TEMPORAL ISSUES IN THE RENDERING 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

M S Preston

Department of Computer Science

October 1992

Contents

List of figures **6**

List of tables **8**

Abstract **9**

Declaration **11**

Dedication **12**

Acknowledgements **13**

1 Introduction **14**

2 Rendering Techniques **19**

2.1 Principles of 3D Rendering 19

2.2 Local illumination model 24

2.2.1 Gouraud Shading 26

2.2.2 Phong Shading 26

2.3 Global illumination models 27

2.3.1 Ray Tracing 27

2.3.2 Radiosity 31

3 Parametric surfaces & NURBS 35

3.1 Bézier Curves 39

3.2 Splines 42

3.2.1 NURBS 47

3.2.2 NURBS Surfaces 49

3.3 Rendering of NURBS 50

3.3.1 Parametric to Geometric 51

3.3.2 Geometric to Parametric 51

3.4 4D NURBS 52

3.4.1 Creating a 4D NURBS 54

4 Temporal Rendering issues 58

4.1 Conventional photographic rendering 58

4.1.1 Approximation of camera model in computer graphics . . . 60

4.2 Motion Blur . . . 63

4.2.1 Lining . . . 63

4.2.2 Multiple Instantaneous exposures . . . 64

4.2.3 Post-process motion blur . . . 67

4.2.4 Distributed ray tracing . . . 71

5 Motion Blur of NURBS **83**

5.1 Multiple Exposures . . . 84

5.2 Post-process motion blur . . . 85

5.3 Distributed ray tracing . . . 88

5.4 Optimisation . . . 89

6 Implementation & Results **99**

6.1 Optimisations . . . 99

6.1.1 Control Volume Intersection . . . 99

6.1.2 Four Dimensional Volume intersection . . . 100

6.1.3 Multiple exposures/Distributed ray tracing . . . 102

6.2 Implementation Issues . . . 103

6.3 Results 104

7 Conclusions **106**

References **116**

List of Figures

2.1 Determining the angle of reflection 21

2.2 Specular and Diffuse reflection 22

2.3 Tracing 2 rays through a scene 28

2.4 Incoming and outgoing energy at a patch 33

3.1 Cubic Bézier curve with 4 control points 39

3.2 The Bernstein Blending functions 41

3.3 A Cubic NURBS Curve 48

3.4 Determining the intersection point between a ray and a patch by repeatedly subdividing the patch alternately along the u and v axis 52

3.5 Simple in-between 56

3.6 The 5 slices which form the 4D NURB 57

3.7 Calculating the 3D NURBS at various points in time 57

4.1 Pin hole & Modern camera model 59

4.2 Abstract view of the raster image generation process 61

4.3 Sampling $f(t)$ above and below the Nyquist limit. The arrows indicate the sampling points, and the circles the sampled value. In the lower sampling the sampled values incorrectly represent a lower frequency sine wave. 75

4.4 Jittering a time sampling pattern 78

5.1 2 Slices of the 4D Control array 95

5.2 Intersecting a ray against part of a 4D Volume 95

List of tables

3.1 Creating the control mesh for a NURBS surface 54

4.1 A prototype time pattern for a 4×4 grid of sample points 80

6.1 Timings for application of the 3 different optimisations to a sample animation 105

Abstract

UNIVERSITY OF MANCHESTER

ABSTRACT OF THESIS submitted by **M S Preston** for the Degree of Master of Science and entitled **Temporal Issues in the rendering of Non-Uniform Rational B-Spline surfaces**

Month and Year of Submission: October 1992

In order to produce high quality computer animated films it is necessary to extend the rendering equations to account for the temporal artefacts exhibited by real cameras, such as *motion blur*.

Existing motion blur techniques rely on supersampling the continuous scene throughout the exposure time in order to approximate the ideal result. These methods are computationally expensive, and do not work well when using parametric definitions of objects, such as Non-Uniform Rational B-Splines (NURBS).

This thesis proposes 3 different optimisations to the motion blur algorithms which

minimise the work required when producing motion blurred scenes containing NURBS.

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.

Dedication

To my parents, for putting up with me.

Acknowledgements

Special thanks to my supervisors, Prof. F. H. Sumner and Terry Hewitt, for help and advice throughout the year.

Thanks to (in no particular order) Mary McDerby, Chris Lilley, Tony Arnold, Andy Grant, Dave Yip, Steve Larkin, Neil Gatenby, Fenqiang Lin, and the multitudes which have added their helpful comments.

Chapter 1

Introduction

Photo realistic computer animation is primarily concerned with the creation of films and videos which appear to have been filmed in a conventional manner. Movies are generated as a result of a process which approximates a continuous scene (the real world) as a finite number of static images.

Like any sampling operation, filming runs the risk of causing aliasing as a result of the limited temporal resolution. However, the undesirable effects of aliasing are reduced as a result of the technology involved in filming. Whilst the filming process is essentially producing a series of snapshots in time, which when presented to the human eye at sufficient speed are resolved into a continuous stream, each snapshot is taken over a finite time period, not the zero length idealised sampling.

As a result of the exposure time real film will exhibit blurred frames when it is recording a fast moving object, as the sampling process being performed is an integration of light energy received over the exposure time on the film, i.e., if the shutter is open $1/125^{th}$ of second, then an object need only traverse 0.01m (2

pixels on a standard TV) in that time for blurring to occur.

The majority of rendering work in computer graphics has been directed towards the production of scenes through an idealised model of a camera. This imitates an instantaneous sampling process, where each frame represents an infinitesimally small sampling time.

Whilst these frames are perfectly acceptable for stationary pictures, or scenes with slow moving objects, the limitations of such a model become apparent when these scenes are used as part of a larger piece of animation. The temporal aliasing caused when viewing a film produced using this technique quickly becomes unpleasant.

The problem lies in the rendering models used in computer graphics, which make no allowance to the temporal effects of filming. By changing the rendering equations used it is possible to emulate more faithfully the integration of light energy through time inherent in the process.

Several different techniques have been developed to include both more realistic models of the camera, and temporal issues in the rendering equation. These provide more realistic animated films, but incur a cost in rendering time. As true integration through time is impossible to perform one is forced to approximate the process through super-sampling. In order to produce the required quality of frames a large number of samples must be taken, which makes this technique expensive.

This cost is only acceptable when simple modelling techniques are used, such as polygonal approximation of objects & scenery in the film. However those

models tend to be cumbersome for capturing realistic scenes, and therefore require inordinate amounts of storage for complicated objects.

More advanced representations exist, where the object is manipulated at a higher level of abstraction, for example items can be represented as parametric surfaces, rather than hundreds of polygons. This allows easier manipulation of the surface as a whole, instead of requiring the user to be concerned with manipulating many hundreds of vertices, but imposes a cost in rendering time.

Consequently most computer animators today must either choose to use more advanced object representations to accurately model the scene, or use approximations to enable the film to be correctly rendered in reasonable time.

An advanced surface representation which is becoming increasingly popular among animators is the Non Uniform Rational B–Spline (NURBS). The NURBS is a parametric tensor product surface which is capable of representing geometric or free form surfaces [23, 29], and which is used extensively in CAD/CAM systems.

NURBS are difficult to render without hardware assistance, and so consequently are exorbitantly expensive to render using temporal anti-aliasing techniques. These costs are beginning to inhibit the use of NURBS in high-quality photo-realistic animation, despite their demonstrable advantages.

The various algorithms developed to model motion blur, and in doing so to address other failings of the limited camera model used in computer graphics, exhibit certain failings when used in rendering motion blurred NURBS.

The temporal anti-aliasing methods can be classified as:

Post-process motion blur induces no additional costs as a result of rendering NURBS, but is only capable of blurring objects on a linear trajectory. The principle advantages of NURBS revolve around the ability to model complicated deformation & motion, which could not be anti-aliased using post-process motion blur.

Modifications to the rendering model are primarily based on super-sampling in the time dimension, in an attempt to approximate the integration qualities of film. As NURBS are expensive to render the cost quickly becomes disproportionate to the benefits.

This thesis will look at the various methods used to implement temporal anti-aliasing, often called *motion blur*, and the techniques used to adapt these methods for use when rendering NURBS surfaces.

Chapter 2 discusses the rendering equations used in conventional computer graphics. The following chapter outlines the parametric surfaces, with a discussion of the 4 dimensional NURBS format used as a basis for this work. A review of the different techniques developed to approximate the effects of motion blur is presented in Chapter 5.

In Chapter 6 these techniques are related to parametric surfaces, and 3 possible improvements to the traditional motion blur algorithms for use with NURBS are proposed, together with a discussion of the relative advantages of each.

These improvements consist of limitations to the amount of work required to super-sample a scene, and reductions in the degree of super-sampling to an acceptable minimum.

Finally, in Chapter 7, timing results from the use of the different optimisations are presented and discussed.

Chapter 2

Rendering Techniques

2.1 Principles of 3D Rendering

In this chapter the principles of three dimensional rendering which forms the basis for the *instantaneous rendering model* are discussed.

Rendering 3 dimensional scenes on a computer is performed by simulating the way we view the world. Light in a scene emanates from a set of light sources. The light then travels in straight lines around the scene, colliding with the scenes contents, and reflecting from any intervening objects to continue in a different direction. As the light wave interacts with an object, its' wavelength changes to take on some of the properties of the surface. The human eye accepts light from the scene, focuses it by passing the light through a cornea & lens, and then measures the incoming light by an arrangement of photo-receptors arranged around the back of the eye.

The different wavelengths detected by the photo-receptors are interpreted by the human brain as colour. Different surfaces affect the light waves in different ways, and in doing so appear to exhibit a different colour to the human eye. In computer graphics we normally approximate the continuous spread of wavelengths by measuring the intensity of light at 3 distinct wavelengths, Red, Blue & Green. From now on we will only consider the behaviour of light at a particular wavelength, but in practice we will need to evaluate the rendering equations once for each of the 3 wavelengths which concern us.

In order to accurately render a scene on a computer it is necessary to model light behaviour in an environment, with particular attention paid to the eye.

The model can be broken down into several distinct areas.

Light sources Each light source in a scene emits light at a particular wavelength, or spread of wavelengths. Therefore, from the human eyes point of view, each light source can be said to emit a 'colour'.

Surface properties When light hits the surface several events can occur. Some of the light will be absorbed by the surface, some will be reflected from the surface, in a direction influenced by the angle of incidence of the colliding light wave & the properties of that surface, and, if the surface is translucent, some of the light will pass through the surface, again in a direction determined by the direction of incidence.

Rendering any moderately realistic scene using such a simplistic model would quickly become expensive, primarily due to the representation of the light sources.

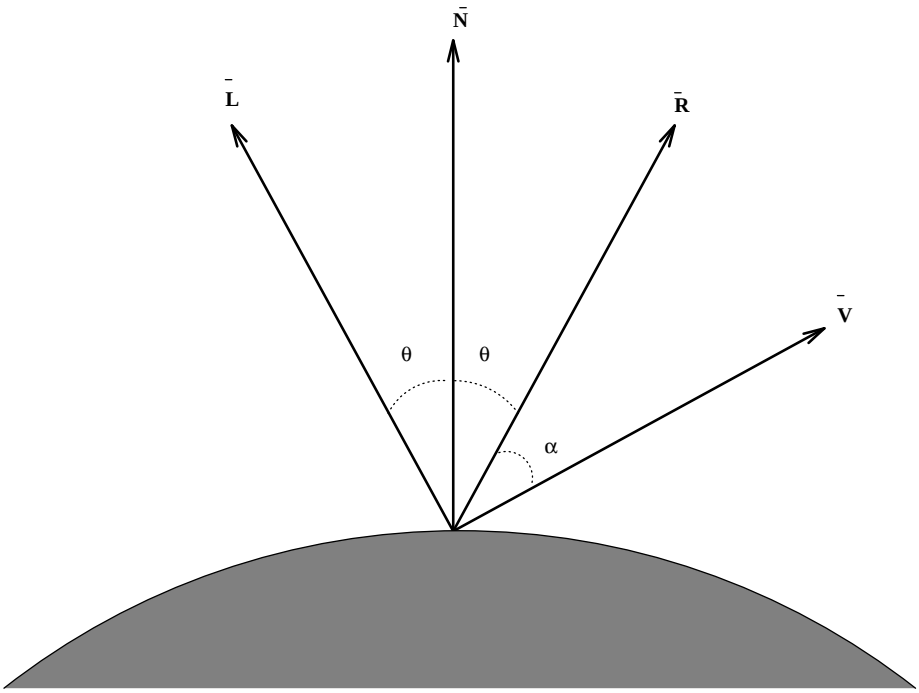


Figure 2.1: Determining the angle of reflection

In any real scene one must consider an infinite number of light waves, from a large set of light sources.

In order to simplify the work it is possible to approximate much of the light in a scene to be *ambient light*, that is light which will affect all surfaces in the scene, without the renderer being concerned about its origin.

Light can be reflected from a surface in different ways, in varying degrees of diffuse or specular reflection (See figure 2.2). Diffuse reflection takes place when the energy of the reflected light wave is uniformly spread across a possible variety of angles. Specular reflection occurs when the reflected energy is tightly focused around a particular angle of reflectance. Specular reflection is most commonly seen on mirrored or highly polished objects.

In order to simplify the model we can pretend that varying quantities of two distinct reflection types are occurring, where in reality they are simply the two

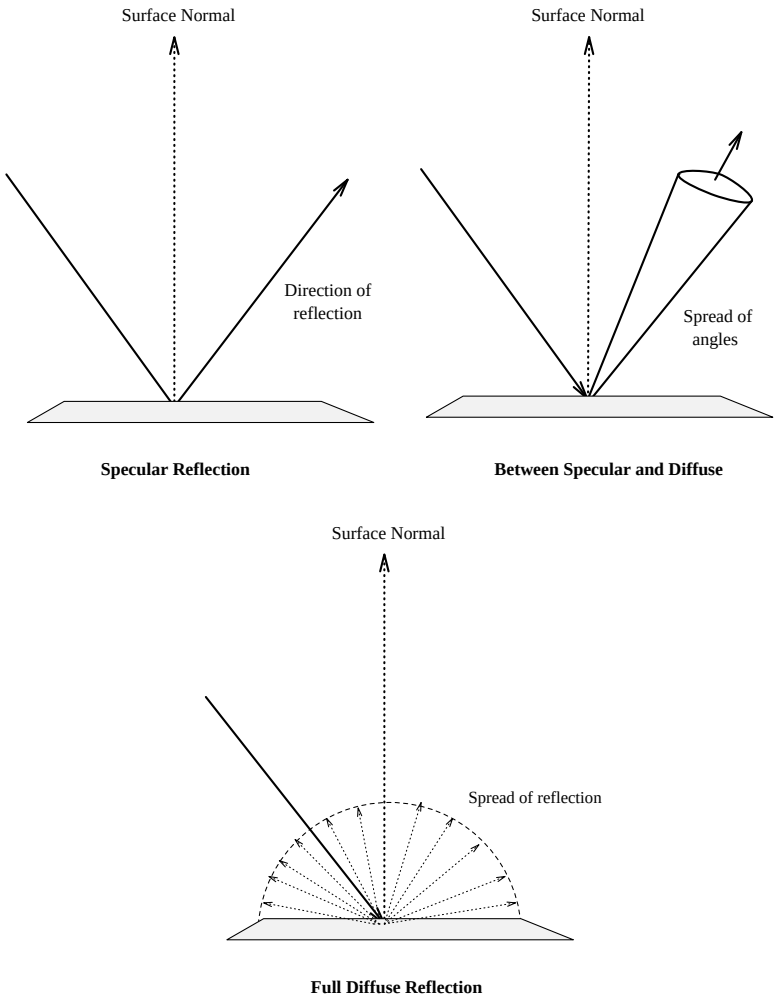


Figure 2.2: Specular and Diffuse reflection

extremes of a continuum.

When light at a particular wavelength collides with a surface it is necessary to determine the quantities of light reflected. As we have split the light in a scene into two categories, diffuse and specular, it is necessary to separately quantify how much of each will be reflected for a particular object. Each individual surface will reflect light differently dependent on its wavelength, therefore it is also necessary to quantify the amount of light reflected dependent on the lights wavelength.

The two terms,$O_{d\lambda}$ and $O_{s\lambda}$, can represent the intensity of light at wavelength λ reflected in a specular or diffuse manner from the surface of the object.

The quantities of energy reflected are dependent on the angle of incidence of the light ray with respect to the surface normal at the point being considered. In addition certain surfaces will appear to reflect differing amount of light dependent on the angle that the viewer is seeing the surface at. To this end it is necessary to also consider the position of the viewer.

The viewer in rendering is normally represented by a perspective projection of the three dimensional view onto a two dimensional plane, the screen. It is necessary to consider the distance of the screen from the scene, the position of the 'eye', which is normally used as the centre of projection, and the focal distance.

Most computer graphics ignores the effect on the image of the lens in the human eye, and so all the image is equally in focus.

Whilst the theoretical model of the light behaviour in a scene is useful there are times when such accuracy is not required in computer graphics. Simplifications

can be made to the model which reduce the amount of work which needs to be done to render a scene.

In order to quicken the visualisation of scenes it is not necessary to model the interaction between objects in the scene, only between light sources and the particular object we are concerned with. This is called the *local illumination model* as opposed to the full *global illumination model*. As far as the production of temporally anti-aliased pictures, we are only concerned with the global illumination model, but, as many of the techniques used are derived from the simpler local model it is worthwhile outlining its principles.

2.2 Local illumination model

One of the main benefits accrued from the use of a local illumination model is that it is possible to render each object in isolation, producing a pixel map which can then be combined with other maps to produce the final picture. This final stage, which determines which parts of the objects are visible, called the hidden-line & hidden surface removal problem or *HLHSR*, is often implemented through use of a z-buffer [13].

As the rendering equation used in a local illumination model need only concern the object in question and the light sources this simplified rendering model can be expressed for the intensity of a point on each surface, at the wavelength λ.

$$I_\lambda = \underbrace{I_{a\lambda} k_a O_{d\lambda}}_{Ambient} + \underbrace{\sum_{i=1}^{m} f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\mathbf{N} \cdot \mathbf{L_i}) + k_s O_{s\lambda} (\mathbf{R_i} \cdot \mathbf{V})^n]}_{Specular}$$

where

m Number of light sources

$I_{a\lambda}$ Intensity of ambient light at wavelength λ

k_a Ambient reflection coefficient

k_s Specular reflection coefficient

L Vector to light source

N Surface normal

f_{att_i} Light source attenuation factor

I_p Point light source intensity

$O_{d\lambda}$ Objects diffuse reflection intensity at wavelength λ

$O_{s\lambda}$ Objects specular reflection intensity at wavelength λ

R Direction of reflection

V Direction of viewer

λ Wavelength of light being considered

The local illumination model is most commonly used for debugging the composition of scenes, and as such is normally used on simplified objects which consist solely of polygon meshes.

Rather than evaluating the rendering equation for each pixel on every object two techniques have been developed which are aimed at generating pictures of

scenes containing polygonal meshes. Both methods are particularly well suited for implementation in hardware, something which has enhanced their popularity.

2.2.1 Gouraud Shading

Gouraud shading [15] enables efficient implementation of local illumination by only calculating the I_λ at the vertices of the polygon in question. From there the intensity of light is linearly interpolated between the vertices to give a spread of intensities down the side of the polygon. As the same is done for the opposing edge it is also possible to linearly interpolate across the polygon to shade extremely quickly.

2.2.2 Phong Shading

Like Gouraud shading, Phong [22] shading only evaluates the equation at polygon vertices. However the Phong technique calculates the surface normals at the vertices, instead of the intensity directly. From then on it is only necessary to calculate a small section of the equation again for each pixel.

As Phong shading takes into account the surface normal inside the polygon, unlike the Gouraud technique, it is possible to render highlights which fall inside a polygon.

