on the surface, and that occur at u,v knot points. The end points of the newly formed Bézier patches.

$$\begin{array}{llll} \mathbf{P_{0,0}}, & \mathbf{P_{0,m}}, & \ldots & \mathbf{P_{0,km}} \\ \mathbf{P_{n,0}}, & \mathbf{P_{n,m}}, & \ldots & \mathbf{P_{n,km}} \\ \ldots & & & \\ \mathbf{P_{ln,0}}, & \mathbf{P_{ln,m}}, & \ldots & \mathbf{P_{ln,km}} \end{array}$$

Note

1. The u, v steps are constant only between original knot spans, rather than over the whole surface. Thus this process is not equivalent to evaluating the NURBS equation at set intervals. However using this method normally works out better for evenly parameterized NURBS, as knot spans tend to contain the equivalent amounts of curve complexity.
2. `x_split` and `y_split` are constants set by the user, the more complex the surface the higher these values should be set.
3. The Oslo algorithm is already implemented in a NURBS library [19] used in the project.

Intersection procedure

The basic intersection procedure consists of checking the ray with each triangle in the mesh. This is speeded up somewhat by the implementation of bounding boxes discussed later.

Finding the normal

The normal of the triangle that the ray intersected could be used as an approximation to the surface normal. This however would produce an unevenly shaded object which would not be very satisfactory. The program uses an averaging technique, called Phong's Method, to get smoother shading. It takes a weighted average of the normals to the surface at each of the vertices of the triangle with which the ray intersects.
We have the triangle equation 6.3 and values of α and β for the intersection point. Let $\mathbf{N_0}$, $\mathbf{N_1}$, and $\mathbf{N_2}$ be the normals of the surface at the triangle points $\mathbf{P}$, $\mathbf{P}+\mathbf{A}$ and $\mathbf{P}+\mathbf{B}$ respectively, and $\mathbf{N}$ be the normal used for the intersection point. Let $\mathbf{N}$ be a linear interpolation of $\mathbf{N_0}$, $\mathbf{N_1}$ and $\mathbf{N_2}$.

$$i.e., \mathbf{N} = (a_1\alpha + a_2\beta + a_3)\mathbf{N_0} + (b_1\alpha + b_2\beta + b_3)\mathbf{N_1} + (c_1\alpha + c_2\beta + c_3)\mathbf{N_2}$$

We have three set cases

1. When $\alpha, \beta = 0$ the normal should be $\mathbf{N_0}$.

   therefore $a_3 = 1$ and $b_3, c_3 = 0$.

2. When $\alpha = 1, \beta = 0$ the normal should be $\mathbf{N_1}$.

   therefore $a_1 = -1, b_1 = 1, c_1 = 0$.

3. When $\alpha = 0, \beta = 1$ the normal should be $\mathbf{N_2}$.

therefore $a_2 = -1, b_2 = 0, c_2 = 1$.

Equating these cases gives us the following formulae for smooth shading

$$\mathbf{N} = (1 - \alpha - \beta)\mathbf{N_0} + \alpha\mathbf{N_1} + \beta\mathbf{N_2}$$

The normals at the triangle vertices needed for this formulae could be taken as an average of the joining triangle normals. However this is an approximation, the best results would be achieved if the exact normal of the surface was used for these points. This can nearly always be found by working out the cross product of the u, v partial derivatives of the rational Bézier segments. Finding the partial derivatives follows a very similar process to that described on page 50, but the derivatives of the Bézier patch are calculated instead of the NURBS surface as they are computationally simpler and have already been found via the surface point routine.

There is a case when finding the cross product of the partial derivatives will not work. Consider the Bézier patch shown in figure 6.1. If we want to find the normal at the point u=0, v=0 we would normally perform the calculation.

$$\frac{\partial \mathbf{S}}{\partial u}(v = 0, u = 0) \times \frac{\partial \mathbf{S}}{\partial v}(u = 0, v = 0)$$

but

$$\frac{\partial \mathbf{S}}{\partial u}(v = 0, u = 0) = (0, 0, 0)$$

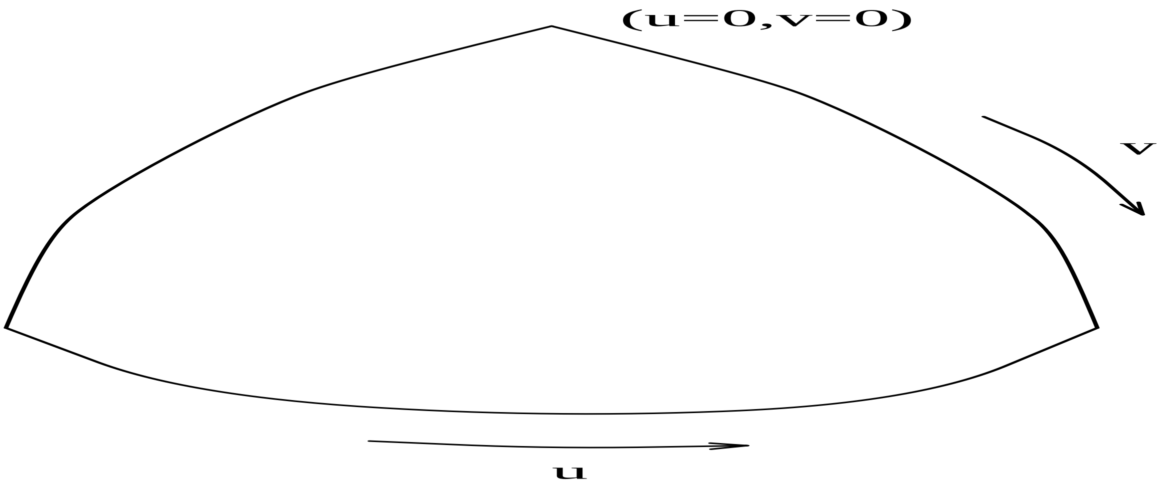

Figure 6.1: Three sided Bézier patch.

