FLINDERS
UNIVERSITY
ADELAIDE
AUSTRALIA

Trilinear Projection

by

Scott Vallance

B.Sc. (Hons) (Flinders University of South Australia) 1999

A thesis presented to the
Flinders University of South Australia
in total fulfillment of the requirements for the degree of
Doctor of Philosophy

Adelaide, South Australia, 2005

© (Scott Vallance, 2005)

Certification

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

As requested under Clause 14 of Appendix D of the *Flinders University Research Higher Degree Student Information Manual* I hereby agree to waive the conditions referred to in Clause 13(b) and (c), and thus

- Flinders University may lend this thesis to other institutions or individuals for the purpose of scholarly research;
- Flinders University may reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed Dated

Scott Vallance

Abstract

In computer graphics a projection describes the mapping of scene geometry to the screen. While linear projections such as perspective and orthographic projection are common, increasing applications are being found for nonlinear projections, which do not necessarily map straight lines in the scene to straight lines on the screen. Nonlinear projections occur in reflections and refractions on curved surfaces, in art, and in visualisation.

This thesis presents a new nonlinear projection technique called a trilinear projection that is based on the trilinear interpolation of surface normals used in Phong shading. Trilinear projections can be combined to represent more complicated nonlinear projections.

Nonlinear projections have previously been implemented with ray tracing, where rays are generated by the nonlinear projections and traced into the scene. However for performance reasons, most current graphics software uses scanline rendering, where a scene point is imaged on a screen as a function of the projection parameters. The techniques developed in this thesis are of this nature.

This thesis presents several algorithms used in trilinear projection:

1. An algorithm to analytically determine which screen locations image a given scene point.

2. An algorithm that correctly connects projected vertices. Each scene point may be imaged multiple times, which means a projected scene triangle may form from one to four different shapes of from two to nine vertices. Once connected, the projected shapes may be rendered with standard scanline algorithms.

3. An algorithm to more accurately render the curved edges between projected vertices.

4. A scene-space edge-clipping algorithm that handles continuity issues for projected shapes across composite projections.

The trilinear projection technique is demonstrated in two different application areas: visualisation, and reflections and refractions. Specifically, various nonlinear projections that are congruent with pre-existing visualisation techniques are implemented with trilinear projections and a method for approximating the reflections and refractions on curved surfaces with trilinear projections is presented. Finally, the performance characteristics of the trilinear projection is explored over various parameter ranges and compared with a naive ray tracing approach.

Contents

List of Figures vii

List of Tables x

1 Introduction 1

1.1 Nonlinear Projection . . . 1
1.1.1 Artistic Nonlinear Projection . . . 2
1.1.2 Strip Cameras . . . 2
1.1.3 Multi-Perspective Images as a Basis for Resynthesis . . . 4
1.1.4 Visualisation with Nonlinear Projection . . . 4
1.2 Ray tracing and Scanline Rendering . . . 6
1.3 Thesis Scope . . . 6
1.4 Thesis Overview . . . 7

2 Nonlinear Projection Surfaces 9

2.1 Projection Surfaces . . . 9
2.2 Mesh Surfaces and Surface Continuity . . . 10
2.3 Representing a Curved Surface with a Trilinear Interpolated Triangle . . . 11
2.3.1 Interpolation . . . 11
2.3.2 Phong Shading . . . 12
2.4 Ray Tracing with a Trilinear Projection . . . 13
2.5 Summary . . . 13

3 Projecting a Point with a Trilinear Projection 15

3.1 Projecting a Point with a Trilinear Projection . . . 16
3.2 Treating the Trilinear Projection as a Parametric Triangle . . . 16
3.3 Determining the Coplanarity of the Parametric Triangle and the Scene Point 17
3.4 Barycentric Coordinate Conversion . . . 18
3.5 Multiple Solutions . . . 20
3.6 Precalculating Partial Coefficient Values . . . 20
3.7 Parametric Triangle Sides and Containment . . . 21
3.8 Algorithm for Projecting a Scene Point . . . 24
3.9 Summary . . . 25

4 Projecting a Triangle with a Trilinear Projection **26**
4.1 Drawing a Scene Triangle . . . 26
4.2 Determining Shapes . . . 27
4.3 Drawing Shapes . . . 28
4.3.1 An Algorithm to Connect Vertex Lists . . . 31
4.4 Example Shape Images . . . 32
4.5 Tessellation . . . 37
4.5.1 Scene Triangle Tessellation . . . 37
4.5.2 Parametric Triangle Slices . . . 38
4.6 Viewing Plane Intersection . . . 40
4.7 Summary . . . 42