2.3 Global illumination models

Global illumination needs to accurately model the full interaction of light and surfaces in the scene. Unlike local illumination it is possible to represent most of the artefacts which contribute to a photo-realistic image, such as reflections, shadows, refraction.

There are two main global illumination methods, ray tracing & radiosity.

2.3.1 Ray Tracing

The technique of ray tracing [3, 33, 2] centres around the image plane. Light is modelled as a series of rays, which each have a start position and direction vector, together with a wavelength. An infinite number of light rays leave each light source.

When a light source collides with an object it is necessary to determine the result. The lights' wavelength is normally influenced by the surface, and then either reflects or refracts dependent on the surface properties.

When the light ray eventually collides with the image plane, the wavelength of the ray is measured, and this contributes to the image.

Due to the infinite number of rays leaving each light source the cost of implementation of such a technique would be prohibitive. Therefore an *inverted* approach is taken. In a full implementation the energy collected at each pixel is influenced only by the rays which collide with that portion of the plane. Of all the rays in

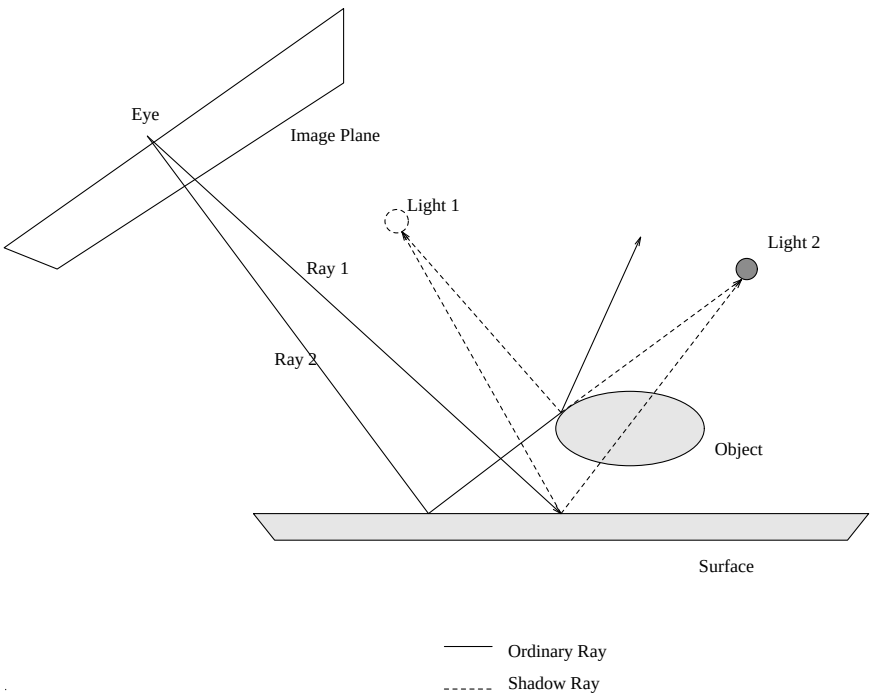


Figure 2.3: Tracing 2 rays through a scene

the scene only a small number of them will collide with that portion of the plane, as only a certain combination of reflections off objects will direct the ray towards that point.

Therefore it is possible to approximate the effect of this brute force ray tracing by only tracing those rays which are guaranteed to collide with the plane. This is done by tracing the ray from the eye point towards the plane, and then out into the scene. Whilst this inverted technique gives only an approximation to the picture which would be generated by a full implementation, the picture quality is usually acceptable.

For each ray we keep a track of the wavelength & direction. This is done until we deem the ray to be contributing nothing meaningful to the energy of the pixel it passed through. This decision is usually made either by

- The ray having hit a certain number of objects. If we were ray tracing using

the brute force solution then we would be modelling the energy drop off as each surface absorbed some of the energy of the light ray which it began with. If the ray hits many objects then it would have lost so much energy that it would contribute nothing useful to the plane when it hit it.

In this simplified method we are simply modelling this in reverse. If it collides with many objects then by the time it hit the plane the light would have dimmed sufficiently for it not to be worth tracing it any further.

As ray tracing is commonly implemented as a recursive algorithm this is implemented by limiting the depth of recursion.

- The ray leaving the scene. Here we can assume that any collision would contribute little to the energy of the light, as by the time it reached the plane the light intensity would would have dropped.

The collision of a ray and an object causes a decision to be made as to how the light ray reacts. First a ray is cast towards each of the light sources. If that ray reaches the light source without first colliding with something else in the scene then that point receives light from there. Otherwise the point is in shadow from that light source.

Next it is necessary to determine the directions of reflected & refracted light. A variety of algorithms exist to help model the surface properties of objects, all of which alter in some way the direction & wavelength of the transmitted light. These rays are then traced in exactly the same manner as the intersecting ray.

One sample algorithm, the Phong illumination model [22], deals with the relationship between the quantity of energy reflected, and the difference between

the angle of reflection and the direction of the viewer (α). This model assumes that the most energy is specularly reflected as α is 0, and falls off rapidly as α increases. The fall-off is approximated by the function $\cos^n \alpha$, where n is the materials *specular reflection exponent*. The higher the value of n the sharper the reflections become, with $n = 1$ providing a gentle fall-off. If we were modelling a perfect reflector such as a mirror we would want to represent n as ∞.

The full Phong illumination model is then,

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{i=1}^{m} f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} \cos\theta + W(\theta) \cos^n \alpha] \tag{2.1}$$

The Phong illumination model tends to produce smooth plastic looking surfaces, which may not be what the user desires. More complicated illumination models exist, which describe the surface in a more realistic manner.

For example the Cook-Torrance model [30] assumes each surface is inherently rough, and is composed of many thousands of micro facets, each of which will possess a different normal. This effect is quantized as the proportion of micro-facets which are shadowing other facets. This geometric factor G,accounts for the degree of 'roughness', ranges from 0 (total shadowing) to 1 (no shadowing).

A combination of the direction of light, reflection, viewing and G determine the specular reflections observed. The *Fresnel* [13] function is used as part of the illumination model to relate incident light to reflected light for the perfectly smooth surface of each micro-facet.

By varying the formulation of the Fresnel function it is possible to model the physical characteristics of materials, and therefore render them in a physically

realistic manner.

Once the ray has been traced, i.e., when either the ray passes out of the scene or intersects a certain number of objects, the colour at that pixel can be determined, and then the next ray can be considered.

The entire ray tracing process can be viewed as an evaluation of a tree structure, where the branches of the tree denote intersection with surfaces requiring tracing of further subtrees, and the leaves either signify rays leaving the scene, or rays which do not reflect, refract or transmit. This tree is normally implicitly represented in an implementation by the recursive algorithm.

Like any discrete sampling process ray traced images can suffer from aliasing, though various techniques exist to limit this.

Ray tracing, using a variety of surface transmission models, handles specularly reflecting objects well. In the real scene a reflected light wave is actually spread out through a cone, and so the abstraction of this cone to a single ray begins to become unsuitable once diffusely reflecting objects are present.

2.3.2 Radiosity

Unlike ray tracing, radiosity models the behaviour of light in the scene in a view independent manner. The technique works by modelling the distribution of energy, in this case light energy, about the scene.

The scene is normally split into a finite number of *patches*, which are then treated as the smallest area of interest. Each patch both receives energy from other

patches, and emits energy into the scene, in the form of light. Due to conservation of energy any light emitted by a patch must be absorbed elsewhere in the scene. Therefore if we view a particular patch we can say the radiosity (energy present) at the patch consists of the energy emitted, plus any energy received from other patches in the scene. Note that the energy may be coming from either light patches or patches which are simply reflecting the light.

The behaviour of energy at the ith patch of N in the scene can then be expressed as

$$A_i B_i = A_i E_i + \rho_i \sum_{j=1}^{N} B_j F_{ji} A_j$$

where

A_i is the surface area of the ith patch.

ρ_i is the coefficient of reflectivity for the ith patch.

B_i is the radiosity of the ith patch

E_i is the energy emitted per unit area per unit time by the ith patch

F_{ji} is the form factor of the jth patch relative to the ith patch

An implementation of the *progressive refinement* radiosity algorithm could then consist of several iterations through all the patches in the scene. At each patch we consider the energy emitted, which is high if this patch is a light source, and the energy received. In order to calculate the latter we need to run through all the other patches in the scene, finding how much of their energy they can contribute to this current patch.

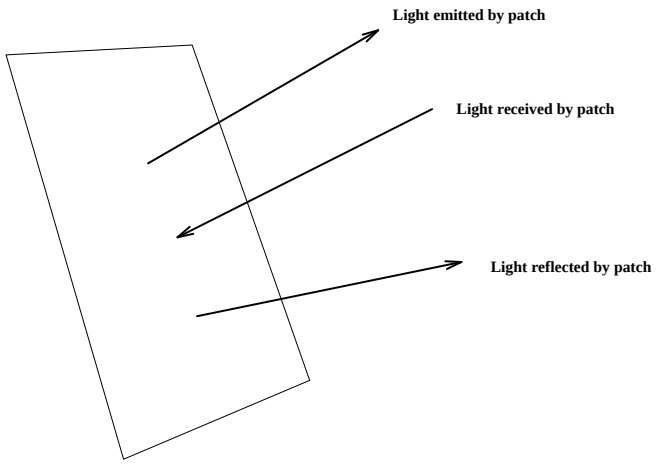


Figure 2.4: Incoming and outgoing energy at a patch

Firstly we can assume that no partial occlusion of patches occur, i.e., energy either reaches the patch or it doesn't. The calculation to determine the amount of energy incoming is based on the attitude of the patches to each other, the distance separating the patches, and the surface area of the current patch. The result of the calculation to determine the attitude & distance is usually called the *form factor*.

Once the calculation process has completed a satisfactory number of steps the scene is stored for rendering. The rendering process then simply consists of converting the energy measured at each patch into a display-able colour.

Then the rendering stage can simply consist of displaying polygons, possibly using Gouraud shading [15].

The radiosity technique accurately models diffuse inter-reflection in a scene, though it cannot model specular reflections when used on its own, as it is a view independent rendering method. This allows it to be used in situations where the scene changes little, but the viewpoint can move, such as in virtual reality or architectural walk-throughs.

Radiosity & ray tracing are frequently combined to help alleviate the failings of each technique when used on its own. Normally this takes the form of a 'second pass' to the rendering process, where after the view-independent radiosity stage has been completed, a view-dependent ray tracing stage is performed to take into account the specular properties observed at each point.

Chapter 3

Parametric surfaces & NURBS

In order to be able to render high quality near-photo realistic images it is necessary to have some way of modelling real world objects. Most computer graphics concentrates on representing scenes as collections of polygons, possibly grouped hierarchically to help facilitate manipulation.

However as the power of computers has increased, global rendering models such as those outlined in chapter 2 have been developed, and these have illustrated the limitations of such a primitive surface modelling method.

Alternative techniques have been developed to address these failings

- All polygonal mesh representations are only approximations to curved surfaces, and however good they are it will always be possible to find cases where the model is insufficient.
- In order to model complicated objects large numbers of polygons are needed, and the storage requirements quickly become costly.

- Once an object has been approximated to a collection of polygons it becomes difficult to perform modifications with respect to the original object. Any modifications must be made to the polygons, which may be difficult and time-consuming.

The need for accurate curve modelling with minimum storage cost led to the development of parametric boundary-representation, or B-Rep, models. The parametric representation for a curve is not the only possible one usable by computer graphics, but it has become popular for several reasons.

Explicit representations, such that $z = f(x, y)$ are initially appealing but have several flaws. Most explicit representations aren't rotationally invariant. Consequently projection of such curves would require conversion to another format before they could be displayed. This in itself would not preclude explicit forms for internal systems use were it not for other problems. It is impossible to represent self-intersecting curves in an explicit form, a limitation which reduces the set of representable curves. Finally modelling a vertically sloped curve through an explicit representation causes problems, as a slope of infinity is difficult to express.

Implicit equations such as $f(x, y, z) = 0$ are also possible representation forms. However it is difficult to model, as part of the equations themselves, limits of the extents of the curve. Implicit equations are best used to model infinitely extending artefacts, and to use them to model the majority of general shapes a user may require in a system, requires the addition of stored constraints. Again this does not totally preclude their use, but it complicates any system using them.

Parametric equations model a curve as 3 equations, based on a single parameter, t, e.g.,

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x, \tag{3.1}$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y, \tag{3.2}$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z, \quad a \leq t \leq b \tag{3.3}$$

Whilst cubic curves are shown the polynomials can be of any degree, though in practice cubic curves are often used. Higher degree curves are most expensive to render, and tend to be more unstable, both factors which limit their use.

The parameter t is limited to the $[a, b]$ interval, to enable us to deal with finite elements of the curve. The curve is alternatively represented as a matrix of coefficients,

$$C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & b_z \\ d_x & d_y & d_z \end{bmatrix}$$

and together with,

$$T = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

allows us to represent the curve as

$$\mathbf{Q}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = TC \tag{3.4}$$

The derivative of $\mathbf{Q}(t)$ is the parametric *tangent vector* of the curve. Applying to this we obtain;

$$\begin{aligned} \tfrac{d}{dt}\mathbf{Q}(t) &= \begin{bmatrix} \frac{d}{dt}x(t) \\ \frac{d}{dt}y(t) \\ \frac{d}{dt}z(t) \end{bmatrix} \\ &= \tfrac{d}{dt}TC \\ &= C \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 3a_x t^2 + 2b_x t \\ c_x 3a_y t^2 + 2b_y t + c_y \\ 3a_x t^2 + 2b_z t + c_z \end{bmatrix} \end{aligned} \tag{3.5}$$

The flexibility of parametric representation tends to make generation of a given curve somewhat difficult, as there are always a number of ways of modelling a curve. In order to simplify the generation of curves several distinct varieties of parametric curves have been developed. NURBS are merely one variety of representation, and in order to illustrate their advantages it is necessary to review some of the simpler forms.

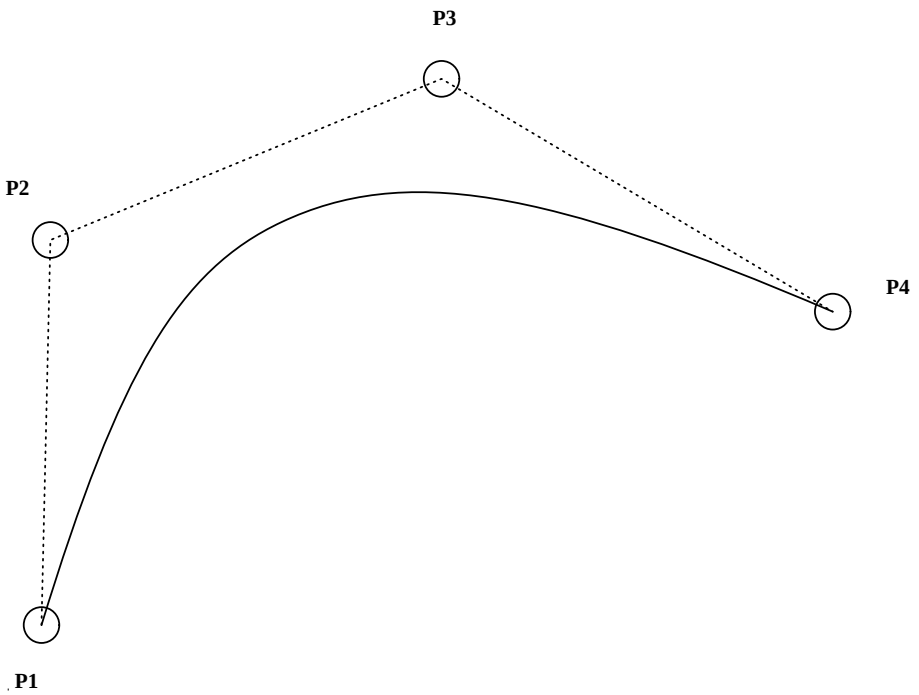


Figure 3.1: Cubic Bézier curve with 4 control points

3.1 Bézier Curves

One form of the polynomial curve segment, the Bézier curve, named after Pierre Bézier, indirectly specifies the behaviour of the curve at the end points by specifying two points which are not on the curve [4, 5].

If we have a cubic Bézier curve with 4 points, then the curve will interpolate the beginning and end points, and approximate the other two. So given the points $P_1 \ldots P_4$ then the tangent vector at the beginning and end points of the curve will be $P_2 - P_1$ and $P_4 - P_3$ respectively.

This 4 point Bézier curve will consist of a geometry vector,

$$G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

and the *Bézier basis matrix*. This specifies the effect each point has on the behaviour of the curve, in this case all the controlling points in the geometry vector exert an equal pull on the curve.

$$M_B = \begin{bmatrix} -1 & 3 & 3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.6}$$

The four polynomials $B_B = M_B T$ are known as the cubic *Bernstein polynomials* and are shown in figure 3.2. The curves denote the effect each of the controlling points in the geometry vector have on the curve. For example at $t = 0$ all curves other than B_1 are zero, so the curve is solely effected by P_1, which is why the curve interpolates the start point.

In order to represent more complicated curves we use more points in the geometry vector, and increase the degree of the polynomials. However due to the basis matrix all of those points exert equal pull on the curve. In a 4 point curve this makes little difference, but the more points we add, the harder it becomes to control the path of the curve. Every time a point is changed the entire curve moves, necessitating many more changes to points to offset any undesired effects.

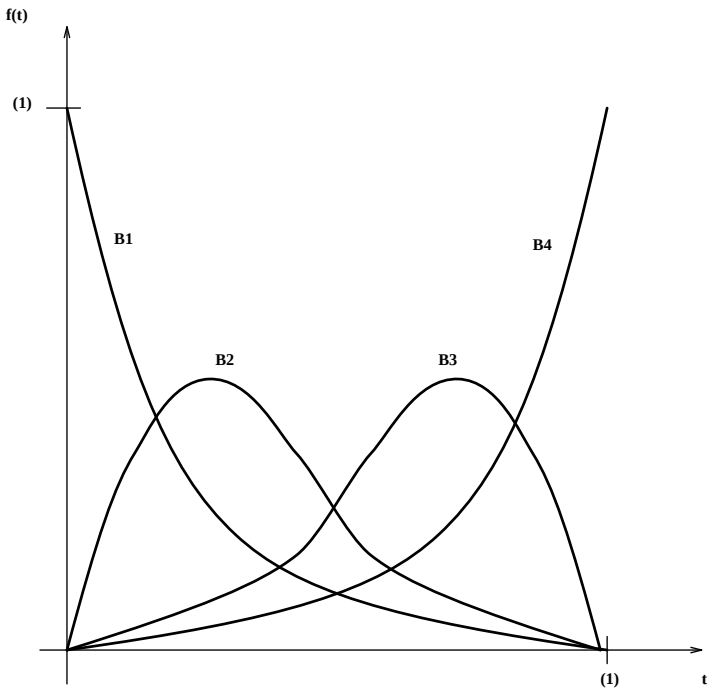


Figure 3.2: The Bernstein Blending functions

Consequently representing complicated shapes using a single Bézier curve becomes difficult. However we can represent the larger curve by using multiple Bézier curve segments, and duplicating the control points to make the curves join.

This patchwork pattern of curve segments is commonly used in parametric representations. Representing an arbitrary curve using a single polynomial is a complicated task which is likely to result in a complicated high-order equation, which will be expensive to work with. However by using small segments of low order piecewise polynomials the same result can be obtained at reduced cost.

In order to make the different segments appear to form one curve it is necessary to ensure more than just the end points meet. A range of possible degrees of continuities of curves are both possible, though in practice we are mainly concerned with curves which are C^2 continuous.

Whilst the Bézier representation cannot model many curves without resorting to several segments they have interesting characteristics which make them useful tools to the writer of a computer graphics system.