as the surface at v=0 is a constant, thus our normal would be (0,0,0) which would be wrong . The correct normal can be found, in this case, by taking the cross product of $\frac{\partial \mathbf{S}}{\partial v}$ at two different values of u.

$$\frac{\partial \mathbf{S}}{\partial v}(v = 0, u = a) \times \frac{\partial \mathbf{S}}{\partial v}(u = b, v = 0)$$

where

- $a \neq b$. ($a > b$ if the normal is to be in the same direction as before).

Similar problems occur if the surface stays constant for the v parameter at a point where a cross product is taken. This can be handled as above but taking partial derivatives with respect to u.

Subdivision method

The subdivision process used in the program is based on Veen's paper [26], this method was described in chapter 4.

The first step of the process involves changing the representation of the surface from NURBS to rational Bézier. This has already been done to calculate the surface points in the tessellation process, so these routines are used. The size of the rational Bézier segments can be controlled by the constants `x_split` and `y_split` as in the tessellation process.

Intersection procedure

Veen's technique is used in a subdivision process to find the exact intersection point between a ray line and a Bézier patch, if one exists. The search routine discussed latter controls the Bézier patches that will be checked for intersection.

Veen's test needs the ray line aligned with the x axis. To achieve this three transformations need to be applied to each point in the control mesh to orientate it to the correct position relative to the ray line.

1. A translation to compensate for the ray start point being put at the origin.
2. A rotation about the y axis.
3. A rotation about the z axis.

These can be combined into one matrix multiplication operation by using homogeneous coordinates.

$R_z \cdot R_y \cdot T$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -O_x \\ 0 & 1 & 0 & -O_y \\ 0 & 0 & 1 & -O_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta\cos\phi & -\sin\theta & \cos\theta\sin\phi & -O_x\cos\theta\sin\phi + O_y\sin\theta - O_z\cos\theta\sin\phi \\ \sin\theta\sin\phi & \cos\theta & \sin\theta\sin\phi & -O_x\cos\phi\sin\theta - O_y\cos\theta - O_z\sin\theta\sin\phi \\ -\sin\phi & 0 & \cos\phi & O_x\sin\phi - O_z\cos\phi \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

- R_z is the rotation about z axis
- R_y is the rotation about y axis
- T is the translation.

Once this has been done the following procedure is applied to try and find an intersection point between the ray and Bézier patch.

```
Find Bezier intersection (Bezier patch).
     If minimum solution distance >= current ray solution distance.
        Return.
     If Bezier patch parameter range accurate enough.
        Let current solution equal average of the patches
        parameter range.
        Return.
```

```
Split Bezier patch, call children left and right, and
calculate parameter ranges associated with these.
If possible intersection with left and right patches.
   If left patch minimum solution < right patch minimum
   solution.
       Find Bezier intersection (left Bezier patch).
       Find Bezier intersection (right Bezier patch).
   Else
       Find Bezier intersection (right Bezier patch).
       Find Bezier intersection (left Bezier patch).
Else
   If possible intersection with left patch.
       Find Bezier intersection (left Bezier patch).
   Else
       Find Bezier intersection (right Bezier patch).
```

Note

- The possible intersection test is an implementation of Veen's test [26]. It checks the positions of the control points to see if they enclose the x axis. The test is described on page 57.
- The minimum solution distance of a patch is taken to be the minimum x control point of that patch.

The splitting procedure for the above algorithm splits in the u and v parameter directions alternately, Veen suggested that this would give the best results. The Bézier splitting algorithm is taken from [4]. This is given below for the u splitting direction.

For j=0 to m.

    For i=0 to n.

        $\mathbf{S_{i,j}} = \mathbf{V_{i,j}}$.

    $\mathbf{T_{n,j}} = \mathbf{V_{n,j}}$.

    For i=1 to n.

        tmp2=$\mathbf{S_{i-1,j}}$.

        For k=i to n.

            tmp1=tmp2.

            tmp2=$(\mathbf{S_{k-1,j}} + \mathbf{S_{k,j}})/2$.

            $\mathbf{S_{k-1,j}}$=tmp1.

        $\mathbf{S_{n,j}} = \mathbf{T_{d-i,j}}$ =tmp2.

where

- $\mathbf{V_{i,j}}$ are the homogeneous control points of original Bézier segment of degree n,m
- $\mathbf{S_{i,j}}$ and $\mathbf{T_{i,j}}$ are the homogeneous control points of the left and right splits of the surface respectively. Each of these new surfaces have the same order as the original surface.

The subdivision in the v direction is very similar, the indices are just swapped.

Convergence criteria

The program keeps splitting the surface in the u and v directions alternately until the u and v ranges come within an acceptable tolerance. The mid value of these ranges is then taken as the intersection point.