5 Multiple Surface Triangles **44**
5.1 Screen Space Clipping . . . 44
5.2 Scene Space Clipping . . . 45
5.2.1 Algorithm for Determining Intersection t Values . . . 46
5.2.2 Integrating Intersection Points into Drawing Primitives . . . 47
5.3 Summary . . . 50

6 Nonlinear Projection for Visualisation **51**
6.1 Detail and Context . . . 51
6.2 Multiple Perspective Views . . . 53
6.3 Mappings . . . 56
6.4 Summary . . . 58

7 Reflections and Refractions **60**
7.1 Integrating Reflection and Refraction Projections into a Scene . . . 60
7.2 Reflections . . . 60
7.2.1 First-Hit Reflections on a Polygon Mesh . . . 61
7.2.2 Approximated First-Hit Reflections . . . 62
7.2.3 Multi-Hit Reflections on a Polygon Mesh . . . 63
7.3 Refraction . . . 63
7.3.1 First-hit Refraction on a Polygon Mesh . . . 64
7.3.2 Approximating Refraction . . . 64
7.3.3 Multi-Hit Refraction on a Polygon Mesh . . . 64
7.4 Example Projections . . . 65
7.5 Summary . . . 66

8 Performance Evaluation **70**
8.1 Experimental Conditions . . . 70
8.1.1 Data . . . 71
8.1.2 Caveats . . . 71
8.2 Ray Tracing . . . 72
8.3 Trilinear Projection . . . 74
8.4 Speed up . . . 76

8.5 Tessellation Methods . . . 76
8.6 Multiple Trilinear Projections . . . 80
8.7 Nonlinear Projection for Visualisation, and Reflections and Refractions . . . 82
8.8 Summary . . . 84

9 Related Work **85**
9.1 Ray Tracing . . . 85
9.1.1 Beam Tracing . . . 86
9.1.2 Spatial Subdivision . . . 87
9.1.3 Hardware Ray Tracing . . . 87
9.2 Ray Tracing Nonlinear Projections . . . 88
9.2.1 Ray Tracing with Extended Cameras . . . 88
9.2.2 Cubism and Cameras: Free-form Optics for Computer Graphics . . . 89
9.2.3 Multi-Perspective Images for Visualisation . . . 90
9.2.4 General Linear Cameras . . . 90
9.3 Scanline Rendering . . . 91
9.3.1 Multi-Pass Rendering . . . 91
9.3.2 Reflections on Spheres and Cylinders of Revolution . . . 92
9.3.3 Multiple-Center-of-Projection Images . . . 93
9.4 Object Distortion for Nonlinear Projections . . . 93
9.4.1 Distortion Methods for Visualisation . . . 93
9.4.2 Interactive Reflections on Curved Objects . . . 95
9.4.3 Specular Path Perturbation . . . 95
9.4.4 Region of Influence Cameras . . . 96
9.5 Approximating Reflections on Curved Objects with Image Based Rendering 96
9.5.1 Environment Mapping . . . 97
9.5.2 Extended Environment Mapping . . . 97
9.5.3 Parameterized Environment Maps . . . 98
9.5.4 Light Field Rendering . . . 99
9.6 Summary . . . 99

10 Conclusion **101**
10.1 Summary . . . 101
10.2 Contributions . . . 101
10.2.1 Projecting a Scene Point with Trilinear Projection . . . 102
10.2.2 Projecting a Scene Triangle with Trilinear Projection . . . 102
10.2.3 Parametric Triangle Slicing . . . 103
10.2.4 Scene Space Clipping . . . 103
10.2.5 The Application of Trilinear Projection in Visualisation . . . 103
10.2.6 The Application of Trilinear Projection in Rendering Reflections and Refractions on Curved Surfaces . . . 104
10.3 Further Work . . . 104
10.4 Conclusion . . . 105

A Expanded Equations **106**
A.1 Parametric Triangle and Scene Point Coplanarity Test 106
A.2 Line Segment Intersection Coplanarity Cubic 107
A.3 Precalculation for Cubic Coefficients 109

B Vector Properties **110**

C View and Scene Data **111**

D Tabulated Performance Results **114**
D.1 Ray Tracing Results 114
D.2 Trilinear Projection Results 118
D.3 Trilinear Projection with Scene Space Clipping Results 121
D.4 Ray Tracing on Different Configurations 124
D.5 Trilinear Projection on Different Configurations 126
D.6 Trilinear Projection with Scene Triangle Tessellation Results on Different Configurations 128
D.7 Trilinear Projection with Parametric Triangle Slicing Results on Different Configurations 131