It can be demonstrated that as the Bernstein polynomials sum to 1 for any value of t, the Bézier curve is a weighted sum of the control points. This means that all portions of the curve will fall within the *convex hull* [13] defined by the controlling points. This enables the convex hull to be used as a simplified clipping boundary by the computer system. Calculating the trajectory of any parametric curve is necessarily slower than an approximation to the curve based on straight lines. Consequently any tool to aid in the efficient display of a parametric object is useful.

In order to use parametric curve & surface representations as a general storage form in a graphics system we need to be able to represent any curve which the user might wish to define, in a manner which encourages easy and intuitive manipulation.

In order to do this we need to tackle some remaining limitations on surface representation. The family of curve representations based on the spline enable this.

3.2 Splines

Like the Bézier curve, Splines are polynomials. The word *spline* comes originally from a method of bending wood used in ship-building, where wood strips are

soaked until pliant, glued together and then bent into shape, and held there between pegs, or *knots*, fixed into position. As the wood dries the shape becomes fixed.

The mathematical natural cubic spline is a continuous cubic polynomial which intersects none of its control points. Like the Bézier curve splines do not exhibit local control, that is moving any one point in the geometry vector will affect the shape of the entire curve.

Recall that the reason for this non-locality is the Bernstein basis matrix, through which all points exert some pull on the curve. By changing the matrix, and using one which embodies local control we can represent a more generally useful spline.

B(asis)-splines are curves which exhibit locality of control. Consequently they are cheaper to render, as changing one are of the control point vector will only cause us to have to recalculate a portion of the full curve.

On the curve, at the junction of each segment, there is a join point, or *knot* between the segments, at the parameter value t_i. This value is called the knot value, and there will be a non-decreasing sequence of knot values for the curve, which specify how the curve is constructed based on its control points. The beginning & end points of the curve are also termed knots, so there will always be knots at the beginning and end of the parameter range. Note that it is conventional, though not strictly necessary for knots to always lie in the range 0 ... 1.

For a uniform B-spline each of the knots must be evenly spaced, consequently the basis matrix remains simple. Like the cubic Bézier, each curve segment is

affected by four control points, but they are always the 2 control points either side of the current point on the curve.

Curve segment i is defined by the four control points P_{i-3}, P_{i-2}, P_{i-1} & P_i. The *B-spline geometry vector* for the curve segment Q_i is then

$$G_B = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, 3 \leq i \leq m$$

The basis matrix for the uniform B-spline is

$$M_{Bs} = 1/6 * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{3.7}$$

In this case the blending functions are the Bernstein polynomials. As the polynomials always sum to 1, so the B-spline curve shares the Bézier curves' property of always remaining within the convex hull formed by the control points.

Most of the uses of B-splines call for interpolating end-points, and this is implemented by repeating the knots at the end points. As the influence of the curve is drawn from the four nearest points for a cubic curve we force a B-spline to interpolate a point by making the curve at that point draw on the same 4 points.

As a B-spline is not always cubic we can say in the general case that in order to force the curve to interpolate a point on the curve we repeat the knot at that

point k times where k is the order of the B-spline, and corresponds to the degree + 1.

It is difficult to add control points to a uniform B-spline in order to help manipulate it. New knots cannot be inserted into a uniform sequence without re-calculating that sequence, which is usually complicated, or produces a non-intuitive control point list.

Non uniform B-splines, however, allow us greater control over continuity, and knots & control points can be added in sequence to help the user manipulate the B-spline intuitively.

A non-uniform knot sequence results in the inability to use a single basis matrix. The curve segment, Q_i then, is defined by four control points, P_{i-3}, P_{i-2}, P_{i-1} & P_i and by four blending functions, $B_{i-3,4}(t)$, $B_{i-2,4}(t)$, $B_{i-1,4}(t)$, $B_{i,4}(t)$.

When a knot is repeated a sufficient number of times (4 for a cubic B-Spline), the curve segment, Q_i, becomes a single point. This point segment allows discontinuities to be represented in a non uniform B-spline easily.

The functionality of the basis matrix is provided in the non-uniform B-spline functions recursively defined in terms of a lower order blending function. The functions are dependent on the interval between the knots, so knots placed further away from the point on the curve in question will exert less influence on the curve than a closer knot.

Like any spline a non-uniform B-spline can be represented by a collection of polynomials of any degree. The recursive definition of the blending functions will

then work for any degree of polynomial, though obviously higher degree B-splines will take long to evaluate. In B-spline terminology the order of a B-spline is the degree + 1, so cubic B-splines are said to have order 4.

The basis functions are defined using the recurrence relation:

$$N_i^0(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & otherwise \end{cases} \tag{3.8}$$

$$N_i^k(t) = \frac{(t - t_i) * N_i^{k-1}(t)}{t_{i+k} - t_i} + \frac{(t_{i+k+1} - t) * N_{i+1}^{k-1}(t)}{t_{i+k+1} - t_{i+1}} \tag{3.9}$$

Again the blending functions sum to 1, and consequently the non-uniform B-splines always lie within the convex hull formed by the control points. Note that it is quite common for multiple knots to occur in the knot sequence, and this would result in a division by zero in the blending functions. For the purposes of the blending functions $0/0 = 0$.

Its worth noting that by presenting a knot sequence of $[0, 0, 0, 0, 1, 1, 1, 1]$ with 4 control points the non-uniform B-spline is a Bézier curve. The ramifications of this are that it is always possible to convert a non uniform B-spline curve into a sequence of Bézier curve segments by repeating the knots, and then treating the relevant subset of knots & control points as the control points for the Bézier curve. This technique is frequently used to help in the rendering of B-Splines, as the uniform blending functions in a Bézier representation are usually easier to solve.

3.2.1 NURBS

Non-uniform B-splines can represent a greater set of possible curves than uniform B-splines, but there are still limitations. Namely

- Non-uniform B-splines use conventional (x, y, z) representation for control points, and as a consequence are not closed under a perspective transformation.
- In addition they can only approximate all of the possible conic sections.

By changing the control points to be defined in homogeneous coordinates, we can think of the curve as being placed in homogeneous space,

$$Q(t) = \begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \\ W(t) \end{bmatrix}$$

The principle benefit to using homogeneous coordinates is that it enables us to efficiently transform the control point matrix using 4x4 matrices, including perspective transformations. Conversion from homogeneous space to 3-space involves dividing by W(t), so we are storing each control point as

$$\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{bmatrix} \tag{3.10}$$

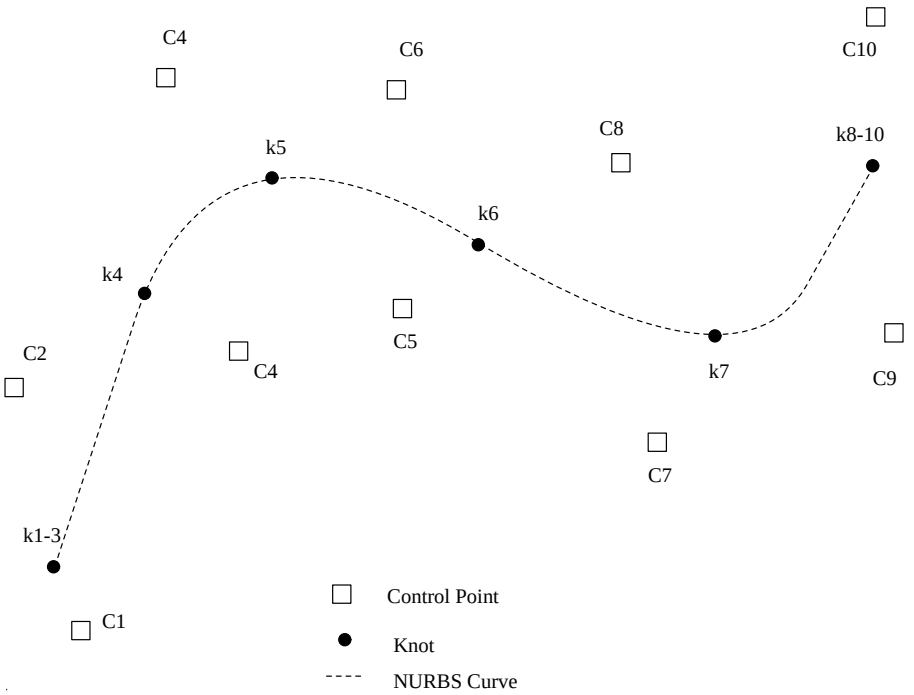


Figure 3.3: A Cubic NURBS Curve

When the spline being represented rationally is a non-uniform B-spline we call the result a non-uniform rational B-Spline, or *NURBS*.

Thus by representing curves as NURBS curves we are able to

- Represent infinite slopes.
- The resulting curve is closed under a 4x4 transformation.
- We may add knots & control points at any point in the NURBS without having to alter any existing values.
- Permit multiple curve segments to pass through given x & y, something difficult to achieve using explicit representations.
- When representing many objects the NURBS representation will require less storage space than a polygonal approximation to that object.

- Can interpolate or approximate points as required
- Provide local control to the user, thereby supporting intuitive modification

3.2.2 NURBS Surfaces

Parametric surfaces are a generalization of parametric curves. Instead of having a single parameter prescribing the path of the curve, two parameters, u & v, are used to denote a position in parametric space.

There are several distinct ways of representing parametric surfaces. Bézier surfaces, in much the same way as Bézier curves, may consist of an arbitrary number of control points, but the Bernstein blending functions mean that for large numbers of points the surface will tend to become flat.

Consequently it is common for surfaces constructed using Bézier surfaces to be represented as a collection of Bézier patches. Again the problem of ensuring continuity across the patch boundaries appears, which forces the user to do more work to create a smooth surface.

A NURBS surface shares the same benefits as a NURBS curve, but extends them into a separate parametric dimension.

The equation for a NURBS surface is defined as

$$\mathbf{S}(u,v) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathbf{P}_{i,j}R_{i,j}(u,v)$$

Where

$\mathbf{P}_{i,j}$ are the control points

$w_{i,j}$ are called the weights,

$N_{i,p}(u)$ normalized B-spline of degree p

$N_{j,q}(v)$ normalized B-Spline of degree q

$R_{i,j}(u,v)$ are the piecewise rational basis function of degree p & q.

The 3D NURBS surface is stored then as

- For the u & v directions :
  - Order of complexity in this direction, i.e., the highest degree of the underlying polynomials + 1.
  - Non-decreasing sequence of knots.
- Array of control points, which positions the NURBS surface in 3-space.

3.3 Rendering of NURBS

Whilst NURBS provide numerous advantages in the creation and maintenance of objects they cause complications when the objects need to be rendered. As the NURBS is defined in parametric space, determining where the NURBS appears in 3-space is non-trivial.

There are two classes of rendering methods for use with NURBS. The simplest is where most of the work is done in parametric space. To determine which points

in parametric space we need to know the corresponding geometric position of, and then use the blending functions to determine that. The other class is where we have a given point in 3-space, and need to determine whether the NURBS exists at that point.

3.3.1 Parametric to Geometric

Whilst the process of determining the position of a point in u, v space is well defined it is difficult, but not impossible, to choose u, v coordinates which give a representative indication of the shape of the surface. The non-uniform nature of the NURBS means that choosing a uniformly spaced pattern of u, v points across the surface may give a very poor impression of its shape.

Consequently static tessellation of the surface causes unpredictable results, which precludes this method except in cases where speed is required. Dynamic tessellation however can be equally problematic, as it is necessary to determine the optimal 'next sample'. Abi-Ezzi *et. al.* [1] developed a technique to generate optimal step sizes for a given section of a surface, which works well provided that surface is well parametrised.

3.3.2 Geometric to Parametric

In rendering methods such as ray tracing it is necessary to determine whether a NURBS passes through a particular point. The inability to move from a point in 3-space, to one in u, v space, means that we must resort to numerical approximations to determine the u, v of the surface.

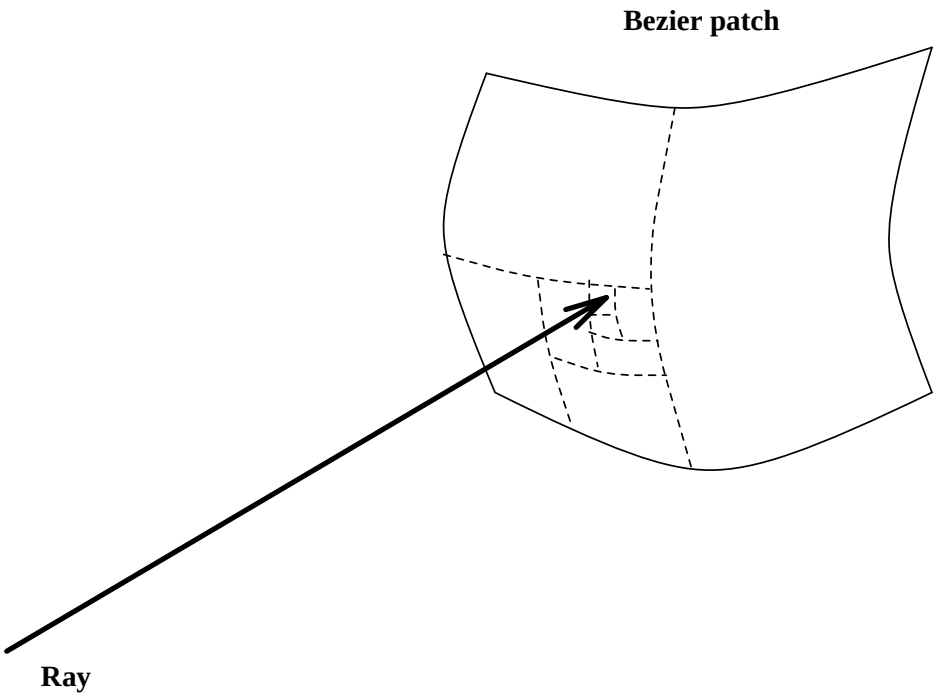


Figure 3.4: Determining the intersection point between a ray and a patch by repeatedly subdividing the patch alternately along the u and v axis

The method used in the ray tracer upon which much of the work in this thesis was based [9] works by splitting the NURBS into a number of Bézier patches [31]. Once this has been done the convex hull of the patches can be used as primitive bounding boxes, to reduce the area which must be tested for intersection. When we know that a ray collides with a particular patch the patch is split recursively until we are testing for intersection against a patch so small that it is irrelevant where on the patch the intersection takes place.

3.4 4D NURBS

As the NURBS surface presents considerable benefits to an animation system, principally simplicity of storage, (as all surfaces can be represented as NURBS), and code re-use, it is necessary to provide some capability to describe a NURBS

surfaces behaviour through time.

It is necessary then to be able to store a 4 dimensional NURBS surface, which describes the behaviour of the 3D surface in a time period t_0 to t_1.

Recall that the 3 dimensional NURBS surface is represented by two parameters, u & v, which describe the shape of the NURBS in parametric space. To represent a 4 dimensional NURBS in geometric space will require 3 parameters, which we will call u,v & t.

To represent a 4 dimensional NURBS we will need to store an order & knots sequence for u,v & t as well as a 3 dimensional array of control points.

A representation of the 4 dimensional NURBS [27] would then be

$$\mathbf{4S}(u,v,t) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{o}\mathbf{P}_{i,j,k}R_{i,j,k}(u,v,t) \tag{3.11}$$

where

$P_{i,j,k}$ are the control points arranged in a 3D array

$R_{i,j,k}(u,v,t)$ are the basis functions.

If the principle storage mechanism for animated entities in a system is to be a 4D NURBS it is worth illustrating how one would be created. The technique to be used is based upon a method of creating conventional 3D NURBS surfaces, skinning.

$Sec1_1$	$Sec1_2$	$Sec1_3$	$Sec1_4$	...	$Sec1_n$
$Sec2_1$	$Sec2_2$	$Sec2_3$	$Sec2_4$	...	$Sec2_n$
$Sec3_1$	$Sec3_2$	$Sec3_3$	$Sec3_4$	...	$Sec3_n$
$Sec4_1$	$Sec4_2$	$Sec4_3$	$Sec4_4$	...	$Sec4_n$
⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1: Creating the control mesh for a NURBS surface

3.4.1 Creating a 4D NURBS

Skinning [23] is a technique used first in the construction of conventional models. A number of rigid wire sections are constructed which approximate the object we wish to model, e.g., an aircrafts' wing. These sections are then 'skinned' by stretching a cover across the frame of the model, to convert a series of 2 dimensional sections into a single 3 dimensional surface.

This method is particular useful in the creation of NURBS surfaces, as it is possible to represent the sections as NURBS curves, so there is a logically clean raising of the dimensionality of the NURBS. We also add the concept of a spine NURBS, onto which the sections are spaced. This spine will provide the v knots & order for the resulting surface.

The algorithm works by ensuring that each of the NURBS sections has a like order & number of control points. This can be ensured by the use of various algorithms developed to raise the degree (elevation), and changing the control points of a curve to fit a new pattern of knots [7, 8].

Once this has been performed on each section it is possible to fill in the resulting 3D NURBS surface using the control points of each curve to provide a line of the surfaces' control point mesh (See table 3.1).

The v knots & order of complexity are copied from the spine NURBS, and the u components can be copied from any of the elevated sections, as they will all be alike.

Note that this approach is not thoroughly general, as it assumes that the number of sections will match the number of knots in the spine curve. Whilst this is insufficient for general use it is enough to illustrate 4 dimensional skinning.

In order to create a 4 dimensional NURBS using this technique we would like to take several 3D NURBS surfaces, and a trajectory (which takes the place of a spine), and produce a 4 dimensional NURBS. From this 4 dimensional NURBS we would like to be able to produce the 3D NURBS at any particular t so we can display the NURBS at that point in time.

As the skinning approach works by converting n 2D sections & a spine into a 3D surface, it is simple to take a 3D surface and decompose it into n 2D sections. A 4 dimensional NURBS would then provide a way of modelling the behaviour of these curves and spines.

Given m sections, each of i by j control points, (which we can ensure by elevation & refinement), the ith curve in the mth section represents the appearance of the ith curve in the m_nth section at a different time.

Given that the behaviour of the nth curve is modelled by m curves, in addition to a spine, which is the trajectory spline, we can use the $2 \rightarrow 3$ skinning algorithm to create a surface. If we perform this operation on all the j curves in each section we will produce j surfaces, plus a spine which describes the v direction of all the original 3D NURBS sections.

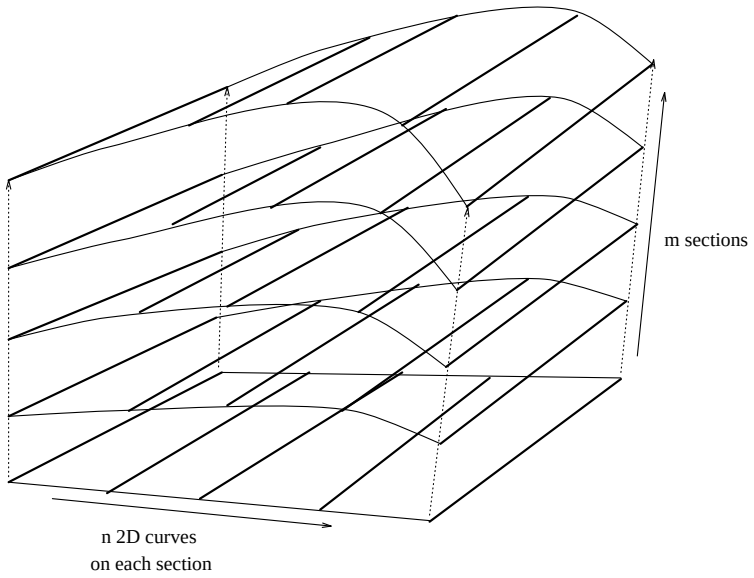


Figure 3.5: Simple in-between

If the resulting control points of each surface are stored in slices of a 3D array, along with the 3D NURBS spine, we have sufficient to hold a 4 dimensional NURBS.

For this format to be useful it is necessary to be able to evaluate the 4D NURBS at a particular t, to find the NURBS to render at that time.