A better approach might be to keep subdividing until a subdivided patch approximates a plane defined by three of the corner control points of the patch. If sufficiently close, an intersection point could be worked out between the the newly defined plane and the ray, this would approximate the exact intersection point and normal vector. The problem with this technique is that each control point would need to be checked for its approximity to a plane at each subdivision iteration. This may make the method slower practice than the simple one described above.

The above technique used finds the intersection in terms of the surface parameters. The cartesian coordinates of the intersection, needed for the ray tracing calculations, can be found by simply evaluating the surface at the intersection parameters.

Finding normal

Once the u,v parameter intersection point has been established the normal can be found by working out the cross product of the partial derivatives.

Bounding box implementation

As the technique stands every object in the whole scene has to be checked for intersection with each ray line, and then the results compared to find the nearest intersection to the start of the ray if one exists. This is no problem for the object types sphere and polygon as their intersection techniques are very fast, however any NURBS objects in the scene could slow down the ray tracing process considerably. A way to speed this up is to enclose NURBS surfaces in bounding boxes. Thus each ray intersection problem could first check if it intersected the box enclosing a surface segment before actually checking the surface segment itself. The speed of the NURBS surface intersection procedure can be improved further by having a set of bounding boxes within the main one, these would enclose smaller parts of the surface, but together would enclose the whole surface. Clearly other sets of bounding boxes could be set up within these, and so on producing a hierarchical structure. This technique is used in the program for both the NURBS intersection methods. The bounding boxes used are all parallel to the axes this makes the intersection process between them and a line computationally simpler.

Three problems exist :

1. How to structure and create the bounding boxes.
2. How to efficiently test for intersection with a bounding box.
3. How to search this structure of bounding boxes efficiently to find the section of the surface that the ray line intersects with first.

The first two problems are addressed below the third is encompassed in the minimum distance search routine discussed in the next section.

Bounding box creation

The recursive algorithm used to create a set of bounding boxes for a surface in the program is :

```
Create bounding box.
    If only one surface segment.
        Assign surface segment to bounding box.
        Find minimum bounding extents such that bounding box
        contains all of the surface segment.
    Else
        If surface segments form two dimensional grid.
            Split two dimensional grid formed into four equal size
            sections.
            Create bounding box for each of these sections, set as
            children to current bounding box.
        Else
            If surface segments number less than four.
                Create bounding box for each of these segments
                set as children to current bounding box.
            Else
                Split surface area segments equally into two
                sections.
```

```
        Create bounding box for each of these segments,
        set as children to current bounding box.
    Find minimum bounding extents such that bounding box
    contains all children bounding boxes.
```

The size of the surface segment and how it is represented is dependent on the method.

For the tessellation method a surface segment is set as a portion of the square mesh that results when all the surface points are joined up. The size of the portion is governed by the constant leaf range. Each leaf bounding box will contain an area of the grid less than the size leaf_range × leaf_range.

In the subdivision method each leaf bounding box encloses a Bézier surface segment. i.e., The surface segments obtained when the original NURBS segment is transformed into Bézier segments.

Ray/Bounding Box Intersection

The method used is taken from [16].

The bounding boxes are parallel to the three axis, thus can be defined by two points.

- Box minimum extent : $\mathbf{B_s} = (B_{s1}, B_{s2}, B_{s3})$
- Box maximum extent : $\mathbf{B_m} = (B_{m1}, B_{m2}, B_{m3})$

The ray line can be expressed as

$$(O_x, O_y, O_z) + \lambda(D_x, D_y, D_z)$$

where

- (O_x, O_y, O_z) is the starting point of the line.
- (D_x, D_y, D_z) is the direction of the line (Normalized).
- λ is real.

The algorithm is as follows, it returns the intersection distance :

Set $d_{near} = -\infty$ and $d_{far} = \infty$.

If O lies within $\mathbf{B_s}$ and $\mathbf{B_m}$.

    Line starts within bounding box.

    return 0.

For each plane i = (x,y,z).

    If line parallel to current plane, i.e,.$D_i = 0$.

        If O_i not within limits B_{si} and B_{mi}.

            No intersection.

            Return.

    Else

        Calculate intersection distances to parallel planes

            $d_1 = (B_{si} - O_i)/D_i$

            $d_2 = (B_{mi} - O_i)/D_i$

            $d_{min} =$ minimum of d_1 and d_2

$d_{max} =$ `maximum of` d_1 `and` d_2.

`If` $d_{min} > d_{near}$.

$d_{near} = d_{min}$.

`If` $d_{max} < d_{far}$.

$d_{far} = d_{max}$.

`If` $d_{near} > d_{far}$.

`No intersection.`

`Return.`

`If` $d_{far} < 0$.

`No intersection, box behind ray.`

`Return.`

`Intersection occurred at` t_{near}.

`Return` t_{near}.

If there is an intersection, the line goes in the box at distance t_{near} and leaves it at t_{far}. The intersection point can be found by putting t_{near} into the line equation.

Figure 6.2 shows two examples of using the test in a two dimensional case.