E Context in Planar 3D Navigation **134**

F Multi-Perspective Images for Visualisation **142**

G Inward Looking Projections **151**

Bibliography **156**

List of Figures

1.1 "Fishermans Evening Song" by Xu Daoning, circa 11th Century 2
1.2 "High and Low" by M. C. Escher, an example of a nonlinear projection . . 3
1.3 A strip camera photograph of a man's head [Dav01] 4
1.4 A multi-perspective image for use in image resynthesis [Chu01] 5
1.5 A multi-perspective image for use in image resynthesis [WFH+97] 5
1.6 A child's depiction of a cube, subsequently redrawn [Wil97] 6

2.1 Projection surfaces (top view): (a) discontinuity of ray directions on a projection surface defined by perspective projections and (b) continuity of ray directions on a shared-normal interpolated projection surface 10
2.2 Triangle with arbitrary normal vectors 11
2.3 Triangle with interpolated normal vectors 12
2.4 A Phong shaded cube: (a) normal vectors perpendicular to the faces (b) normals coincident with the cube centre 13
2.5 Ray tracing a trilinear projection with rasterising 14

3.1 Interpolated surface ray intersecting a scene point 15
3.2 A parametric triangle shown at different values of t 17

4.1 A (2,2,2,3) shape configuration: (a) ray trace (b) trilinear projection 34
4.2 A (3,3,3) shape configuration: (a) ray trace (b) trilinear projection 34
4.3 A (4,2,3) shape configuration: (a) ray trace (b) trilinear projection 34
4.4 A (4,5) shape configuration: (a) ray trace (b) trilinear projection 35
4.5 A (6,3) shape configuration: (a) ray trace (b) trilinear projection 35
4.6 A (2,7) shape configuration: (a) ray trace (b) trilinear projection 35
4.7 A (9) shape configuration: (a) ray trace (b) trilinear projection 36
4.8 Example of the error inherent in the linear approximation of curved shapes 37
4.9 Scene triangle tessellated into 25 triangles approximating a (4,5) shape configuration 38
4.10 Scene triangle sampled at 5 extra t levels per shape approximating a (4,5) shape configuration 39
4.11 A shape partially behind the viewing plane 40

5.1 A scene triangle spanning two trilinear projections with a discontinuity . . 45

5.2 A trilinear projection edge swept out into scene space and intersected with a scene triangle . . . 46

6.1 A Distortion-Oriented Display mesh projection surface . . . 52
6.2 A Distortion-Oriented Display projection surface and a cube scene . . . 53
6.3 A perspective projection of a cube . . . 54
6.4 A Distortion-Orientation projection of a cube (a) ray trace (b) trilinear projection . . . 54
6.5 A maze distorted in a cylindrical fashion to show context . . . 55
6.6 A multiple-perspective approach to showing first person detail and side view context . . . 56
6.7 A cube rendered from a surface derived from Figure 6.6 (a) ray trace (b) trilinear projection . . . 57
6.8 An spherical mesh mapping a relation between the scene data and surface . 58
6.9 A cube rendered by a spherical projection surface (a) ray trace (b) trilinear projection . . . 59

7.1 A diagram of a first-hit reflection . . . 61
7.2 Error in an approximation of first-hit reflections . . . 63
7.3 A cube reflected in a sphere . . . 65
7.4 A cube reflected on a sphere: (a) ray traced, (b) 1x1 surface, (c) 2x2 surface, (d) 3x3 surface, (e) 4x4 surface, (f) 5x5 surface . . . 67
7.5 A cube reflected on a plane with perturbed normals, implemented as a 5x5 trilinear projection surface: (a) ray traced (b) trilinear projection . . . 68
7.6 A cube refracted through a plane with spherical normals: (a) ray traced, (b) 1x1 surface, (c) 2x2 surface, (d) 3x3 surface, (e) 4x4 surface, (f) 5x5 surface 69