This is performed by evaluating each of the j slices to give a sequence of i curves. Given this list we can perform a $2 \rightarrow$ 3D skinning operation with the 3D spine, in order to find the NURBS surface at t.

So, in summary, a 4 dimensional NURBS we can represent the behaviour of a NURBS surface throughout a given time period.

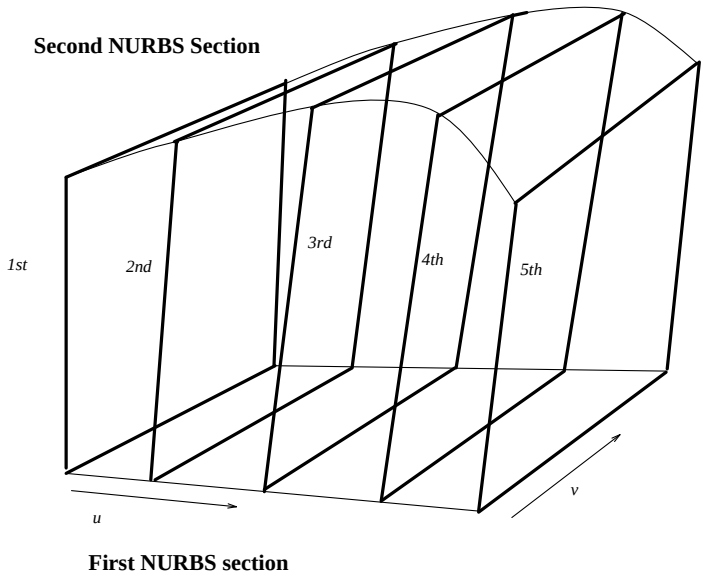


Figure 3.6: The 5 slices which form the 4D NURB

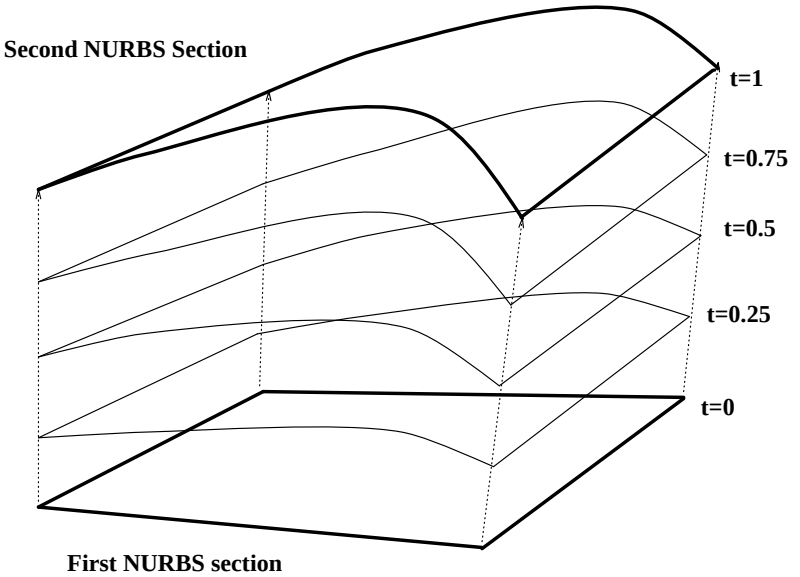


Figure 3.7: Calculating the 3D NURBS at various points in time

Chapter 4

Temporal Rendering issues

4.1 Conventional photographic rendering

As has been previously mentioned the rendering equations used in computer graphics serve as an approximation to the behaviour of a physical camera. To understand the failings of some of the models it is necessary to reiterate some of the principles of a camera's operation.

The camera acts as a container for photographic film. This film has been chemically treated with silver halide to make it sensitive to light falling on it. When light falls on the film the silver halide particles spread dependent on the pattern of light.

The more light which reaches the film, the 'lighter' the colour of the film at that point becomes. The sensitivity of the silver halide means that if too much light falls on the film the pattern created becomes indistinct, and blurred. For this

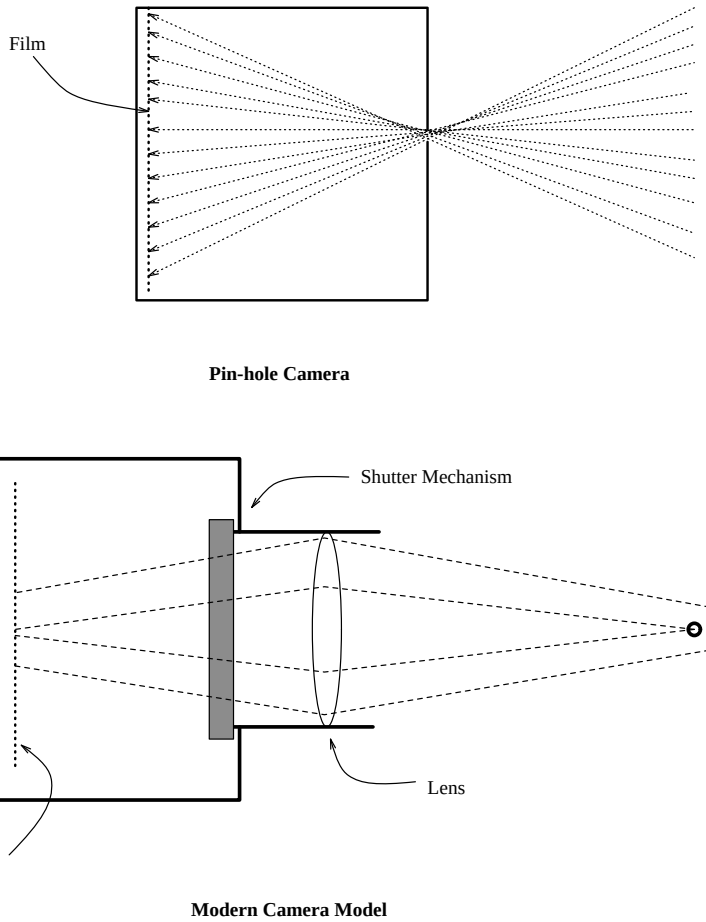


Figure 4.1: Pin hole & Modern camera model

reason the camera needs to ensure that only the right amount of light strikes each portion of the film, in order to keep the image sharp. This *exposure time* is controlled by a shutter mechanism in the camera, which, when an exposure is to be taken, is opened and closed at precisely the correct times to ensure a good quality image.

There are two distinct varieties of shutters.

Slit shutter Here a gap between two metal sheets moves horizontally across the film, exposing a portion of the film at any one time. By ensuring the shutter moves at the right speed the film is equally exposed.

Iris shutter A complicated system of thin leaves irises outwards when the film

is exposed. With this shutter mechanism not all of the film is exposed for exactly the same time.

It is worth emphasising that the shutter remains open for a finite amount of time, in order to allow a sufficient amount of light to strike the film.

The mechanism, as it stands, will produce correctly exposed images, but it will be difficult to ensure the object/scene being photographed will be sharply exposed. To help take sharp images at various distances from the camera, the light passes through a lens system before it reaches the shutter. These lens' focus light coming from a particular distance onto the film, giving a sharp image for objects at that position. The side effect of this is that light reaching the lens from other distances won't be focused. This *depth of field* accounts for the illusion of depth in photographs.

This same mechanism is used in motion picture cameras, with the process being repeated approximately every 25th of a second.

4.1.1 Approximation of camera model in computer graphics

The filming process, which is an integration of light received at each point throughout the exposure time, can be expressed as

$$O^{'}(x^{'}, y^{'}) = \int_{t_{min}}^{t_{max}} f(x, y, t)\ dt\ dx\ dy \tag{4.1}$$

where

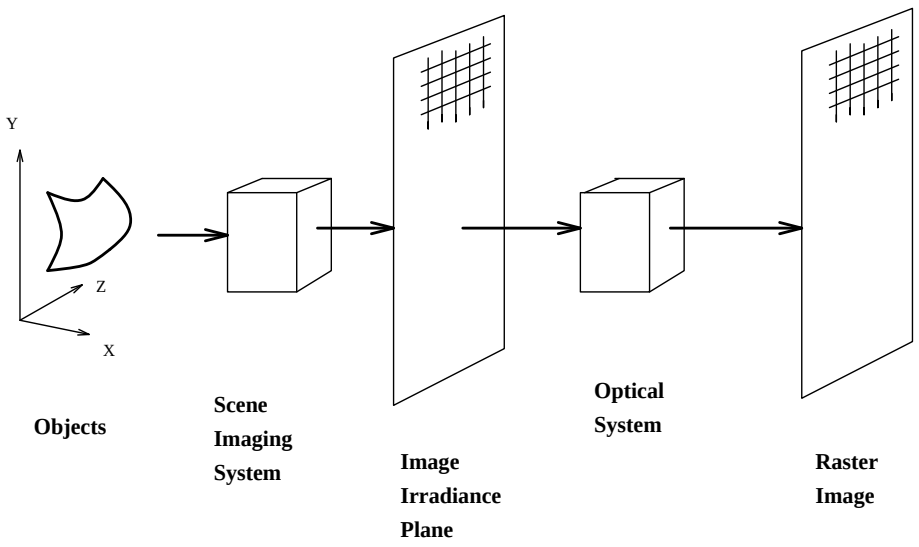


Figure 4.2: Abstract view of the raster image generation process

$f(x, y, t)$ expresses the quantity of light reaching the image irradiance plane

$O^{'}$ is the image irradiance plane (See Figure 4.2)

However this form is normally approximated to a comb function,

$$f(i\Delta x^{'}) = \sum_{i=0}^{M}\sum_{j=0}^{N} \delta(i\Delta x^{'})(j\Delta x^{'}) f(x^{'}, y^{'}) \tag{4.2}$$

where

$\Delta x^{'}$ The distance between samples in the x direction.

$\Delta y^{'}$ The distance between samples in the y direction.

The simplified approximation forms the basis for most of the rendering equations used in computer graphics. However the gross simplifications made adversely affect the quality of the final image. The principle problems are

- The instantaneous exposure time creates images which give no visual clues as to the behaviour of the scene during the exposure. For example an

image showing a football suspended above a field gives no clues as to the balls direction of motion, or indeed if it is stationary. This exposure is often perfectly acceptable for stationary photographs, but when the images form a part of an animated film the discretisation of the continuous image causes unacceptable temporal aliasing artefacts, seen by the human eye as strobing.

- The lack of any expression to account for the focusing of the image causes problems when rendering a scene with a distinct foreground & background cluster of objects. The lack of depth clues for the viewer make distinguishing the composition difficult.

- All the film is exposed for a constant amount of time, thereby not accounting for the effects of shutter motion. Modern cameras have developed complicated mechanisms to minimise the effects of shutter motion in real cameras, so its absence in computer imagery is of minimal importance.

- Real lens suffer from imperfections as a result of the production process which degrade the quality of the image produced in a real camera. However the computer model of a camera makes no allowance for aberrations in the lens, something which precludes production of images by computer which appear identical to those filmed on 'real' cameras.

The absence of a model to account for the finite exposure times & lens effects result in images which are inferior to the quality of image produced on a camera. In order to produce computer animation of an equivalent standard to that obtained by conventional filming it is necessary to modify the computer rendering

model.

Whilst all of these faults are serious the human eye is particularly receptive to patterns in anything it sees, and so human viewers of computer animation find strobing, and indeed any form of aliasing, particularly unpleasant. All animation relies on the human eyes inability to sample at a particularly high rate, and this is used to provide the illusion of smooth motion where in fact there are only a finite number of discrete images.

If this process breaks down, or is accentuated by the strobing caused by temporal aliasing, the illusion of motion is shattered. Therefore it is necessary to address the issue of exposure lengths in order to create high-quality photo-realistic animation.

4.2 Motion Blur

Various techniques have been developed to address the issue of temporal aliasing, or *motion blur*, and the unrealistic exposure length problem in general.

4.2.1 Lining

Traditional cell animators are under less pressure to produce photo-realistic images, though they still feel the need to address the problem of strobing images, largely as cell animation gives minimal visual clues to the human eye to ensure the film is seen as continuous motion.

They [6] tackle the problem by drawing lines radiating from the moving object backwards in the direction of motion. This simplistic approach confuses the area behind the moving object in an attempt to ameliorate the motion flaws.

4.2.2 Multiple Instantaneous exposures

As has been previously mentioned computer graphics relies on its ability to sample a continuous function (the image) in discrete chunks which are amenable to manipulation on a computer. This motion blur technique extends that philosophy into discretisation of temporal effects.

Spatial aliasing problems are often minimised by increasing the resolution of the sampling. In a conventional computer animated film the product consists of a series of static images, which are designed to presented to the viewer at a given *frame rate*. This speed is sufficient for the eye to be unable to resolve the film into discrete images, thereby maintaining the illusion of motion. This means that the frame rate directly determines the number of samples made of the scene per second of 'film time'.

By increasing this rate, but not changing the frame rate, we are *super-sampling* the scene through time. As in spatial super-sampling it is necessary to find some method of filtering these multiple samples into a single sample which will form the output of the process. In animation terms this consists of resolving the multiple discrete images into a single frame which will form part of the film.

Recall that the filming process is an integration of light energy throughout the exposure time. This means that the quantity of light reaching the physical film

from a given object should remain relatively constant whether that object is moving or not. In practice then an area of film which represents an object in motion will have received the same amount of light as an area which would have represented the object at rest, but spread over a larger film surface area. This means that film exhibiting motion blur will appear dimmer than they would otherwise be.

As part of the final stage of resolving the multiple images into a single image it is necessary to ensure that the resulting object will appear faded to a sufficient degree so as not to appear to glow when in motion.

One technique for creating the final image, which handles this energy dispersion, is *alpha-blending* [24, 12]. Here for each pixel rendered in each of the n images we produce

$$P_{x,y} = \begin{bmatrix} ar & ag & ab & a \end{bmatrix}$$

where the a, or *alpha* value, represents the coverage of the area represented by that pixel, by the colour rgb. Note that the colour triplet is pre-multiplied by the alpha value to simplify the composition process.

Once we have the discrete images we need to composite them into a single image. First it is necessary to 'dim' each of the n images to account for the spread of light energy. This is achieved by reducing the alpha value for each pixel in each image. Consequently

$$P(x,y) = \begin{bmatrix} ar & ag & ab & \frac{a}{n} \end{bmatrix}$$

Once this has been performed we combine the samples by compositing the n images onto the background of the output image. Therefore for each pixel we

repeat the following operation n times for each input frame.

$$O^{'}(x^{'}, y^{'}) = P(x, y)_a + O(x, y)(1 - P(x, y)_a.\alpha) \tag{4.3}$$

where $P(x, y)_a.\alpha$ is the alpha channel of the ath input image at x, y.

This super-sampling process helps to alleviate some of the effects of strobing, but suffers from two serious flaws.

Firstly the human eye, as has been previously mentioned, is sensitive to patterns. If we are super-sampling a thin object moving in the direction of its thinnest axis then we will see holes appearing between the samples, where the background image will be visible. To make things worse there will be a pattern of holes, which the eye will be naturally drawn to. This effect will also be seen where either the object is moving too fast for the degree of super-sampling being used (remember that super-sampling is only an approximation, and it will always be possible to find a speed that won't be accounted for), and the image becomes a series of disconnected objects. Consequently the projected shape of the object, and its speed & direction of motion determine the quality of animation produced by this method.

Some techniques to reduce the effect of this have been developed, notably a final stage after composition where the moving objects are box filtered to average out the intensity of the image across its area still further, thereby reducing the sharp edged holes which the eye finds so easily.

Secondly, as a consequence of the first fault, it is difficult to determine the degree of super-sampling required to produce a film of given quality. This inability to adaptively super-sample as would be used in spatial anti-aliasing means that the

system is forced to super-sample at a constant rate throughout the film. This is often extremely expensive, and so reduces the benefits of the technique.

However it is worth pointing out that this technique will work when rendering objects in a variety of ways, through both local & global illumination models.

4.2.3 Post-process motion blur

This method of modelling motion blur, and the primitive camera model, relies on a series of post processing steps to 'add' the effects of a real camera to the conventional image produced using traditional rendering models. The principle advantage to this is that little or no change needs to be made to existing systems to add the temporal effects.

Max & Lerner [20] proposed a simple '$2\frac{1}{2}$D' algorithm where the scene is split into its component objects and each is treated in isolation. The Blur algorithm begins with a rendered raster image of the object at rest. They then calculate the direction on motion of the object ($\frac{dx}{dy}$) as the object is projected onto the viewing plane.

In order to perform a raster based blur they skew the image so that the blur can be applied in the direction of the largest blur component vector (dx or dy). Suppose for example that the vector of motion is $\frac{3}{1}$, then we skew so that we can perform a blur in the x direction. To this end we skew each column downwards by $x \times \frac{dy}{dx}$, which is $\frac{x}{3}$. Note that the resulting raster image consists of weighted sums of adjacent pixels in the instantaneous raster, as we often have to account for non-integral shifts.

Now the skewed raster is blurred by the number of pixels corresponding to the magnitude of the largest blur vector component, in this case x. Different varieties of blurring can account for different effects, though to compensate for strobing we need only blur with equal intensity through the exposure time. To this end we need only compute the intensity at a pixel by adding it to the previous dx pixels in that row, and dividing by $dx + 1$.

Alternative blur functions can account for greater intensity towards the end of an exposure to indicate motion in a static frame. Note that whilst this appears to give the correct result for a static image it will cause aliasing effects if used as part of a film.

Finally the reverse of the skewing process is applied to skew the blurred image back into position, to give a blurred raster image.

This approach, whilst simple to implement, has two serious flaws.

- It does not account for non-linear motion, which will cause problems for all but the most elementary films.
- If the object we are blurring has a highlight on it due to its position with respect to a light source, and the object is moving quickly, then the resulting image will have an incorrect blurred highlight.

An alternative approach developed by Potmesil & Chakravty [26, 25] accounts in a more mathematically rigorous fashion for all the temporal effects exhibited by a camera. They abstract the filming process to be a serious of steps with defined inputs and outputs. The process consists of a function which transforms object

definitions from the a homogeneous coordinate system to an *image irradiance place* (See figure 4.2), which accumulates the radiant intensity for each visible surface point. This function is normally a comb function, (Equation 4.2), such as the rendering models outlined earlier.

The transformation from the image irradiance plane $O^{'}(x^{'}, y^{'})$, to the *image output frame*, or raster image, $O^{''}(x^{''}, y^{''})$, incorporates the temporal effects of a camera, such as the degradation of the image, and in doing so simulates the properties of the film. This transformation can be denoted by

$$g(x^{''}, y^{''}) = h(x^{''}, y^{''}; x^{'}, y^{'}, f(x^{'}, y^{'})) \tag{4.4}$$

where $h(x^{''}, y^{''}; x^{'}, y^{'}, f(x^{'}, y^{'}))$ is the optical system transfer function, or *point spread function, (PSF).*

Two assumption are made

- The radiant energy distribution in both the image radiance, and image output plane are either positive or zero. No energy is lost.
- The radiant energy intensity function is additive in both planes, which helps account for the integration properties of film.

Therefore the full transformation from the image-irradiance (film) plane can be expressed as

$$g(x^{''}, y^{''}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x^{''}, y^{''}; x^{'}, y^{'}, f(x^{'}, y^{'}))dx^{'}\ dy^{'} \tag{4.5}$$

Suppose that the image recorded over an exposure time t can be represented as

$$g(x^{''}, y^{''}) = \int_0^T g(x^{''}, y^{''}, t)dt \tag{4.6}$$

The full imaging function over the range t is

$$g(x^{''}, y^{''}) = \int_0^T f(x^{'}(t), y^{'}(t))dt \tag{4.7}$$

Given this basis Potmesil & Chakravty proposed a system to provide motion blur. The object space to image irradiance plane function, Equation 4.4, is performed by a recursive ray tracer. However instead of simply providing radiant intensity values as a consequence of evaluating the recursive tree (see section 2.3.1), the system requires, for each node of the shading tree at $O^{'}(i\Delta x^{'}, j\Delta y^{'})$:

1. The type of intensity information at that node: opaque, reflected, transmitted, hidden or shadow.
2. The i, j coordinates of the sample point.
3. The intensity values at each of the wavelengths under consideration, $I_n\lambda$.
4. If the node isn't a shadow node we require $z^{'}$ depth along the camera optical axis, of the measured point.
5. The object identification number of either the measured point, or the object casting the shadow.