6.3.4 Minimum distance search routine

Individual intersection routines have been discussed above; what has not been discussed is that of finding the minimum intersection distance of the ray line with all the objects in the scene. The bounding boxes of the NURBS complicate the matter in that if a ray hits a bounding box it does not necessarily hit the surface

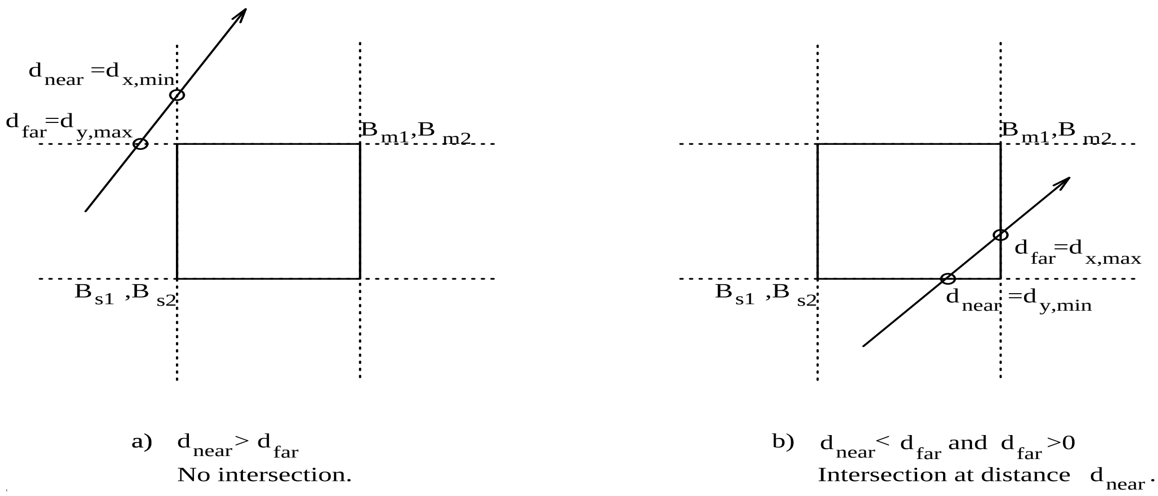

Figure 6.2: Ray/Bounding box intersection examples

contained within it. The search method used is :

```
min_distance = ∞.
Repeat for all objects.
    If object of NURBS type.
        If ray intersects with outer bounding box of NURBS surface
        and the intersection distance < min_distance.
            Add bounding box, intersection distance and object
            number to intersection list.
    Else
        If ray intersects object and object has smaller intersection
        distance than min_distance.
            Add intersection distance and object number to
            intersection list.
            min_distance = object intersection distance.
While intersection list not empty and bounding box record at start
of list.
```

```
    If leaf bounding box record at start of list.
        If intersection with surface.
            Add intersection distance and object number to
            intersection list.
    Else
        Find all children bounding boxes.
        For all children bounding boxes.
            If ray intersects bounding box and intersection
            distance < min_distance.
                Add bounding box, intersection distance and object
                number to intersection list.
    Delete original bounding box record at start of intersection
    list.
if list not empty.
    The first record in the intersection list contains the minimum
    intersection distance with the objects in the scene and the
    object the intersection was made with.
```

The intersection list is always kept in order of increasing intersection distance.

Intersection distance with the bounding boxes is always taken from the point at which the ray first hits the box.

6.4 Anti-aliasing

Fixed Super-sampling, as described in section 3.4.1, is used as an anti-aliasing technique. This is achieved by simply sending four evenly distributed rays out per pixel and averaging the result.

Chapter 7

Results

Two NURBS ray tracing programs were written to demonstrate the work in this thesis. Their implementations were discussed in chapter 6.

They have the following in common :-

- They use an extended version of Whitted's ray tracing model discussed in chapter 3.
- They can display the primitive objects spheres, triangles, parallelograms and NURBS.
- They use bounding boxes around NURBS to speed up the intersection process.

The main difference between the programs is in the way they deal with a NURBS surfaces.

The first program written uses a simple static tessellation method. A NURBS surface is approximated by triangles and these triangles are then displayed using the Phong shading method.

The second program displays a NURBS surface directly, by using an iterative intersection algorithm based on subdivision. It has also been extended so that the ray tracing can be performed on more than one processor. The strategy used for this was discussed in chapter 5.

This chapter shows some photographs of screen images produced by these programs.

The first picture (plate 1) was produced by the tessellation program. The spheres are of sphere object type, while the vase is of NURBS object type.

It shows the main drawback of the tessellation techniques in that it is hard to tell how much splitting is needed, before tessellation, to produce a satisfactory result. In the example the vase is represented quite well, however the refraction of the sphere in the vase is badly represented. A greater number of initial splits would be needed on the vase to get a more accurate result.

Plate 2 was produced by the subdivision program. It shows two rows of reflexive spheres. The top row of spheres are of the sphere object type, while the bottom row are of the NURBS object type.

The difference in the two rows is negligible thus the subdivision method is giving an accurate intersection between the light rays and the spheres.

The final examples, (plates 3, 4 and 5) are all different views of the same data, they were produced by the subdivision program. They show the realistic effects

Vase and two spheres

Plate 1: Tessellation program example

Two sphere rows

Plate 2: Example of the accuracy of the subdivision method

Front show of room

Plate 3: Frontal shot of room

that can be achieved using Whitted's' illumination model. If a more complicated model were used, and texture mapping added, very realistic images could be produced.