8.1 Execution time versus resolution for ray tracing different configurations . . 72
8.2 Execution time versus resolution for ray tracing across different complexity scenes . . . 73
8.3 Execution time versus number of scene triangles for ray tracing across different resolutions . . . 73
8.4 Execution time versus resolution for trilinear projecting different configurations 74
8.5 Execution time versus resolution for trilinear projection across different complexity scenes . . . 75
8.6 Execution time versus number of scene triangles for trilinear projection across different resolutions . . . 75
8.7 Relative speedup versus resolution for trilinear projection across different configurations . . . 76
8.8 Relative speedup versus resolution for trilinear projection across different complexity scenes . . . 77
8.9 Relative intensity of the difference mask versus tessellation factor for parametric triangle slicing over different configurations . . . 78
8.10 Relative intensity of the difference mask versus tessellation factor for scene triangle tessellation over different configurations . . . 78

8.11 Relative intensity of the difference mask versus tessellation factor averaged over each configurations for parametric triangle slicing and scene triangle tessellation . . . 79
8.12 Execution time versus tessellation factor averaged over each configurations for parametric triangle slicing and scene triangle tessellation . . . 79
8.13 Execution time versus number of trilinear projections for ray tracing across different resolutions . . . 80
8.14 Execution time versus number of trilinear projections across different resolutions . . . 81
8.15 Execution time versus number of trilinear projections across different resolutions . . . 81
8.16 Relative execution time for clipped and non-clipped trilinear projection versus number of trilinear projections across different complexity scenes . . . 82

9.1 A conventionally rendered set of columns . . . 88
9.2 Columns rendered from a torus surface . . . 89
9.3 A hand-drawn nonlinear projection of a street scene [Gla00] . . . 90
9.4 A cube rendered from a hemisphere surface [VC01b] . . . 91
9.5 Catacaustic of the reflection congruence [Gla99] . . . 92
9.6 A nonlinear projection of an elephant [RB98] . . . 94

List of Tables

4.1 Possible shape configurations 29

8.1 Execution time for rendering examples in this thesis 83

B.1 Vector properties of the trilinear propjection 110

C.1 112
C.2 113

D.1 Ray tracing results over random scene data 117
D.2 Trilinear projection results over random scene data 120
D.3 Trilinear projection with clipping results over random scene data 123
D.4 Ray tracing results on a 2,2,2,3 configuration example 124
D.5 Ray tracing results on a 2,3 configuration example 124
D.6 Ray tracing results on a 3,3,3 configuration example 124
D.7 Ray tracing results on a 4,2,3 configuration example 124
D.8 Ray tracing results on a 4,5 configuration example 124
D.9 Ray tracing results on a 6,3 configuration example 124
D.10 Ray tracing results on a 9 configuration example 125
D.11 Trilinear projection results on a 2,2,2,3 configuration example 126
D.12 Trilinear projection results on a 2,3 configuration example 126
D.13 Trilinear projection results on a 3,3,3 configuration example 126
D.14 Trilinear projection results on a 4,2,3 configuration example 126
D.15 Trilinear projection results on a 4,5 configuration example 126
D.16 Trilinear projection results on a 6,3 configuration example 126
D.17 Trilinear projection results on a 9 configuration example 127
D.18 Trilinear projection scene triangle tessellation results on a 2,2,2,3 configuration example 128
D.19 Trilinear projection scene triangle tessellation results on a 2,3 configuration example 128
D.20 Trilinear projection scene triangle tessellation results on a 3,3,3 configuration example 129
D.21 Trilinear projection scene triangle tessellation results on a 4,2,3 configuration example 129

D.22 Trilinear projection scene triangle tessellation results on a 4,5 configuration example . . . 130
D.23 Trilinear projection scene triangle tessellation results on a 6,3 configuration example . . . 130
D.24 Trilinear projection scene triangle tessellation results on a 9 configuration example . . . 130
D.25 Trilinear projection with parametric triangle slicing results on a 2,2,2,3 configuration example . . . 131
D.26 Trilinear projection with parametric triangle slicing results on a 2,3 configuration example . . . 131
D.27 Trilinear projection with parametric triangle slicing results on a 3,3,3 configuration example . . . 132
D.28 Trilinear projection with parametric triangle slicing results on a 4,2,3 configuration example . . . 132
D.29 Trilinear projection with parametric triangle slicing results on a 4,5 configuration example . . . 133
D.30 Trilinear projection with parametric triangle slicing results on a 6,3 configuration example . . . 133
D.31 Trilinear projection with parametric triangle slicing results on a 9 configuration example . . . 133

List of Listings

3.1 Projecting a scene point onto a parametric triangle 25
4.2 Sorting vertex lists into shapes 28
4.3 Sorting vertex lists into shapes and ordering by connectivity 32
4.4 Sampling with discrete parametric triangle slices 40
4.5 Calculating view-plane intersections 42
5.6 Finding edge intersection t values 47
5.7 Finding clipping points in shapes 48
5.8 Inserting clipping points into shapes 49