This information is split into several raster images with the the points in a particular frame **f** sharing the same *projected* direction of motion $r^{'}(t)$. Each of these images is passed to the motion blur processor, along with the direction of motion

$r^{'}(t)$, the exposure time t_{frame}, and the exposure length, T_{frame}, with which the motion blur PSF, $h(x^{''}, y^{''}; x^{'} y^{'})$ is computed.

The motion blur processor computes the blur exhibited by the pixel in **f** using the PSF, placing the result in the raster image **h**. The two images **f** & **h** are then convolved into a blurred image.

After the motion blur PSF has been applied it is possible to apply a shutter PSF to approximate the effects of shutter motion. Shack [28] derived the functions for the two common shutter varieties.

In addition [25] outlines a PSF for handling the effects of the lens on the image produced. This process takes the image-irradiance plane as its input, and needs the z value in the cameras optical axis for each sample. The PSF then projects the sample onto the image output frame as a *circle of confusion* whose size depends on the difference between the camera focal point, and the sample point distance, i.e., the further away from the focal distance the point is the larger the area that energy is spread across, and so is more blurred & indistinct.

Finally the resulting blurred frames are composited with the images of the stationary objects, using alpha blending, to produce the blurred frame.

4.2.4 Distributed ray tracing

The Potmesil & Chakravty method correctly copes with many of the problems of motion blur in a computer generated film. However the approach of separating the motion blurring & depth of field calculations from the initial rendering stage

causes a number of difficulties.

When an object remains completely behind a foreground object, then for a portion of the exposure time it may be that the background object becomes visible due to the motion of the foreground object. Any post-processing solution will have difficulty dealing with this situation elegantly.

The fundamental problem of splitting the scene into background & foreground, so that they can be rendered separately, becomes a problem, as it is difficult to devise a general algorithm. For example if two objects were to move in unison, with one totally occluding the other, the blur of the motion should not show the farthest object. However the post-processing solutions would be forced to class the farthest object as part of the background, and so would be visible.

These problems, along with the not insignificant cost of motion blurring objects with non-trivial motion paths, make post-processing in general a problematic solution.

Distributed ray tracing [10] is in itself a brute force solution to the fundamental problems of modelling an integration process by discretisation. Conventional ray tracing distributes the rays in two dimensions, the u & v of the image plane. These discrete samples in 2 dimensions approximate a 2 dimensional continuous image. Consequently if we wish to render a scene in more dimensions, we can distribute the rays through the extra dimensions in the same manner we do for the 2D image.

In practical terms this means, for motion blur, that each ray exists at a particular time t, as well as a position in 3-space. We are approximating the integration

properties of film by super-sampling in t on a ray by ray basis.

The principle advantage of this is that the algorithm to perform recursive ray tracing is a relatively simple & elegant solution, which will not need significantly changing to ray trace through time as well as space. Therefore all the optimisations for ray tracing can be equally well applied to temporal rendering.

Distributed ray tracing, i.e., the practice of distributing *individual* rays in additional dimensions can be used to approximate the other deficiencies of the camera model.

- Depth of field can be approximated by distributing the rays across the lens rather than a flat image plane in 3-space. As the lens will be non-flat the rays leaving the lens will not all travel in a straight direction, i.e., the lens bends the light ray. This approach handles the problems inherent in the Potmesil & Chakravty process, where no allowance is made for the lens changing the direction of the rays, and therefore making certain things visible which weren't with a flat plane.

- A glossy surface can be approximated by distributing the reflection angle around the perfect reflection angle (See figure 2.2).

- Penumbras account for the soft edges around shadows, where a light source is partially obscured. Penumbras can be approximated by distributing the shadow rays towards different positions on the light sources. Special effort must be made to accommodate different intensities on the light source itself so the resulting shadow blurs correctly.

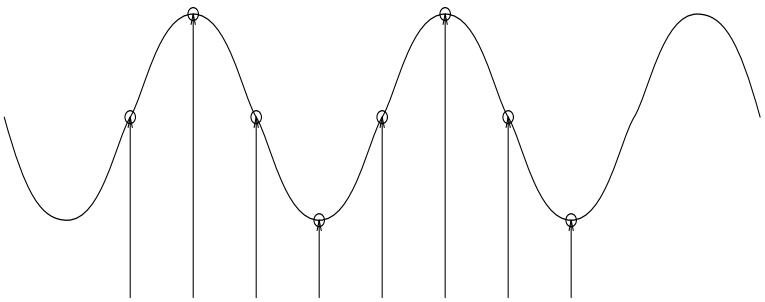
In order to generate pictures of acceptable quality it is necessary to super-sample the images, and then apply a filter to generate the correct image. Whilst such a traditional technique would be acceptable it would not totally remove aliasing artefacts from the resulting image, merely reduce it. In a technique such as distributed ray tracing the increased dimensionality accentuates the problems of aliasing.

To help ameliorate the worst effects of aliasing *stochastic sampling* is used to replace aliasing artefacts with noise. As the human eye is accustomed to seeing noise, then the resulting image should be more acceptable.

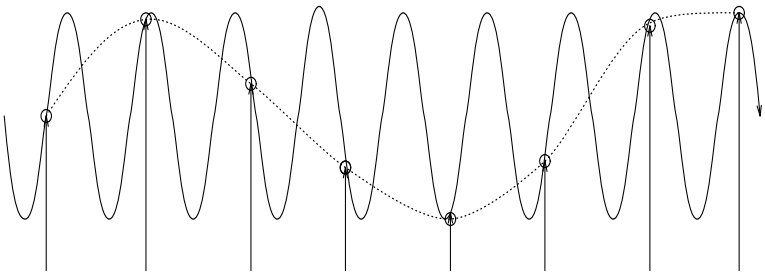
Stochastic sampling theory

Ray tracing is normally a uniform discrete point sampling process, and like all such processes, sampling frequencies greater than a certain limit will cause the image to appear to be a representation of a lower frequency.

This is easiest illustrated when sampling a 1 dimensional signal $f(t)$ (See figure 4.3) where sampling is performed at regular intervals, so the nth sample takes place at nT. The sampling frequency is said to be $\frac{1}{T}$. If the signal we are sampling oscillates at a frequency, and we are sampling at a sufficient frequency then we will see a reasonable approximation of the signal. However there reaches a point at which if we sample below that frequency then the signal we see will give a misleading view of the real signal. For example if we are sampling a sine wave we will not notice some of the peaks, and the resulting image will appear to represent a different signal.



Sampling within Nyquist limit



Sampling outside the Nyquist limit

Figure 4.3: Sampling $f(t)$ above and below the Nyquist limit. The arrows indicate the sampling points, and the circles the sampled value. In the lower sampling the sampled values incorrectly represent a lower frequency sine wave.

This limit is known as the *Nyquist* limit, and is half the signal frequency. If T falls below this limit then we will see an incorrectly approximated signal.

The definition of a correct sampling states that sampling a signal, followed by an ideal reconstruction filter will produce the original signal. If we are sampling a 2 dimensional pattern then the resultant image will have incorrect patterns embedded in it, which are indistinguishable from the original image. Consequently an image with sampling errors in it cannot be reconstructed to produce the original image.

The inability to produce signals when sampling below the Nyquist limit is inherent in the sampling process, but aliasing only occurs when the samples are uniformly spaced, i.e., when we see a pattern which isn't there. If we use non-uniform sampling then we produce *noise*, which differs from aliasing in that it has no distinct frequency components, and so no false patterns.

The human eye works by sampling the image received in the eye by the use of photoreceptors, which collect the light, and pass signals to the brains for interpretation. There are a finite number of photoreceptors in the eye, and so sampling theory should indicate that the human eye should see sampling errors in the image it sees when presented with certain patterns. However our eyes alleviate these errors by two techniques.

Directly opposite the lens in the eye is an area known as the *fovea* where the nerves do not obscure the photoreceptors, and so most of the light reaching this area is sampled by the photoreceptors. This area, which is the portion of the eye measuring the place the eye is looking towards has a densely packed hexagonal

pattern of photoreceptors. The density, and the pattern mean that, together with the lens' ability to act as a low pass filter, all sampling errors are avoided.

Outside the fovea the photoreceptors are more sparsely spread, and are covered by the nerves, meaning that the image is less thoroughly sampled, and so you would expect to find aliasing.

In fact the human eye avoids aliasing in the extrafoveal area of the eye by distributing the photoreceptors in a pattern known as the *Poisson disk distribution* [18]. This spread is such that there is no distinguishable pattern, and there is no bunching of photoreceptor cones, so an even sampling spread is assured.

This distribution replaces aliasing for noise. However as the noisy area of the eye is not the centre of attention for the brain the effects of any imperfections are not noticeable. Noise is reduced by ensuring that the cones are placed at a minimum distance from their nearest neighbour, so that the image will still be acceptably sampled.

As the human eye is accustomed to receiving noise it is advantageous to swap aliasing for noise in computer graphics. In theory this could be performed by sampling the image using a Poisson disk distribution, however the disk distribution, which is easiest visualised by putting a wooden disk round the sampling point which prevents any other sample from getting too close, is expensive to calculate. One possible implementation would be to repetitively pick random positions on the lens, and then discard the point if it were too close to any other previous sample. Obviously this technique is extremely expensive, and therefore ill-suited to implementation on a computer.

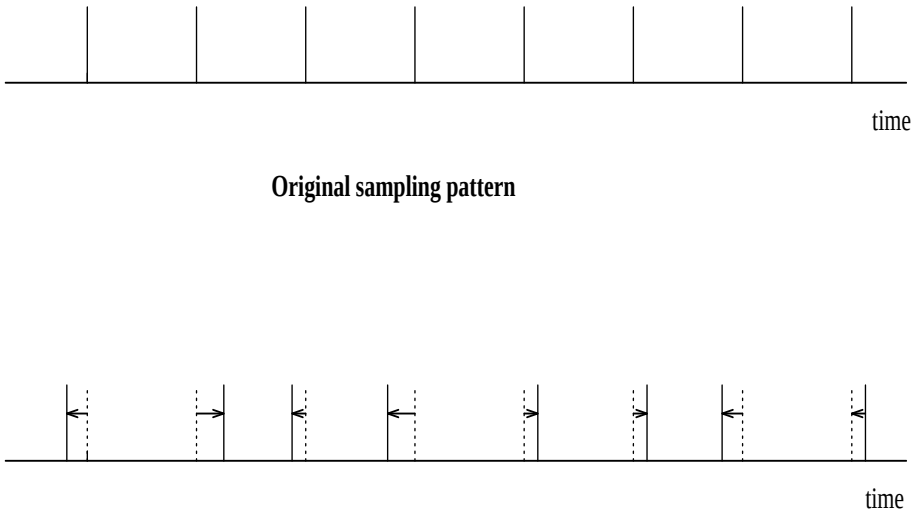


Figure 4.4: Jittering a time sampling pattern

A more cost effective approximation is to begin with a regular sampling pattern, to which an uncorrelated jitter is applied to each position. For example if we are sampling through time, we would normally have samples at regular gaps, such that the nth sample was at nT. If we jitter the sample by ζ_n then it occurs at $nT + \zeta_n$. If the jitters are uncorrelated the jittering has the following effects.

- High frequencies are attenuated.
- The energy lost due to this reduction in signal intensity appears as uniform noise. The intensity of the noise is the same as the intensity of the attenuated part of the signal.
- The basic composition of the spectrum remains unchanged

Frequency attenuation would not normally occur through a sampling process, however if the jittered sampled signal is passed to an ideal reconstruction filter it would.

If the signal we are sampling has a whole number of cycles within the range of possible jitters which can be applied to a sampling point then all of the cycle has an equal possibility of being sampled, and so the signal is reduced to pure noise. However if there isn't an integral number of cycles in the possible jitter range then certain areas of the cycle are more likely to be sampled than others. In this case some attenuated aliasing will occur.

Using jittered sampling to approximate the behaviour of full Poisson disk stochastic sampling provides many of the benefits, but little of the cost. Sampling in 2 dimensions is relatively straightforward. Before ray tracing the ray in question is jittered by a certain amount (in the u & v dimension of the image plane), and then ray traced as normal. Some degree of super-sampling is used, so when all the rays for a given pixel have been jittered then traced, it is necessary to reconstruct the image at that point. The simplest filter is simply to average the resultant intensities at each wavelength, though weighted reconstruction filters can give better variance reduction.

To jitter in non-spatial dimensions, such as time to model motion blur, we need to provide a way of distributing the same points evenly throughout the sampling range. This can be performed by dividing the time into slices, and then randomly assigning the slice to a particular ray [10]. The exact time at which the ray exists is determined by jittering. For example we may wish to shoot 16 rays per pixel, in a 4×4 pattern.

We devise a prototype pattern which distribute the time slices as evenly as possible around the grid, to ensure that one side of the pixel will not be weighted towards one end of the exposure time. A sample prototype time pattern is shown

7	11	3	14
4	15	13	9
16	1	8	12
6	10	5	2

Table 4.1: A prototype time pattern for a 4×4 grid of sample points

in table 4.1.

The sample in the xth column in the y row would have a prototype time of

$$t_{xy} = \frac{P_{xy} - 0.5}{16} + t_0 \tag{4.8}$$

where P_{xy} is the value at x, y in the prototype time pattern, and t_0 is the time at which the exposure begins. A random jitter of $\pm\frac{1}{32}$ is added to t_{xy} to obtain the actual time at which the ray exists.

For example if we are firing the ray in the upper right corner of the pixel then the ray can exist in the exposure range $\frac{26}{32} \leq t \leq \frac{30}{32}$

Once we know the time at which a ray exists it is necessary to change the scene store to represent the state at that time. This is normally the job of any animation system, which needs to do all the temporal calculations for every ray.

Optimisation

As it stands distributed ray tracing consist of, for each pixel in the image, subdividing the pixel into 16 rays, which are stochastically jittered from a grid pattern to limit spatial aliasing, and are then distributed through time to determine their temporal position. Then, for the current ray the entire scene is updated to show the positions at that time, and finally the ray is shot.

Whilst this process is simple, and relatively elegant, it can quickly become extremely costly, as much of the work will be performed for no benefit. The problems can be split into two categories.

- Even if the ray will miss all the objects it is necessary to update all the scene.
- We are always supersampling by a fixed degree, this means considerable effort is expended even if the resulting intensity differs little from the result produced by a single sample.

In conventional computer graphics the high intersection costs are reduced by the use of bounding box hierarchies to simplify the work that has to be done for each ray, i.e., to determine as quickly as possible if the ray could not intersect anything in the scene. In order to reduce the workload of ray tracing in 4D, which is what we are doing with motion blur we would like to first intersect against 4 dimensional bounding boxes. Only if we know that the ray could intersect an object at that space time will we need to perform more exact intersection tests.

Glassner [14] developed a technique for space-time ray tracing, where the ray itself is intersected against bounding volumes. The ray and box each have 4D points, and so the intersection tests simply have to test for intersection in an additional dimension.

The mathematics is simplified somewhat by the constant time of the ray, i.e., the ray does not change time along its length. The ray can then be defined as

$$(x, y, z, t) + \lambda(x, y, z)$$

The resulting tests can be visualised easiest by considering the problem in 3D, where 2 dimensional objects are passing through the time slice. The behaviour of the 2D object through time defines a volume in 3D, where in this case the 3rd dimension is time. A space-time intersection test in this situation is the same as a conventional ray/object test.

Glassner outlines a bounding box calculation algorithm which can be applied in 4D space-time, thereby reducing the amount of work which needs to be performed for each ray if the ray misses all the objects. By using bounding boxes each of the rays used in distributed ray tracing can be checked against the objects in the scene, possibly removing the need to accurately determine the exact scene composition at that time.

Like any super-sampling process there remains the problem of determining the degree of super-sampling to apply to a given frame, and some work has been done on the use of adaptive stochastic super-sampling [11, 21, 19], to concentrate the rays at the area of most interest, primarily as a means of reducing the amount of work which would otherwise be expended stochastically super-sampling areas of little intensity variance.

Chapter 5

Motion Blur of NURBS

The techniques used to approximate motion blur, outlined in the previous chapter, were all developed with the simpler polygonal surface representations in mind, and a non-integrated approach to animation. This means that the 4 dimensional information (the animation) is stored implicitly as a combination of static states, and motion rules (whether those rules were created using key-framing or dynamics).

A 4 dimensional storage format provides an integrated approach to animation, which encourages use of techniques such as motion blur. As the methods used to render parametric surfaces are substantially different from the ones used in polygonal based rendering, the motion blur techniques exhibit certain failings when used with NURBS.

Each of the 3 principle solutions to the temporal aliasing problem is considered in turn.

5.1 Multiple Exposures

As has been previously noted the technique of generating multiple exposures, each of which represents the scene when viewed with an instantaneous exposure time, is intended to cause the *animation* to appear to be blurred rather than the individual frames. By compositing the images the frame is seen as sufficiently confused for the eye to register a blur.

The principle fault with multiple exposures is that this simple technique does not totally remove the possibility of a pattern developing in the images of the film. As the human eye is sensitive to aliasing it is quite possible that multiple exposures taken at the wrong time interval will produce gaps between the objects in motion, thereby causing temporal aliasing as unpleasant as would have been seen without any extra work.

In order to ensure no aliasing occurs with this method it is necessary to ensure that a sufficient number of exposures are generated on an object by object basis. Determining the number of exposures is a difficult problem to solve even when using objects moving along a linear trajectory, as the possibility of a pattern depends on both the surface area taken up by an object on the screen, and its projected shape.

When using objects travelling on a non-linear trajectory, such as a parametric curve, the calculation of where the object will be at a moment in time becomes expensive. Consequently it becomes very difficult to determine, on an adaptive basis, how many exposures are needed in order to completely remove the possibility of aliasing occurring.

The only solution then would be to perform a static number of exposures per output frame, with the exposure rate set sufficiently high to preclude the occurrence of temporal aliasing.

The cost of performing motion blur using multiple exposures is heavily dependent on the speed at which a static rendering can take place. If the static, conventional, images take a long time to generate, then motion blurred frames can quickly become very expensive.

As has been previously noted the photo-realistic rendering of NURBS is extremely expensive without hardware assistance, and so generation of motion blurred films containing NURBS, using a technique based on multiple exposures would quickly become too costly to be worthwhile.

5.2 Post-process motion blur

The Max & Lerner [20] approach to post-process motion blur relies on the ability to decompose the projected path of motion to a linear direction for motion blur. As this technique is based on blurring a raster image generated with an infinitely small exposure time it should be perfectly applicable to NURBS.

However one of the key benefits for the use of NURBS in animation is their ability to represent complicated, and non-uniform, trajectories, which cannot be temporally anti-aliased by a linear raster blur. Consequently a straight linear raster stretch would be insufficient to handle many of the situations.

The other principle approach [26, 25] addresses the issue of non-linear motion,

which helps make it a useful technique. The direction of motion function $r'(t)$ can be represented as a NURBS curve, and so there is no reason for the motion blur PSF to have any difficulty in rendering objects on a non-linear, and non-uniform trajectory.

In addition to NURBS providing the ability to model complicated trajectories, they will also be used to model the behaviour of objects through 4D, so during time the object will undergo deformations. For example one possible, albeit extreme, case is where a sphere changes shape throughout a series of exposures to resemble a human face.