The lamp, can, table and pen are all of the NURBS object type, the rest of the objects in the scene; the floor, walls and paper, are parallelograms. The reason for defining these as polygons rather than NURBS was to speed up the production of the pictures.

These pictures were taken from a screen with a resolution of 568×568, four light rays being sent per pixel, for anti-aliasing. Up to six levels of recursion were performed for each light ray. They took between 4 and 8 hours to be produced using twenty six transputers. No doubt these times could be improved upon by optimizing the program used.

Side show of room

Plate 4: Side shot of room

Diagonal show of room

Plate 5: Diagonal shot of table in room

Chapter 8

Conclusion

The purpose of this thesis was to investigate ways of ray tracing Non Uniform Rational B-Spline surfaces. Four main intersection methods were proposed in the thesis; there relative merits will be discussed in the first section of this chapter. The second section of the chapter will discuss the success of the two ray tracing programs written, and improvements that could be made to the implementations.

8.1 Intersection methods

The methods examined in this thesis fall into two different categories: tessellation and exact convergence techniques.

8.1.1 Tessellation methods

These involved approximating the NURBS surface by a triangular mesh. Two main variants were discussed.

- Static tessellation
- Dynamic tessellation

A simple static tessellation method was implemented. A problem found with the implementation was the difficulty in predicting the number of splits needed for an object to be accurately represented. If a dynamic technique were to be implemented this problem would be changed to setting criteria for the whole scene, such as polygon and constant curvature deviation, instead of for each object. This would be easier but problems might still exist with the accuracy of the scene if objects are magnified by reflection or refraction. A way round the problem could be to simply over split the surface by a set amount, which would probably produce accurate pictures for most scenes and views.

One clear advantage of using a tessellation approach is that it is the most common technique used in ray tracing and computer graphics to approximate surfaces, and thus a lot of software and hardware already exist to display a polygonal approximation.

8.1.2 Exact convergence methods

Two main convergence procedures were described: Newton Raphson and subdivision.

Newton-Raphson Method

Yang's and Sweeney/Bartel's methods are based on the Newton Raphson formula. This algorithm has quadratic convergence, but it can be unpredictable in if it converges.

Sweeney/Bartel's method guarantees convergence for a subset of NURBS surfaces (Uniform Cubic B-Spline surfaces) while Yang's method just hopes that the splitting initially done on the surface is sufficient for Newton Raphsons scheme to converge to give a satisfactory result.

Subdivision Methods

The subdivision methods discussed only have linear convergence, so they are slower to converge than the Newton Raphson method. The great advantage of them however is that they will always converge.

Techniques produced by Schneider and Veen were examined. The main difference between the two techniques is the surface used for the subdivision. Schneider performs the subdivision on the NURBS surface while Veen converts the NURBS surface to a rational Bézier before performing the subdivision. Veens is a faster technique because he uses a mathematically simpler surfaces, but they take larger

	Tessellation Methods	
	Static	Dynamic
Accuracy and Memory	Greater accuracy can be attained at the expense of memory. Tends to use more memory than exact methods.	
Speed	Tends to be faster than exact methods.	
User Criteria	Split values for each object.	General criteria for scene.

	Convergence Methods	
	Newton-Raphson	Subdivision
Accuracy	When converges consistently gives very accurate result.	Always gives accurate result.
Memory and Speed	Greater speed can be attained at the expense of memory.	Greater speed can be attained at the expense of memory. Tends to be slower than Newton-Raphson with equivalent bounding box structure.
User Criteria	Subdivision depth for each object.	None

Figure 8.1: Summary of intersection methods discussed in thesis

amounts of memory to store.

A summary of the methods discussed is given in the table 8.1.

8.2 Ray tracing implementations

Two ray tracing programs were written, these mainly differed in the way that they handled the NURBS surface primitive type. The first program written used a static tessellation method while the second used a variation of Veen's subdivision method.

8.2.1 Static tessellation program

This was written as a first attempt at ray tracing NURBS surfaces. It was however a disappointment in the fact that a lot of thought had to go into the split levels that were going to be used for each NURBS surface in a scene. This decision was dependant on several factors including the surface itself, the viewing transformations on it and any reflections or refractions that exist in the scene. This normally turned into a trial and error process. For this reason I went on to implement a subdivision method, as this promised to produce accurate images without need of any split level data.

8.2.2 Veen's subdivision program

The Veen's subdivision method was implemented with very good results as can be seen from the examples in the previous chapter. Although inherently slow the process was sped up by using bounding volumes and parallel processing. Several improvements can be made to the implementation.

Convergence criteria

The method used to see if the subdivision process has converged checks the u and v ranges of the present patch to see if they lie within a certain tolerance, if they do an average of the ranges is taken as the intersection point. This procedure often causes a great amount of subdivision to be done before an accurate enough solution can be found. The basic problems with this is that a lot of time and memory (especially for high order NURBS) are used up reaching a solution.

The memory requirement stems from the fact that every time a subdivision is made and the intersection test reveals that the solution could lie in either of the subdivided patches, one has to be stored while the other is taken for further subdivision. In Schneider's subdivision method he only stores the path to a patch rather than the patch itself thus saving memory. This could be implemented quite easily although it would incur a further time penalty.

One way on saving on the number of subdivisions performed is to try and guess the solution a lot earlier on. This could be done by approximating a patch to a much simpler object such as a plane and finding the exact intersection with this. Careful checks would have to be made to make sure a patch closely approximated the object simpler before this was done.