As the NURBS changes, parts of the surface will become visible which weren't at the start of the exposure time. Post-process motion blur relies on the ability to blur what is visible at the start of the exposure time, and in many cases of animating NURBS this will simply not be sufficient. This problem occurs when animating any object type, but the ability of NURBS to model complicated deformations means that it is likely to occur more often.

A similar problem lies in the non-uniformity of motion across the surface. When animating an object, it is quite likely that one part of the object will move in a different direction, or at a different rate, to the rest of the object. When using NURBS this is extremely likely to occur, as parts of the surface may stretch, whilst some of the surface will not move. In order to apply a post-process blur to this picture it is necessary to apply the blur PSF to all of the scene.

Potmesil & Chakravty address this problem by recommending that the scene is decomposed into multiple composite maps, where the contents of one map

is all moving along one trajectory, and can therefore be treated in isolation. Consequently we decompose the image irradiance plane into multiple planes, $O'^1 \ldots O'^n$, each of which has its own point spread function $g^i(x'', y'')$. The relevant PSF is applied to each plane, and then the final stage of the blur process consists of re-compositing all these maps together.

In order to blur NURBS in this way it would be necessary to determine the trajectories of each point on the surface in isolation, and treat each of them separately. Whilst this isn't impossible, it is extremely difficult to do in such a way as to guarantee an accurate result. For example, if we have intersected a ray against a NURBS to determine the u, v intersection point, we need now to determine the x, y, z of that point at the end of the exposure time by using the 4D NURBS to determine the NURBS at the end of the exposure time. However this point may become invisible during the exposure time, and could therefore not be blurred by a simple PSF.

Quite apart from the extreme cost of calculating the next position, the approximation of the motion of a NURBS surface to several distinct image irradiance planes would result in the blur containing holes, through which the background would be visible, as the approximation of continuous motion would result in inaccuracies. If this problem is combined with the inability to blur a NURBS from a static raster image all the time, then the Potmesil & Chakravty technique will provide inconsistent and poorly blurred images.

Post-process motion blur, whilst theoretically applicable to NURBS, has difficulties blurring many of the things NURBS surfaces are particularly good at, and so will be used for. Consequently neither of the two principle post-process

techniques are usable in the temporal anti-aliasing of NURBS.

5.3 Distributed ray tracing

Distributed ray tracing is capable of motion blurring anything which can be rendered normally using recursive ray tracing. As has been previously mentioned NURBS are ray traced by recursive sub-division of the component Bézier patches [31] until the patch is sufficiently small.

Distributed ray tracing exhibits none of the problems associated with multiple exposures or post-process motion blur, as time dependent sampling is used as a way of approximating the integration properties of the film.

In order to render the NURBS efficiently a bounding box hierarchy is created, with the leaves of the hierarchy the bounds of individual Bézier patches. As a result of this, the process of motion blurring a NURBS would then consist of, for every ray in the scene;

- In-betweening the 4D NURBS to generate the 3D NURBS at time t.
- Splitting the NURBS into Bézier patches.
- Generating a bounding box hierarchy
- Intersecting the ray as normal
- Freeing any space taken by Bézier patches & Bounding volume hierarchy.

This simple solution will handle all the possible deformations a NURBS could

undergo during the exposure time correctly, and produce motion blurred frames of a consistent quality.

Whilst in theory distributed ray tracing would then be the ideal choice for motion blurring NURBS there is a problem. We are super-sampling in t through the exposure period, and this means that for every ray we need to cast we need to go through the in-betweening, split & generation processes. Not surprisingly this quickly becomes extremely expensive. When rendering non-parametric surfaces all but the last 2 steps would entail little expense, but the parametric nature of the NURBS means that naive distributed ray tracing becomes very expensive.

Distributed ray tracing provides the ability to temporally anti-alias frames containing NURBS, but without optimisations the technique would be so costly as to be prohibitive.

5.4 Optimisation

Looking at the series of steps the motion blurring renderer must undertake for each ray it is easy to split the job into 2 distinct stages.

- Generate scene at t.
- Trace ray

If we assume that the actual process of tracing a ray cannot be optimised then the principle area for improvement is in the time taken to generate the scene.

The generation stage, as has been previously noted, consists of in-betweening to generate the 3D NURBS, conversion to Bézier patches, and generation of the bounding volume hierarchy. Each of these steps is comparatively expensive and should be avoided if at all possible.

Once the scene has been generated the first step of ray tracing is to intersect the ray against the entire bounding volume. If the ray misses then we record a ray with the background colour, and move on. Therefore, in the situation where the ray misses an object, which is likely to happen in the average scene, then all the effort expended to generate the scene has been wasted.

By splitting the generation stage into steps, and only performing all of them when we are reasonably sure it will help, we should be able to drastically reduce the time taken in some instances.

If we are to test whether a ray may intersect an object we traditionally use bounding volumes, which are quicker to perform intersection tests against than the scene contents. As we cannot use the Bézier bounding volumes, (as the expense of generating these is the problem in the first place), we need to find some bounding volume which provides a reasonable fit, at minimal intersection cost.

Good bounding volumes, whose function is to approximate the object they enclose in a way which encourages efficient intersection tests [2], are extremely expensive to generate for parametric surfaces. Problems encountered in generating a polygonal approximation to a NURBS apply also to the generation of bounding volumes (See section 3.3). It is necessary to choose a bounding box generation

algorithm which generates the best volume in the minimum time. Weghorst *et al.* [32] noted that if we are using bounding volumes in a scene, we can quantify the costs associated using the formula

$$T = b * B + i * I \tag{5.1}$$

where

T is the total cost function.

b is the number of times that the bounding volume is tested for intersection.

B is the cost of testing the bounding volume for intersection.

i is the number of times that the object in question will need to be tested against for intersection in production of the scene.

I is the cost of testing the actual item for intersection.

When ray-tracing NURBS, I is especially high, and so bounding volumes, in order to give the best benefit, must attempt to minimize the number of times that the NURBS must actually be intersected against. In practice this means that the tighter the fit of the bounding volume around the NURBS, the better the effect it is likely to have on performance, even at the cost of increasing B.

These motion blur algorithms will be used to motion blur 4D NURBS generated with the technique outlined in section 3.4, which are created by raising the component 3D NURBS to a common higher order of complexity. This means, that on average, the NURBS surfaces generated from 4D NURBS will have high orders of complexity.

When a NURBS surface is represented there is a minimal order version of that surface, which means that the geometric shape cannot be represented with a NURBS of lower order. However it is always possible to represent that surface using higher order polynomials. NURBS using these higher order curves can exhibit control point meshes with greater numbers of points, needed to ensure the control over that portion of the surface. The side effect of this is that the control polygon volume of high order NURBS will become closer to an ideal bounding volume as the order increases.

This means that in practice we can hope to achieve a moderately good bounding volume for an in-betweening NURBS by simply using the control point volume. Whilst this has its advantages control volumes can be concave, which reduces the speed at which intersection tests can be made. However the principle reason for using bounding volumes is to save time which would otherwise be required to intersect against the actual NURBS. If we have to use un-optimised intersection tests against each facet of the control volume, instead of optimised axis aligned box tests, we will still gain a considerable benefit, as the more complicated the volume, the higher the order of the enclosed NURBS, which would be even more expensive to test against.

This is difficult to prove empirically, and so it would be advantageous to test it against alternative strategies.

The question also remains as to whether it is necessary to perform the first stage in scene generation, the in-betweening, for each ray, and whether it might be equally efficient to perform a four dimensional intersection test against the space-time volume.

Glassner [14] discusses the use of space-time bounding volumes for the optimisation of motion blur intersections. However the techniques used are not directly applicable to the generation of 4D volumes of parametric surfaces. Consequently it is necessary to find some method of creating a good 4D bounding volume which will encompass t.

As has been previously discussed, the use of 4D NURBS generally has the effect of increasing the order of the NURBS created, and in doing so means the control point mesh gives a better approximation of the surface. As this is true for 3D NURBS generated using the 4D NURBS, it is also true that the 4D NURBS' bounding volume gives an acceptable 4D space-time bounds.

In principle then the control point array of the 4D NURBS can be used as our volume. The 4D NURBS describes the behaviour of the NURBS throughout a time period, not just the exposure time we are concerned with. Consequently the full 4D bounds would give an overly large volume, meaning that for many situations we would still have to generate the Bézier patches in order to determine whether the ray intersected the surface, only to find that the ray misses the surface.

Recall that the 4D NURBS control volume is stored as a 3D array of control points in 3-space, with each slice corresponding to some value t, at which point the NURBS is enclosed by that volume. If we have a 4D NURBS composed of n slices, where the mth slice represents the NURBS at t_m, and we have a ray at t_{ray} then we can guarantee that the NURBS at t_{ray} is enclosed by the 4D bounding volume whose end-points are determined by the slices i & j, such that $t_i \leq t_{ray} \leq t_j$, where $i\&j \leq n$. This is illustrated in figure 5.1.

This means that in order to determine a reasonably tight 4D bounding volume we need only determine the slices before and after t_{ray}, and then use these control points as the bounds.

When we come to test a ray for intersection we need to determine whether the ray intersects any control volume which could be constructed using that portion of the 4D control array. This can be performed by splitting the 4D array section into patches. Each facet on the control point array on one of the slices corresponds to one facet on the next slice. If we view the volume generated between these two facets as the bounding volume, then we can perform intersection tests for all 6 faces.

As we have a 4D NURBS, whose 3D sections are NURBS with n by m control points, then we must perform $6 * n * m$ intersection tests for each ray in order to determine whether we need to in-between to perform an accurate intersection test (See figure 5.2).

This then gives us two different optimisations, each of which attempts to reduce the workload for tracing each ray, by reducing the amount of work required if the ray will miss the object. Of course this will slightly increase the workload if the ray *does* hit the NURBS, but in most circumstances the NURBS will only cover a portion of the viewing screen.

As has been previously mentioned the cost for intersecting the ray against a NURBS is extremely high. If we take this cost as irreducible the ideal situation would then be for only this work to be performed for each ray. If we can remove the work involved in generating the in-betweens completely then motion blurring

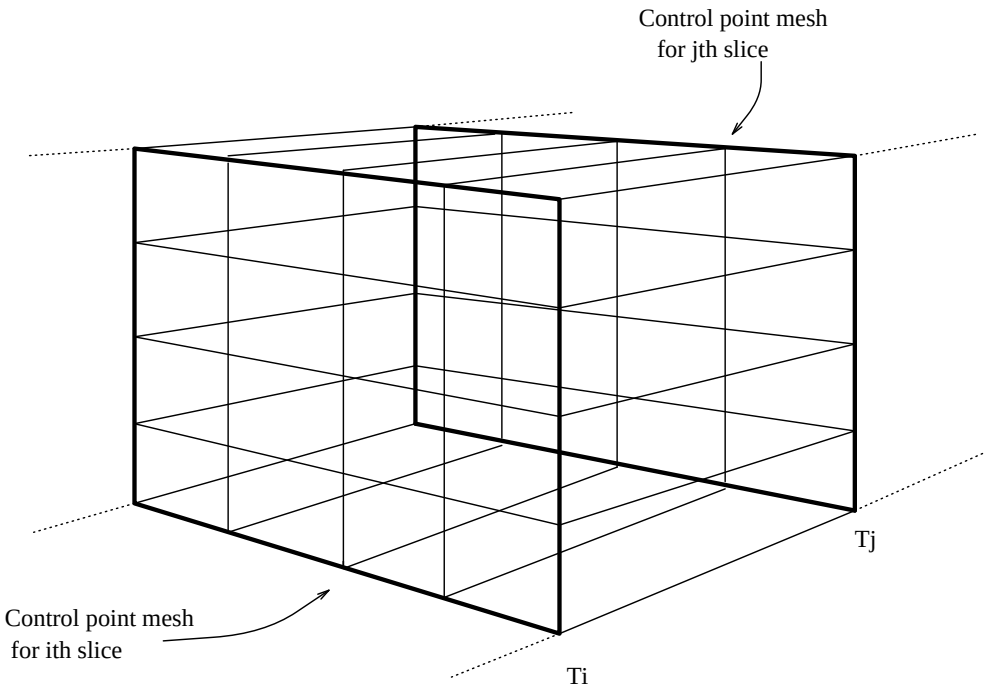


Figure 5.1: 2 Slices of the 4D Control array

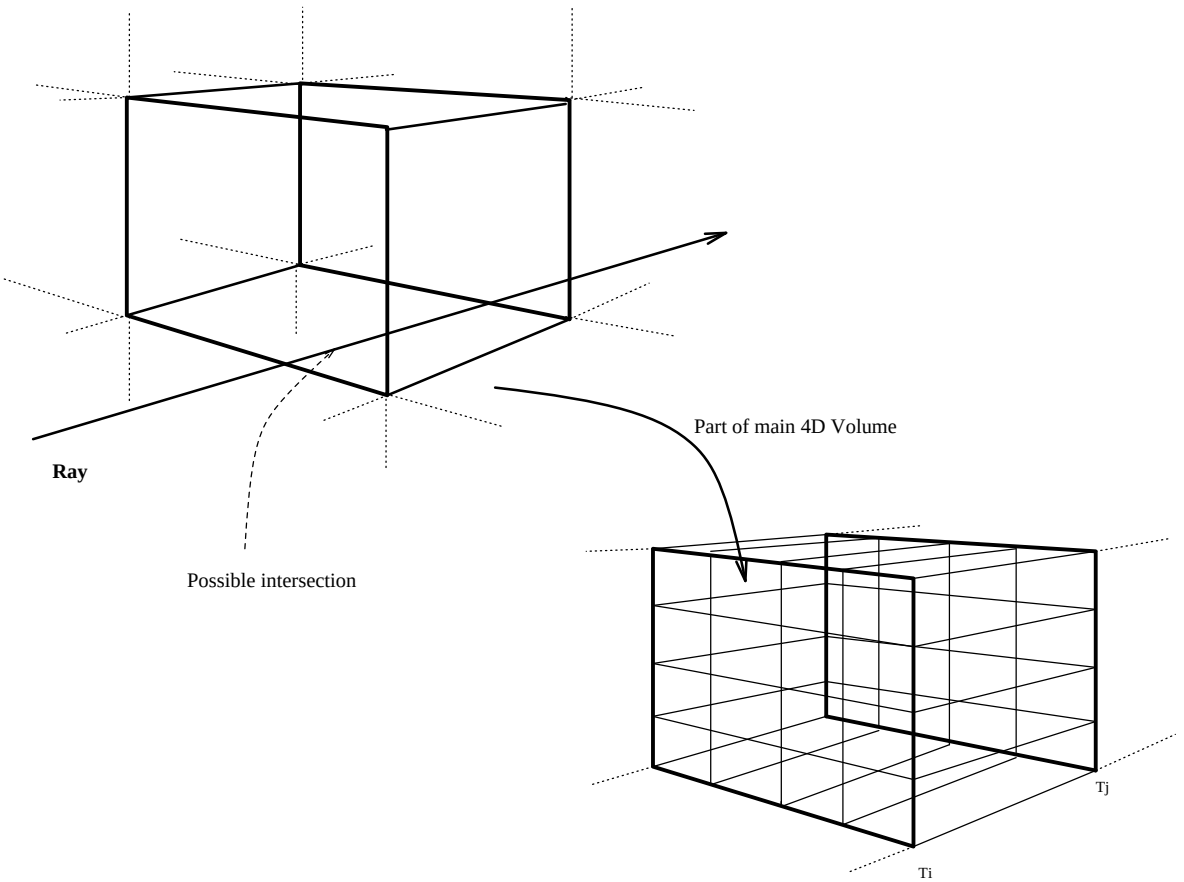


Figure 5.2: Intersecting a ray against part of a 4D Volume

NURBS will cost little more than generating static images.

One obvious way to achieve this is to cache the Bézier patch surface & bounding box hierarchy, thereby removing the need to do all the work unless absolutely necessary. However the stochastic nature of the distributed sampling, (no pattern & no bunching), means that in theory we should never re-use the information generated for a particular t.

A simplified version of this would be to pre-calculate a number of in-betweening steps, which would then be used for each ray. This would mean that the only effort expended would be at the beginning of the scene, and from then on we can generate motion blurred pictures at the same speed as we could static ones. Obviously this means that we are reducing the quality of the approximation to the integration properties of the film, and so we will produce lower quality images.

In order to avoid too much temporal aliasing we can use a stochastically distributed series of samples. This means then at the beginning of the scene we determine a series of t's at which we will sample. We then in-between the NURBS at these times, and generate the Bézier patches and bounding box hierarchies. This data is cached. So we might choose to use 16x super-sampling, which means we need to generate, and store, 16 pre-calculated in-betweens. For each pixel we determine which ray uses which of the in-betweens, which are then ray-traced as normal.

This will produce acceptable results provided

- The scene contains objects which do not move across the field of view too much.

- The camera does not move too quickly

If these constraints are broken then banding will occur in the image, which may be visible to the human eye. However as we will re-calculate the samples for each frame there should be no pattern to the banding, so limited aliasing should occur.

We now have 3 different optimisations which can be measured against the naive implementation of in-betweening, splitting into Bézier patches, and calculating the bounding box for each and every ray. These optimisations are

- In-between for every ray to create the 3D NURBS surface at t. However the ray is then tested for intersection against that control point mesh. If no intersection occurs then we can record a miss and go on to consider the next ray, otherwise we can split the NURBS and proceed as we would in the naive implementation.

- The ray is tested for intersection against the 4D Volume formed by slices of the 4D NURBS control point array. If an intersection occurs then we need to in-between, and split as normal, but otherwise we can proceed to the next ray.

- A number of pre-calculated in-betweens & splits are generated and cached. From them on we use different scene caches for different rays.

The final optimisation will produce pictures in a fraction of the time required by the other implementations, but will produce pictures of a far lower quality. However if the results are better than would otherwise be obtained, at minimal cost, then this technique may be useful when rendering objects such as NURBS.

Consequently we are resolving two questions. Firstly whether it is more efficient to sacrifice a tight bounds for the expense of in-betweening, i.e., whether intersecting against the 4D volume (which requires 6 times as many intersection tests as the ordinary control point polygon) is faster than in-betweening & then testing for intersection.

Secondly whether it is feasible to sacrifice the quality of approximating the integration property of film for the speed of image generation.

Chapter 6

Implementation & Results

6.1 Optimisations

6.1.1 Control Volume Intersection

This optimisation works by intersecting the ray against the volume formed by the control point mesh of the NURBS we are interested in ray tracing. As has been previously mentioned the control volume gives an unreliable fit of the actual surface, as lower order NURBS will have a larger control volume. However the control volume will always enclose all the NURBS surface, so whilst under certain conditions it will give an inaccurate impression of the size of the surface it requires no pre-calculations.

As the volume may be concave it is difficult to use optimised intersection tests, most of which rely on a convex volume. Consequently the easiest way of intersecting a ray against the volume is to intersect it against the planes formed between

adjacent control points.

If we have a NURBS which is defined by a control point mesh of n by m points, then one of the planes which we intersect against is defined by $P_{i,l}, P_{i,m}, \& P_{j,l}$, where $i, j \& l$ are indices into the control point mesh.

```
for(object=1; object < number_of_objects; object++)
{
  inbetween_to_time(t, object);
  distance = intersect_against_poly(object, ray);

  if(distance > 0.0)
  {
    convert_to_bez_segs(object);
    intersect_ray(object,ray);
  }
}
```

6.1.2 Four Dimensional Volume intersection

Intersecting against the four dimensional volume can be performed by determining two control point meshes, which, at different points of t, define a bounding volume of the surface. This volume will give an overly large representation of the size of the surface at a particular time, but it requires no pre-calculation, and therefore if the ray misses this volume will require little extra effort.

The 4D volume may be convex, which means that we must use inefficient plane

intersection tests to determine whether the ray touches the volume. If we have two meshes, which define the surface at t_1 & t_2 respectively, then we can be sure that each shares a common number of control points in $u\&v$. This means that we can think of each planar facet in one mesh having a corresponding facet in the next mesh. Therefore intersection against a 4D volume can be implemented by treating the space between each individual facet as a six sided volume, which can be intersected against using the same algorithms used for conventional control volume intersection tests.