Bounding volumes

Boxes parallel to the axis are presently used to bound the subdivided Bézier patches. Their limits are calculated from the control points of the patch. Closer bounding volumes would reduce the number of needless patch intersection tests. A possible alternative might be to use the control mesh itself, although a fast intersection test would have to be found first.

Subdividing NURBS surfaces

Transforming the original NURBS surface to a rational Bézier surface multiplies the storage requirements by a factor of sixteen. This stage can be eliminated by performing the subdivision intersection method on the NURBS surface itself

(Veen's method). This can be done by using the Oslo algorithm for the splitting. Unfortunately this algorithm is of greater complexity than the Bézier equivalent and thus will take longer to perform. Another problem with using NURBS surfaces directly is that tight fitting bounding boxes are harder to find.

Parallelism

The use of parallelism in the subdivision ray tracing program proved very beneficial, increasing the speed of the process greatly. The speed up is almost but not quite linear to the number of processes being used. The loss of linearity is mainly caused by the work load not being evenly distributed. A simple way of improving this is to give a smaller area than a line to each processor. If the area is to small however the communication overhead will increase to noticeable proportions.

A better technique would be to split the screen into areas which have a constant workload within each of them. A rough idea of work load in an area can be calculated by projecting the NURBS bounding boxes onto the image plane (Since NURBS surfaces take by far the longest time to ray trace) and calculating what proportion of the area is covered be these projections.

Ray tracing

There are a number of basic improvements that can be made to the ray tracing model implemented.

- The lighting model

  The model used was based on Whitted's model. Improvements can be made by using more complex models such as the Cook/Torrence model [9], which produces more realistic surfaces.

- The speed

  There are certain techniques that can speed up the ray tracing procedure such as undersampling[2], although normally some accuracy will be lost.

Appendix A

Data Entry Format

The data format taken by the programs written is described below. Note all characters are skipped in the data file except <digits>, '.', '-','@' and '+'.

< Data File > ::= < View Information > + < Light Information > +
< Object Information >

< View Information > ::= < Image plane > + < Eye point >

< Image plane > ::= < Centre Cords > + < Up vect > + < Along vect >

< Centre cord > ::= < Position >

< Position > ::= < x > + < y > + < z >

< x >, < y >, < z > ::= REAL

< Up vect >, < Along vect > ::= < Vector >

/* Note < Up vect >, < Along vect > are vectors that start at point < Centre cord > and go to the edge, thus defining the size and orientation of the image plane. */

< Vector > ::= < x > + < y > + < z >

< Eye point > ::= < Position >

< Light Information > ::= < Number of lights > + (< Indiv light info >) *

/* Note < Indiv light info > should be repeated < Number of lights > times. */

< Number of lights > ::= INTEGER

< Indiv light info > ::= < Position > + < Intensity >

< Intensity > ::= REAL

< Object Information > ::= < Number of objects > + (< Indiv object info >) *

/* Note < Indiv object info > should be repeated < Number of objects > times */

< Number of objects > ::= INTEGER

< Indiv object info > ::= < Object type > + < Ambient colour > +
< Diffuse reflection coefficient > +
< Specular reflection coefficient > +
< Specular exponent > +
< Transmission coefficient > +
< Reflection index > +
[< Nurbs defn type > + < Transformation No. >] +
< Surface definitions >

/* Note < Nurbs defn type > + < Transformation No. > included only if < Object type > = < Nurb Type > */

< Object type > ::= < Sphere type > | < Parallelogram type > |
< Triangle type > | < Nurb Type >

< Sphere type > ::= 0
< Parallelogram type > ::= 1
< Triangle type > ::= 2
< Nurb Type > ::= 3

< Ambient Colour > ::= < Red comp > + < Green comp > + < Blue comp >

< Red comp >, < Green comp >, < Blue Comp > ::= { 0 - 255 }

< Diffuse reflection coefficient > ::= REAL

< Specular reflection coefficient > ::= REAL

< Specular exponent > ::= REAL

< Transmission coefficient > ::= REAL

< Reflection index > ::= REAL

< Nurbs defn type > ::= < Nurb lib No. > | < Alpha lib No. > | < Lin lib No. >

< Nurb lib No. > ::= 0

< Lin lib No. > ::= 1

< Alpha lib No. > ::= 2

< Transformation No. > ::= { 0 - < Internal Transformation Matrices > }

/* Note 0 represents no transformation to be performed while greater than 0 represents the number of the transformation to be performed. The transformation matrices are defined within the program. */

< Surface definitions > ::= "@" + (< Surface definition >) * + "@" |
< Surface definition >

< Surface definition > ::= < Sphere defn > | < Parallelogram defn > |
< Triangle defn > | < Nurb defn >

/* Note the decision here should correspond to the < Object type > chosen above */

< Sphere defn > ::= < Position > + < Radius >

< Radius > ::= REAL

< Parallelogram defn > ::= < Position > + < Up orient vector > +
< Along orient vect >

< Up orient vector >, < Along orient vect > ::= < Vector >

< Triangle defn > ::= < Position > + < Up orient vector > +
< Along orient vect >

< Nurb defn > ::= < Nurb lib defn > | < Alpha lib defn > | < Lin lib defn >

/* Note the decision here should correspond to the < Nurbs defn type > chosen above and
< u number of knots > = < u order > + < u number of control points >
< v number of knots > = < v order > + < v number of control points > */

< Nurb lib defn > ::= < u order > + < v order > +

< u number of knots > + < u knot values > +

< v number of knots > + < v knot values > +

< Labelled control points >

< Alpha lib defn > ::= < u order > + < v order > +

< u number of knots > + < u knot values > +

< v number of knots > + < v knot values > +

< Control points >

< Lin lib defn > ::= < u order > + < u number of control points > +

< u knot values > +

< v order > + < v number of control points > +

< v knot values > +

< Control points >

< u order >, < v order > ::= INTEGER

< u number of control points >, < v number of control points > ::= INTEGER

< u number of knots >, < v number knots > ::= INTEGER

< u knot values >, < v knot values > ::= (< Knot value >) *

/* Note the number of knot values should match to the sizes given above */

< Knot value > ::= REAL

< Labelled control points > ::= (< Labelled v constant control point line) *

< Labelled v constant control point line > ::= (< Label > +
< Rational control Point >) *

< Control points > ::= (< v constant control point line >) *

/* Note the number control points should match to the sizes given above */

< v constant control point line > :: = (< Rational control point >) *

< Rational control point > ::= < wx > + < wy > + < wz > + < w >

< wx >, < xy >, < wz >, < w > ::= REAL

< Label > ::= INTEGER + INTEGER

Appendix B

Example program data

A small sample of the data file describing the room in the example pictures is given below.

eye_point 50 50 -400

start 50 50 5

y 0 50 0

x 50 0 0

Number of light sources 3

Sun Light Source one

position -300 200 50

intensity 300

Sun Light Source two

position 50 200 -300

intensity 300

Table lamp source

Position 63.5 40.0 54.0

Intensity 100

Number of objects 8

/* Paper on table */

Object type : parallelogram 1

Ambient colour 90.0 90.0 90.0

Diffuse reflective coefficient 1.0

Specular reflective coefficient 0.0

Gloss constant 5.0

Material transmission coefficient 0.0

Refractive index 1.01

start 45.0 23.0 32.0

up 10.0 0.0 0.0

along 0.0 0.0 14.0

/* Table top description */

Object type : nurb 3

Ambient colour 15.0 5.0 26.0

Diffuse reflective coefficient 0.7

Specular reflective coefficient 0.3

Gloss constant 11.0

Material transmission coefficient 0.0

Refractive index 1.01

Nurb format : 1

Transform : 1

@

U: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

-60.000000, 0.000000, -40.000000, 1.000000,

-60.000000, 0.000000, 40.000000, 1.000000,

-60.000000, -6.000000, -40.000000, 1.000000,

-60.000000, -6.000000, 40.000000, 1.000000,

U: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

60.000000, 0.000000, -40.000000, 1.000000,

60.000000, 0.000000, 40.000000, 1.000000,

60.000000, -6.000000, -40.000000, 1.000000, 60.000000, -6.000000, 40.000000, 1.000000,

U: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

60.000000, -6.000000, -40.000000, 1.000000,

-60.000000, -6.000000, -40.000000, 1.000000,

60.000000, -6.000000, 40.000000, 1.000000,

-60.000000, -6.000000, 40.000000, 1.000000,

U: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

60.000000, 0.000000, -40.000000, 1.000000,

-60.000000, 0.000000, -40.000000, 1.000000,

60.000000, 0.000000, 40.000000, 1.000000,

-60.000000, 0.000000, 40.000000, 1.000000,

U: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

60.000000, 0.000000, 40.000000, 1.000000,

-60.000000, 0.000000, 40.000000, 1.000000,

60.000000, -6.000000, 40.000000, 1.000000,

-60.000000, -6.000000, 40.000000, 1.000000,

U: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

60.000000, 0.000000, -40.000000, 1.000000,

-60.000000, 0.000000, -40.000000, 1.000000,

60.000000, -6.000000, -40.000000, 1.000000,

-60.000000, -6.000000, -40.000000, 1.000000,

@

/* pencil cylinder data */

Object type nurb 3

Ambient colour 1.0 1.0 22.0

Diffuse reflective coefficient 0.1

Specular reflective coefficient 0.0

Gloss constant 5.0

Material transmission coefficient 0.0

Refractive index 1.01

Nurb format : 1

Transform : 3

U: ORDER = 3, CPT NO = 9

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 0.000000

3 ==> 0.250000

4 ==> 0.250000

5 ==> 0.500000

6 ==> 0.500000

7 ==> 0.750000

8 ==> 0.750000

9 ==> 1.000000

10 ==> 1.000000

11 ==> 1.000000

V: ORDER = 2, CPT NO = 2

KNOTS VECTOR ==>

0 ==> 0.000000

1 ==> 0.000000

2 ==> 1.000000

3 ==> 1.000000

4D CONTROL POINTS ==>

14.5, 0.000000, 3.4, 1.000000,

14.5, 2.404163, 2.404163, 0.707107,

14.5, 3.400000, 0.0, 1.000000,

14.5, 2.404163, -2.404163, 0.707107,

14.5, 0.000000, -3.4, 1.000000,

14.5, -2.404163,-2.404163, 0.707107,

14.5, -3.400000, 0.0,1.000000,

14.5, -2.404163, 2.404163,0.707107,

14.5, 0.000000, 3.4,1.000000,

125.0,0.0,3.400000,1.000000,

89.221085, 2.404163, 2.404163,0.707107,

125.0, 3.400000, 0.0, 1.000000,

89.221085, 2.404163, -2.404163, 0.707107,

125.0, 0.000000, -3.40, 1.000000,

89.221085, -2.404163,-2.404163, 0.707107,

125.0, -3.400000, 0.0,1.000000,

89.221085, -2.404163, 2.404163,0.707107,

125.0, 0.000000, 3.400000,1.000000

Appendix C

Important Program Constants

Some important constants used by the programs are described below.