There will be a large number of unnecessary tests, but removing these is difficult without expending extra time in pre-calculations. As the motive behind using 4D volume intersection is to remove this work such optimisations are unnecessary.

```
for(object=0; object < number_of_objects; object++)
{
  calculate which two slices form the bounds at t;
  distance = volume_int_func(slice1, slice2, ray);
  if(distance > 0.0)
  {
    inbetween_to_time(t, object);
    convert_to_bez_segs(object);
    intersect_ray(object,ray);
  }
}
```

where the volume intersection functions consists of

```
for(u=0; u< number_of_cps_in_u; u++)
  for(v=0; v < number_of_cps_in_v; v++)
  {
    intersect against each face, if an intersection is found return the
      distance to that point;
  }
```

6.1.3 Multiple exposures/Distributed ray tracing

This last optimisation attempts to combine the reduced work required when producing multiply exposed pictures, and the consistent quality produced using distributed ray tracing techniques.

It employs the technique used in the naive implementation, of in-betweening the NURBS to generate the 3D surface, converting it to Bézier patches, and calculating the bounding volume hierarchy, a static number of times. When the time comes to trace a ray for a given pixel we use one of the pre-calculated scene stores.

To remove as much aliasing as possible the pre-calculations are not assigned to a given ray. So for example the 5th sample will not always be used for the 5th ray, as this may cause problems.

Once the pre-calculations have been performed the screen should take the same amount of work to render, as would be required in a scene without any motion blur. However in a scene containing a large number of moving ojects we need to calculate a static number of in-betweens for each of them. This means that

the storage requirements may quickly become excessive. Due to the extremely high rendering costs of NURBS this may become acceptable, as we will still make considerable savings in time.

6.2 Implementation Issues

The ray tracer used to test the different motion blur optimisations was developed on a Meiko Computing Surface containing T800 Inmos Transputers, Intel i860s and a SPARC chip, using libraries written at the CGU [16, 17]. The parallelisation used was a form of the Master-Slave algorithm, where a large number of ray tracing' processors each ray traces a small portion of the screen, and a single processor coordinates them, ensuring work is not duplicated.

The ray tracer [9] was implemented in Meiko C. The run time library provided by the compiler uses a simplistic method of memory management, which caused problems.

The optimisations used to reduce the amount of work performed by the ray tracer when motion blurring NURBS mean that space will not always be freed in the order in which it was claimed, which means that using the normal library calls it was possible for the program to run out of memory. As the T800 transputers do not implement virtual memory, this meant that an alternate memory allocation library had to be written.

In timings for the first two optimisations this alternate library was used, whereas in the final (pre-calculation) optimisation it was not. This means that the timings

for the first two results are worse than they would otherwise be, though in view of the marked difference between costs this makes little difference to the comparisons, as the types of algorithms are designed to address different problems.

6.3 Results

In order to measure the performance benefits of each of the different optimisations they were implemented, and used to motion blur a sample animation.

The frame being motion blurred forms part of a longer animation showing the melting of a coffee cup. The scene contains 3 objects, a polygon base, a mug body, and handle, with the last two represented using NURBS.

The 4D NURBS was generated using a cubic trajectory NURBS, and 3 sections in time, showing the full cup, a bent cup, and a flattened version. The frame to be motion blurred has an exposure period between $t = 0.2$ & $t = 0.4$. Whilst this particular animation would probably not be motion blurred when used as a part of a film, the scene contains all the problems exhibited in real film production (e.g., the table is static, and so the algorithm must not attempt to blur that).

Plates 1 – 6 show evaluations of the 4 dimensional NURBS used for the tests. Plate 7 was generated using the first (control point) optimisation, and shows the motion blur caused using one ray per pixel on just the mug body. Plate 8 shows a frame generated using the pre-calculation optimisation in 5 hours, with an exposure beginning at $t = 0.2$ and ending at $t = 0.4$, using 16 pre-calculated in-betweens for each NURBS. If the pre-calculation technique is used to motion

Implementation type	*Time for 1x sampling*	*Expected 16x time*[a]	*Actual 16x time*[b]
No optimisations	52 hours	832 hours	...
Control Volume	42.5 hours	680 hours	...
4D Intersection	38 hours	608 hours	...
Pre-Calculated	...	...	5 hours

[a] As timing the performance of this algorithm when performing a full 16x oversampling of the screen is impossible this value is an estimate of the time required.

[b] Measured time taken

Table 6.1: Timings for application of the 3 different optimisations to a sample animation

blur a scene containing too much motion banding will occur, as shown in plate 9, where a large sphere changes into a much smaller egg–cup during one exposure. The mouth of the egg–cup is facing the viewer near the middle of the plate.

Each picture was generated on 30 T800 transputers, and the time taken from beginning the ray tracer to completing the frame is measured.

Chapter 7

Conclusions

The 3 different optimisations attempt to improve upon the costs of performing motion blurring of NURBS by variations on two strategies.

- Determining how much of the work for each ray has to be performed, rather than blindly doing all the work for each and every ray.
- Approximating the integration properties of film by performing a reduced amount of super-sampling through time. By ensuring no pattern exists in the sampling times any undesirable aliasing can be minimised.

It is necessary, then, to resolve two questions. Firstly whether performing intersection tests in space time provides any benefit in motion blur algorithms. Secondly whether approximating the effects of distributed ray tracing gives acceptable results, or is the reduction in quality an acceptable penalty of the reduced execution time.

Firstly it is shown in table 6.1 that both NURBS control volume intersection strategies significantly reduce the time required to produce a motion blurred scene. This means that the costs of generating the Bézier patches & bounding volume hierarchy are so expensive that performing a considerable amount of work to determine whether it needs to be done is advisable.

The first question can be expressed as a comparison between the costs of the time required to produce the full screen. The costs of generating a scene using 3D volume intersection tests can be represented as:

$$T = r_n * c_{in-betweening} * c_3 + h_3 * g \tag{7.1}$$

where

T is the total cost of generating the screen.

r_n is the Number of rays required to render the screen.

$c_{in-betweening}$ is the cost of in-betweening the 4D NURBS to generate a surface.

c_{3d} is the cost of intersecting against the surface control volume.

h_3 is the number of times in the scene which the ray is found to intersect against the 3D volume.

g is the cost of generating the Béezier patches and bounding box hierarchy.

The costs of using the 4D Volume intersection test can also be expressed as:

$$T = r_n * c_4 + (h_4 * c_{in-betweening} * g) \tag{7.2}$$

where

c_4 is the cost of intersecting a ray against the 4D volume.

h_4 denotes the number of times a ray is found to intersect with the 4D volume.

Table 6.1 shows that an optimisation based on intersecting a ray against a 4 dimensional volume, in order to determine whether an in-betweening needs to be performed, gives a slightly better solution in comparison to the strategy of in-betweening the 4D NURBS for every ray, then intersecting the ray against those bounds.

This means it can be said that:

$$(r_n * c_4 + (h_4 * c_{in-betweening} * g)) < (r_n * c_{in-betweening} * c_3 + h_3 * g) \qquad (7.3)$$

Bearing in mind that c_4 is at least 4 times more expensive than c_3, then two conclusions can be drawn from this

Firstly if the 4 dimensional intersection tests provide a higher number of 'false' intersection results (i.e., no intersection actually takes place with the surface, but we do not know that from the 4D test) then the extra cost associated with generating actual results is not enough to make this technique too expensive.

Secondly the cost of generating the in-betweened NURBS surface at t is the most expensive part of these algorithms. By attempting to perform this as infrequently as possible the 4 dimensional intersection tests provide a significant performance boost.

The second major classification of optimisations, the approximation to the 'ideal' blurred frame, is difficult to rate objectively. The technique works by exploiting

the inadequacies of the human eye. Through reducing patterns in the scene, but still only sampling a small number of points through the exposure time, the technique hopes to provide motion blurred animations, rather than frames.

This technique will fail to produce acceptable animation when either the exposure time is set to high, or objects are changing/moving too much during that time. The rough approximation to the integration properties of real film rely on small exposure times, and relatively 'quiet' scenes. The tests performed illustrate that this strategy produces blurred pictures, but without extensive tests it is impossible to determine at what point the quality becomes unacceptable.

To find a measure of the quality it would be necessary to produce a sequence of test animations, showing a given animation using different exposure times, and/or a changing number of pre-calculated samples. By showing the films to an audience of varying ages (and therefore quality of vision) it should be possible determine the points at which the animation quality becomes unacceptable. The expense and time of such a trial has precluded its use for this thesis.

The main disadvantage to this pre-calculation technique, is the cost of holding the Bézier patches & bounding volume hierarchy, for each of the temporal samplings which we cache. If this technique is used to generate motion blurred frames of a complicated scene, with many moving NURBS, then the costs quickly become unrealistic.

This thesis then has presented 3 different optimisations to the motion blur technique of distributed ray tracing for use with NURBS. It has shown that the 4 dimensional nature of NURBS, together with the cost of evaluating the surface

at a particular time, means that it is advantageous to perform surface/ray intersection tests in 4 dimensions, in an attempt to reduce the amount of work required.

In addition an alternate approach to generating motion blurred scenes has been introduced, which combines the benefits of multiple exposures, and stochastic sampling, to produce frames of an animation of a lower quality than would otherwise be achieved, but at a considerably reduced cost.