Organ.c

`nurb_split_x`, `nurb_split_y` :

These hold the number of times a NURBS surface will be subdivided in each direction before being put in bounding volumes. The higher these numbers are the quicker the ray/NURBS surface intersection will be, but the amount of memory needed for each NURBS surface will increase.

Load.c

`screen_min_x`, `screen_min_y`, `screen_max_x`, `screen_max_y` :

These hold the physical pixel limits for the display. To prevent distortion these should be to the same ratio as the image plane definition.

Trace_mod.c

`anti_alias` :

When set to 0 no antialiasing is done, when set to 1 antialiasing is performed.

`max_recurse_level` :

Restricts the level to which reflected and refracted rays are calculated.

`double_error` :

Error allowed in double floating point operations.

`float_error` :

Error allowed in floating point operations.

`length_out` :

Error allowed for distance calculations. The setting depends on the image space size. The larger the image space, the larger the value that has to be used.

`converge_error` :

Defines the precision of the NURBS exact intersection calculation.

Bibliography

[1] Salim S. Abbi Ezzi and Leon A. Shirman. Tessellation of Curved Surfaces Under Highly Varying Transformations. *Eurographics*, 1991.

[2] Takaaki Akimoto, Kenji Mase, and Yasuhito Suenaga. Pixel-Selected Ray Tracing. *IEEE CG&A*, July 1991.

[3] Didier Badouel, Kadi Brouatouch, and Thierry Priol. Ray Tracing on Distributed Memory Parallel Computers: Stratergies for Distributing Computations and Data. *IRISA Research Report No.508*, January 1990.

[4] Richard Bartels, John Beatty, and Brian Barsky. *An Introduction to Splines for use in Computer Graphics*. Morgan Kauffman, 1987.

[5] Eric Barten. Data Concurrency & the Computing Surface. *Guidence Notes : IBM UK Scientific Centre*, 1987.

[6] W. Boehm et al. Inserting New Knots into B-Spline Curves. *Computer Aided Design*, July 1980.

[7] W. Boehm et al. *Computer Aided Geometric Design: A Survey of Curve and Surface Methods in CAGD*. 1984.

[8] Elaine Cohen, Tom Lynche, and Richard Riesenfield. Discrete B-Splines and Subdivision Techniques in Computer Aided Geometric Design and Computer Graphics. *Computer Graphics and Image Processing*, 1980.

[9] Robert L. Cook and Kenneth E. Torrance. A Reflectance Model For Computer Graphics. *Computer Graphics*, August 1981.

[10] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[11] Gerald Farin. *Curves and Surfaces for Computer Aided Design*. Academic Press, 2nd edition, 1990.

[12] Akira Fujimoto, Takayuki Tanaka, and Kansei Lwatz. ARTS: Accelerated Ray-Tracing System. *IEEE CG&A*, April 1986.

[13] Neil Gatenby. Radiosity in Computer Graphics : An Alternative to the Hemi-Cube Algorithm. Master's thesis, The University of Manchester, 1991.

[14] Curtis Gerald and Patrick Wheatley. *Applied Numerical Analysis*. Addison-Wesley, 1983.

[15] Andrew Glassner. Space Subdivision for Fast Ray Tracing. *IEEE CG&A*, October 1984.

[16] Andrew Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1989.

[17] Henri Gouraud. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers*, June 1984.

[18] Roy A. Hall and Donald P. Greenburg. A Testbed for Realistic Image Synthesis. *IEEE CG&A*, November 1983.

[19] W T Hewitt. The NURBS Procedure Library. Computer Graphics Unit, Manchester Computing Centre, The University of Manchester.

[20] Sheue-Ling Lien, Michael Shantz, and Vaughan Pratt. Adaptive Forward Differencing for Rendering Curves and Surfaces. *Siggraph*, July 1987.

[21] Bui Tuang Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, June 1975.

[22] Les A Piegl. On NURBS: A Survey. *IEEE CG&A*, January 1991.

[23] Owen F. Ransen. The Art of Ray Tracing. *BYTE*, February 1990.

[24] Bengt-Olaf Schneider. Ray Tracing Rational B-Spline Patches in VLSI. 1987.

[25] Michael A.J. Sweeney and Richard. H. Bartel. Ray Tracing Free-Form B-Spline Surfaces. *IEEE CG&A*, February 1986.

[26] Alle-Jan Von Der Veen. Intersection Test for NURBS. *IEEE Symposium on Computer Archetecture & Real Time Graphics*, June 1989.

[27] Alan Watt. *Fundamentals of Three Dimensional Computer Graphics*. Addison-Wesley, 1989.

[28] Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, June 1980.

[29] Chang-Gui Yang. On Speeding Up Ray Tracing of B-Spline Surfaces. *Computer Aided Design*, April 1987.