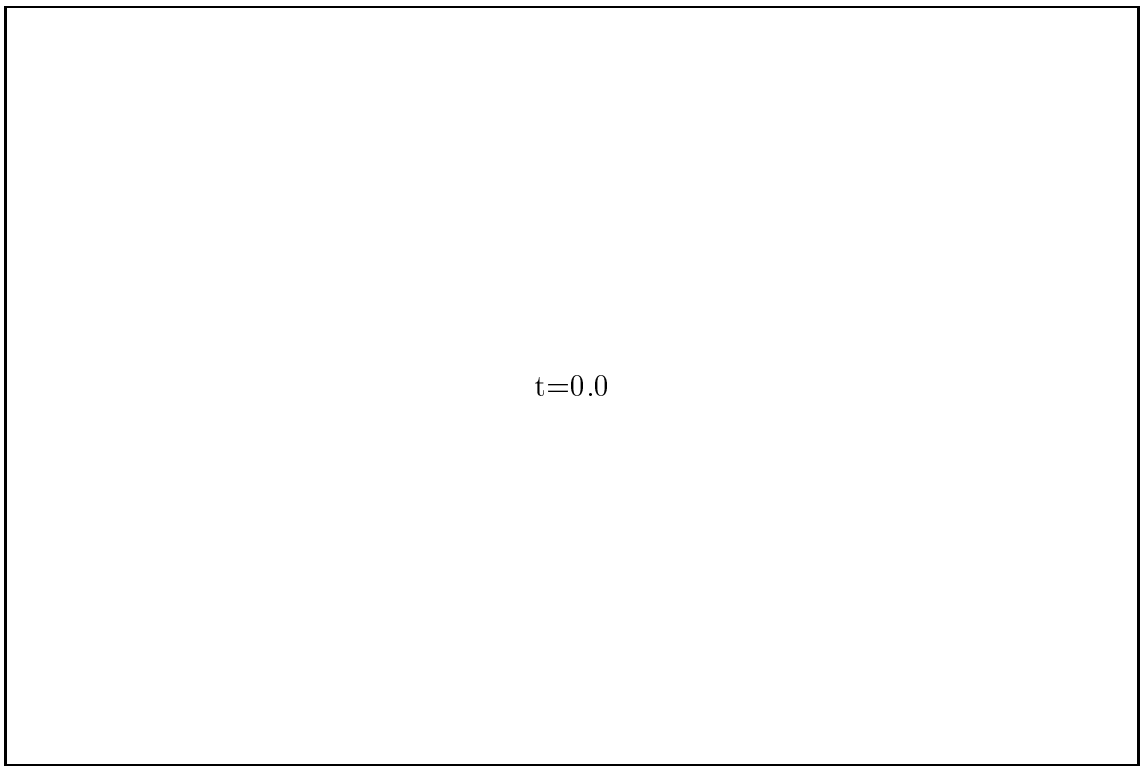


Plate 1: 4D NURBS at $t = 0.0$

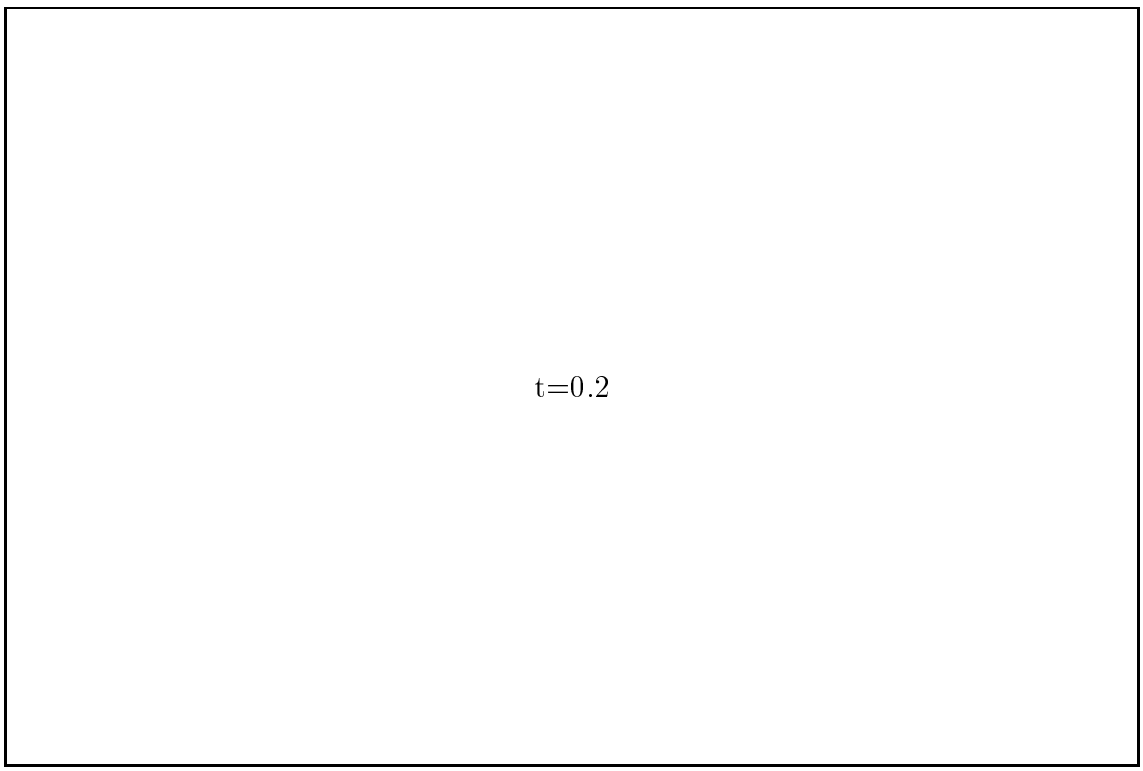


Plate 2: 4D NURBS at $t = 0.2$

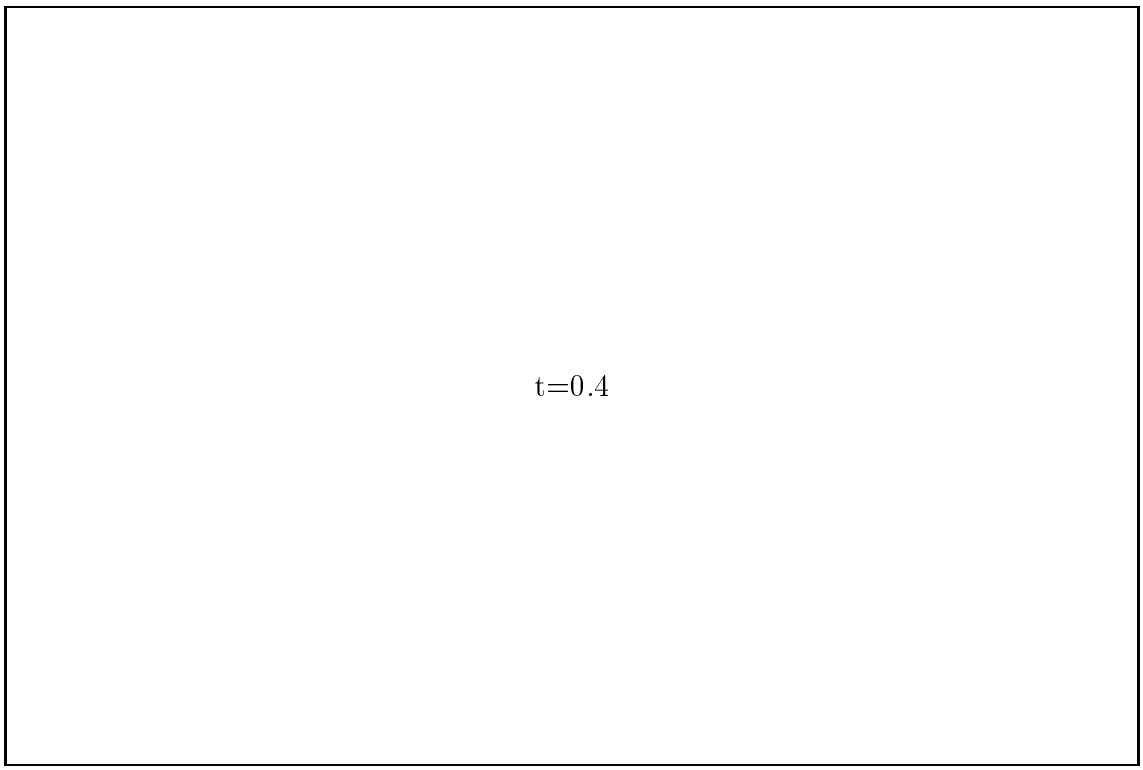


Plate 3: 4D NURBS at $t = 0.4$

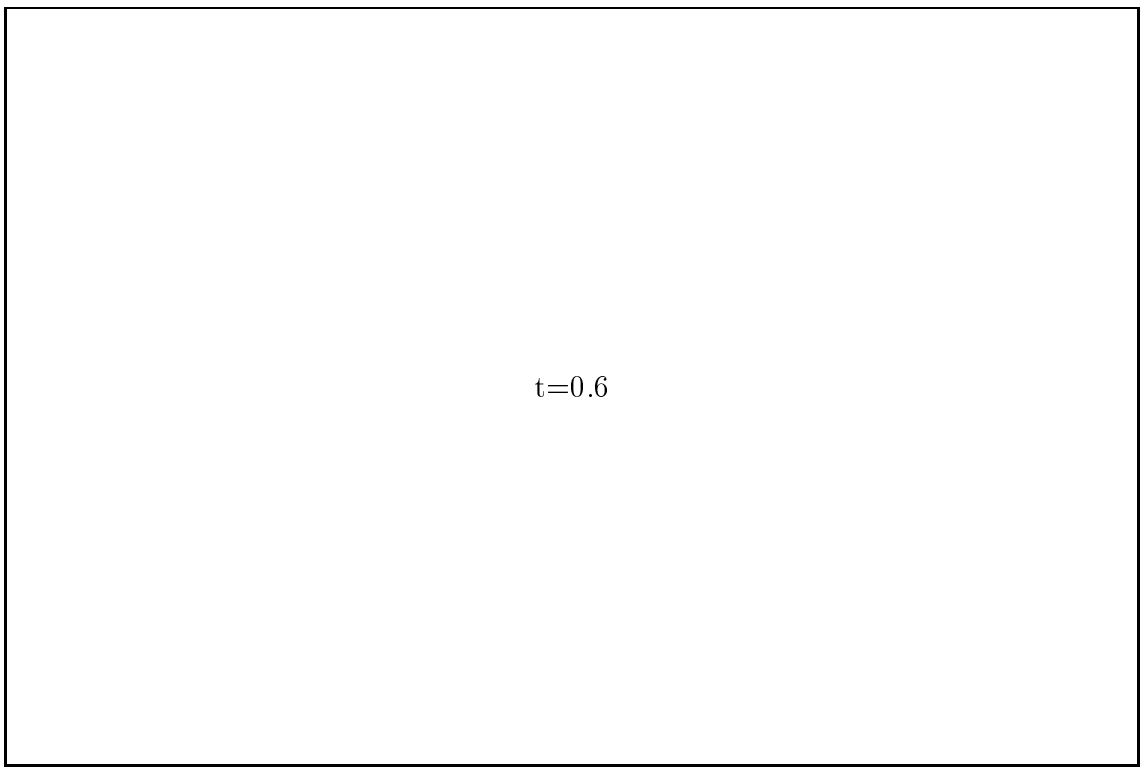


Plate 4: 4D NURBS at $t = 0.6$

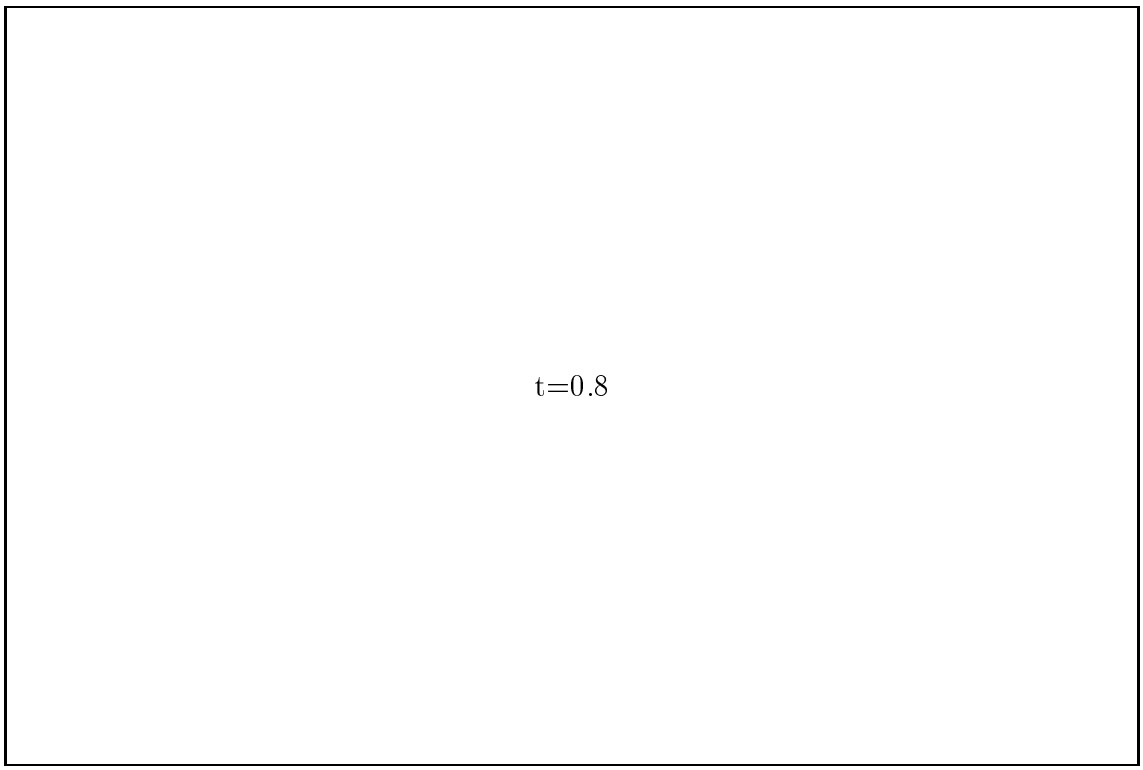


Plate 5: 4D NURBS at $t = 0.8$

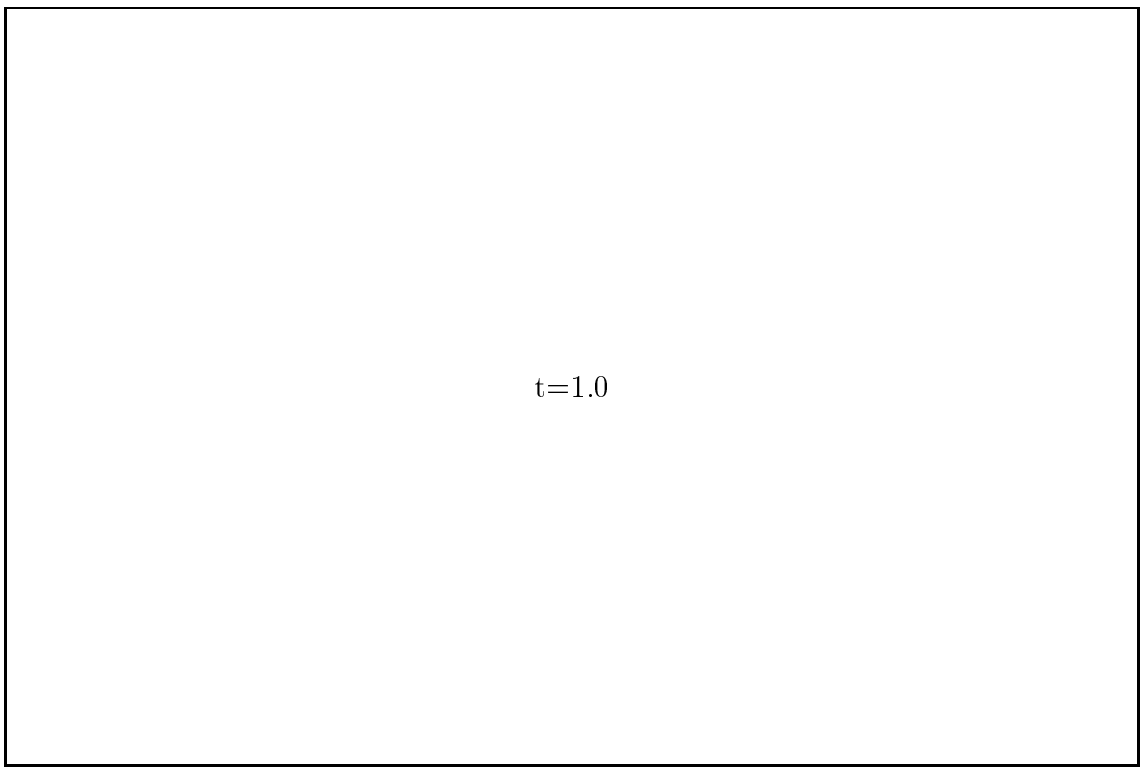


Plate 6: 4D NURBS at $t = 1.0$

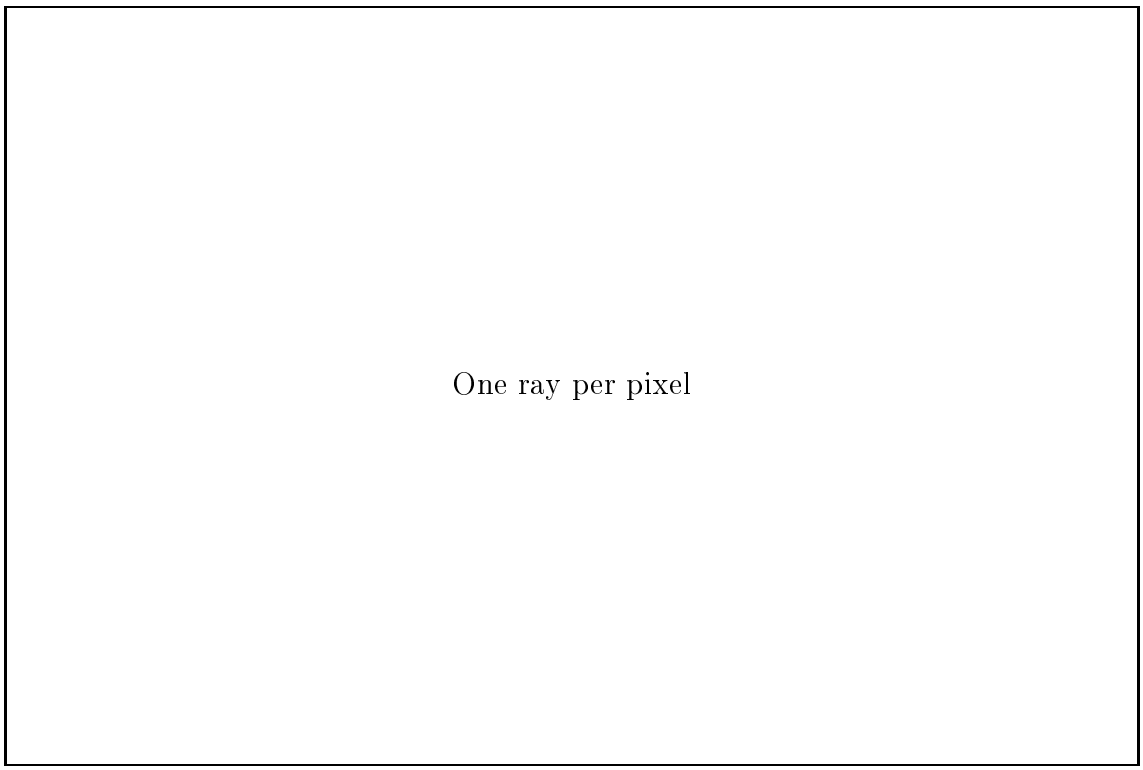


Plate 7: 1 ray per pixel motion blurred cup body

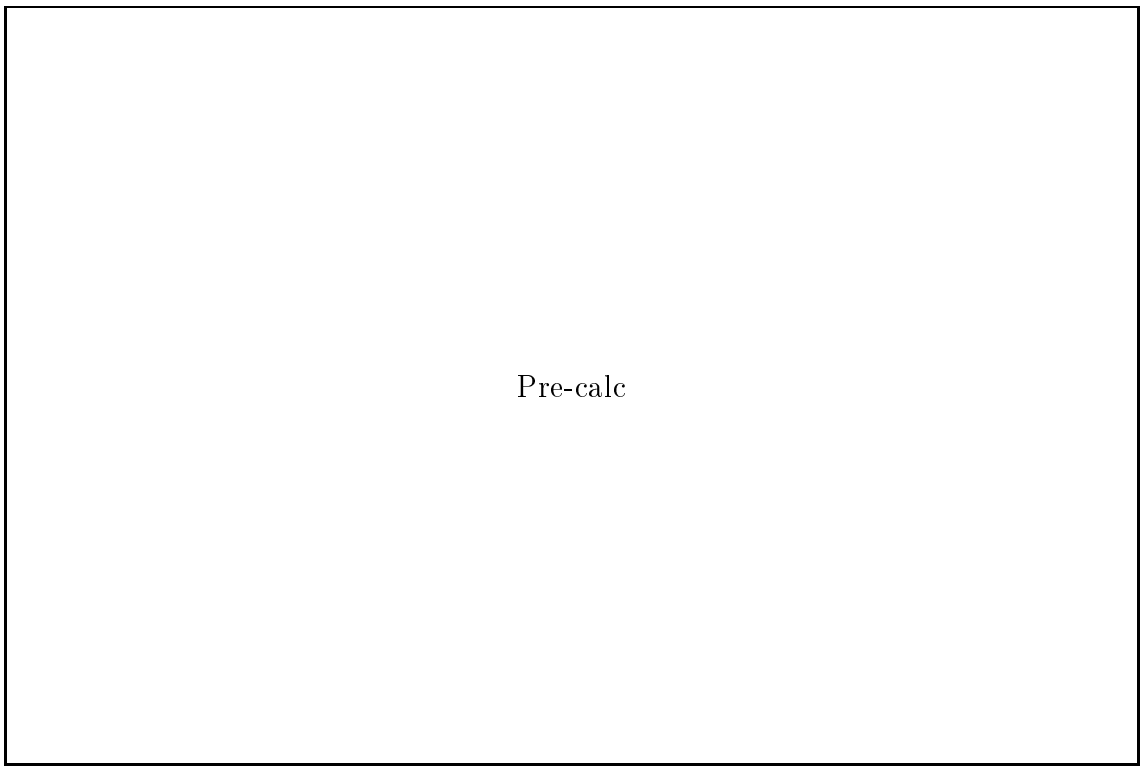


Plate 8: Motion blurred frame with the exposure time between $t = 0.2$ & $t = 0.4$

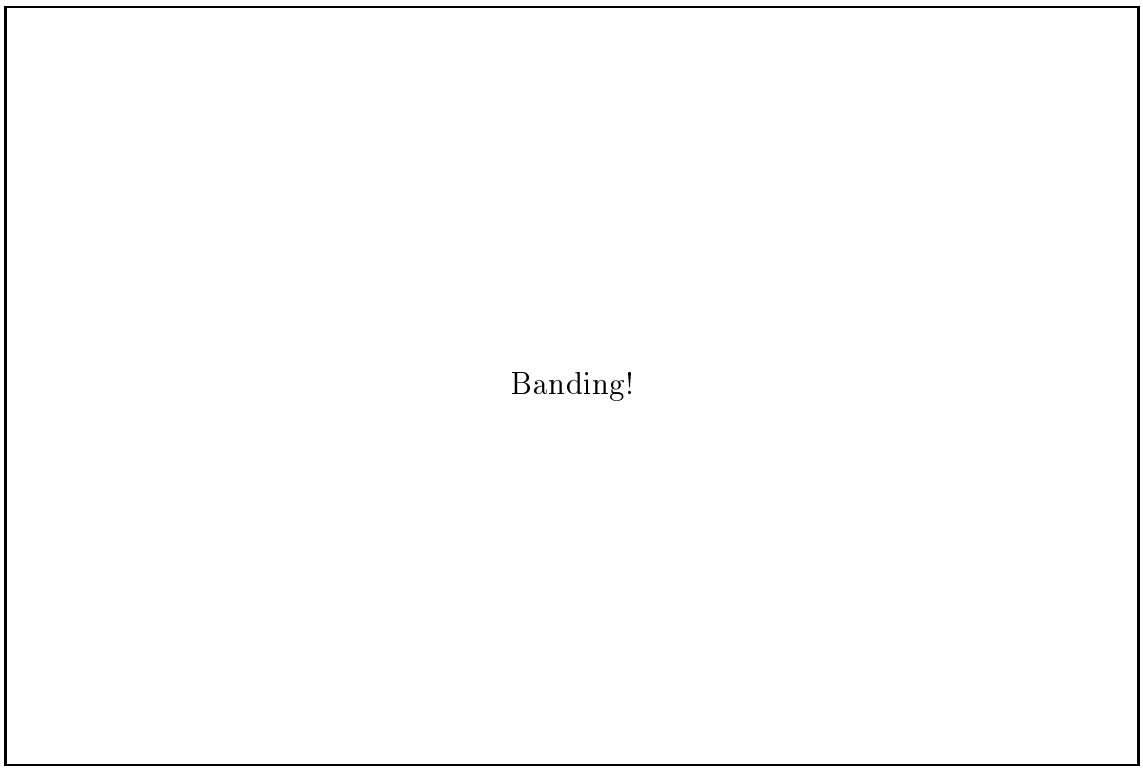


Plate 9: A large sphere changing into a smaller egg-cup.

References

[1] Salim Abi-Ezzi and Leon A. Shirman. Tesselation of Curved Surfaces under highly varying transformations. In *Proceedings of Eurographics '91*. Eurographics, 1991.

[2] Andrew S. Glassner. *An Introduction to Ray-Tracing*. Academic Press, 1989.

[3] A. Appel. Some techniques for shaded machine rendering of solids. In *Proceedings of the Spring Joint Computer Conference*, pages 37–45, 1968.

[4] Pierre Bézier. *Numerical Control — Mathematics and Applications*. Wiley, London, 1972. Translated by A. R. Forrest and A. F. Pankhurst.

[5] Pierre Bézier. Mathematical and Practical Properties of UNISURF. In R. E. Barnhill and R. F. Reisenfeld, editors, *Computer Aided Geometric Design*. Academic Press, New York, 1974.

[6] Preston Blair. *Animation*. Walter T. Foster, Santa Ana CA, 1949.

[7] Wolfgang Boehm. Inserting New Knots into B-Spline Curves. *Computer Aided Design*, 12(4):199–201, July 1988.

[8] E Coehn, T Lyche, and L L Schumaker. Algorithms for Degree Raising of Splines. *ACM Transactions on Graphics*, pages 171–181, July 1985.

[9] Geoff Cook. Parallel Ray Tracing of Non Uniform Rational B-Splines. Master's thesis, University of Manchester, 1992.

[10] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. *ACM SIGGRAPH Computer Graphics*, 18(3):137–144, Jul 1984.

[11] Mark A. Z. Dippé and Erling Henry Wold. Anti-aliasing Through Stochastic Sampling. *ACM SIGGRAPH Computer Graphics*, 19(3):69–78, Jul 1985.

[12] Tom Duff. Compositing 3-D Rendered Images. *ACM SIGGRAPH Computer Graphics*, 19(3):41–44, Jul 1985.

[13] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics, Principles and Practice.* Addison-Wesley, second edition, 1990.

[14] Andrew S. Glassner. Spacetime Ray Tracing for Animation. *IEEE Computer Graphics & Applications*, pages 60–70, Mar 1988.

[15] Henri Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, pages 623–629, 1971.

[16] W. T. Hewitt and D. Yip. The NURBS Procedure Library. Technical report, Computer Graphics Unit, University of Manchester, 1992. CGU 76.

[17] T.L.J. Howard, W.T. Hewitt, J.G. Williams, and S. Larkin. Phigs toolkit user guide. Technical Report PTK/100, University of Manchester, September 1992.

[18] J.I. Yellott (Jr). Spectral consequences of photoreceptor sampling in the rhesus retina. *Science 221*, pages 382–385, July 1983.

[19] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for Distributed Ray Tracing. *ACM SIGGRAPH Computer Graphics*, 19(3):61–67, Jul 1985.

[20] Nelson L. Max and Douglas M. Lerner. A Two-and-a-Half-D Motion-Blur Algorithm. *ACM SIGGRAPH Computer Graphics*, 19(3):85–93, Jul 1985.

[21] Don. P. Mitchell. Spectrally Optimal Sampling for Distribution Ray Tracing. *ACM SIGGRAPH Computer Graphics*, 25(4):157–164, Jul 1991.

[22] Bui-Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[23] Leslie Piegl and Wayne Tiller. Curve and Surface Constructions using Rational B-Splines. *Computer Aided Design*, 19(9):485–498, Nov 1987.

[24] Thomas Porter and Tom Duff. Compositing Digital Images. *ACM SIGGRAPH Computer Graphics*, 18(3):253–258, Jul 1984.

[25] Michael Potmesil and Indranil Chakravty. A Lens and Aperature camera model for Synthetic image generation. *ACM SIGGRAPH Computer Graphics*, 15(3):297–305, Aug 1981.

[26] Michael Potmesil and Indranil Chakravty. Modelling Motion Blur in Computer Generated Images. *ACM SIGGRAPH Computer Graphics*, 17(3):389–399, Jul 1981.

[27] Martin Preston. In-Betweening in 3D Animation. 3rd Year UG Project Report, June 1991. University of Manchester.

[28] R.V. Shack. The influence of Image motion and shutter operation on the photographics transfer function. *Applied Optics*, 3(10):1171–1181, 1964.

[29] Wayne Tiller. Rational B-splines for curve and surface representation. *IEEE Computer Graphics & Applications*, pages 61–69, September 1983.

[30] R. Cook & K. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

[31] Alle-Jan Von Der Veen. Intersection test for nurbs. In *IEEE Symposium on Computer Architecture & Real time Graphics*, pages 101–114, Jun 1989.

[32] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, Jan 1984.

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