Acknowledgements

This work was supported by a Flinders University Research Scholarship for which I am very grateful. None of this would have been possible without the support of a great many people whom I would like to thank. My partner, Fran, for being there when I needed it most, thank you. My family, who supported me in a myriad of ways, from their love down to the practical everyday things. My supervisor, Paul, for his endless advice and enjoyable research discussions. All my fellow research slaves, aka PhD candidates, for their camaraderie, and especially Aaron and Ron for always being there for a chat. To my friends whom I have neglected too much whilst writing this thesis, I apologise.

Chapter 1

Introduction

Geometric projections map multi-dimensional data to a lower dimensional representation. In computer graphics rendering of 3D scenes this dimension reduction is from 3D to 2D. Most projections used in graphics are linear, which means that these projections fall into two categories: perspective and orthographic. Perspective projections simulate the physics of optics; they map data back to a single point in space through a virtual screen, so that distant data is smaller on the virtual screen than nearer data. Perspective projections allow data to be viewed as if through a virtual human eye or camera lens. Orthographic projection maps data to a plane, preserving size over distance. Orthographic projections such as top, side or front views are common in architectural drawing.

Nonlinear projections differ from linear projections in that straight lines in 3D may not be straight lines when projected. Nonlinear projections occur in art, in reflections and refractions on curved surfaces, and in data visualisations. Previous techniques for nonlinear projection have used ray tracing or have relied on distortion of the scene data. This thesis presents new techniques for rendering nonlinear projections in a manner analogous to perspective transformation matrix rendering. The new technique is called a trilinear projection because its basis is a trilinear interpolation. Multiple trilinear projections can be used to represent more complex nonlinear projections while maintaining continuity across sub-projections.

1.1 Nonlinear Projection

Nonlinear projections occur naturally as reflections and refractions on curved objects. The strange and distorted images seen in an amusement park funny mirror are a familiar example

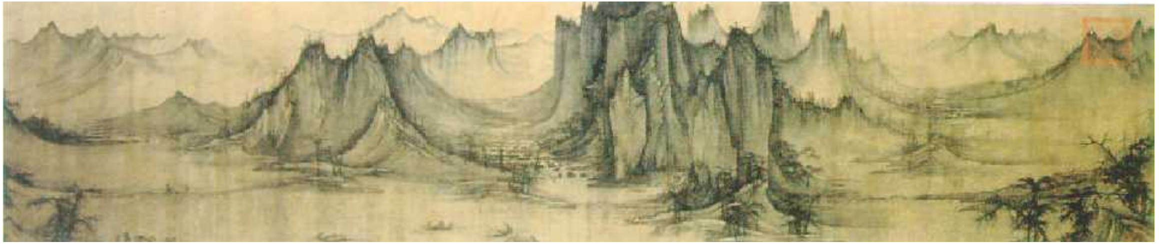


Figure 1.1: “Fishermans Evening Song” by Xu Daoning, circa 11th Century

of the distortion nonlinear projections generate. These images have also been examined in art, photography and computer graphics where they have variously been named cubist images, multi-perspective images, multiple-centre-of-projection images and multi-perspective panoramas.

1.1.1 Artistic Nonlinear Projection

Traditional Chinese landscape paintings frequently contain different foci, or sub-images, which are seamlessly joined. These paintings are similar to the panoramas used for cartoon drawing and image resynthesis, as is discussed in Section 1.1.3. For example, in Figure 1.1 the perspective shifts from left to right, following the path of the stream.

German artist M. C. Escher frequently depicted views with multiple vanishing points, or perspectives. For example, “High and Low” [Esc92] shown in Figure 1.2, has five different vanishing points: top left and right, centre, and bottom left and right. While the automatic generation of an image like this from 3D geometry may not be practical, it illustrates the concept and the aesthetic potential.

1.1.2 Strip Cameras

Strip cameras are widely used in surveillance and mapping. These cameras have a continuous roll of film that slides past a slit as a picture is being taken. The camera may be moved whilst shooting, providing a change in point of view from one section of the film to another. For example, if used from a moving aeroplane a strip camera can capture a long section of curved earth as if it were flat. The technique has also been used for artistic purposes, capturing strange and unusual images, such as in Robert Davidhazy’s work show in Figure 1.3.

Roman *et al* [RGL04] use cross-slit cameras to render seamless multi-perspective

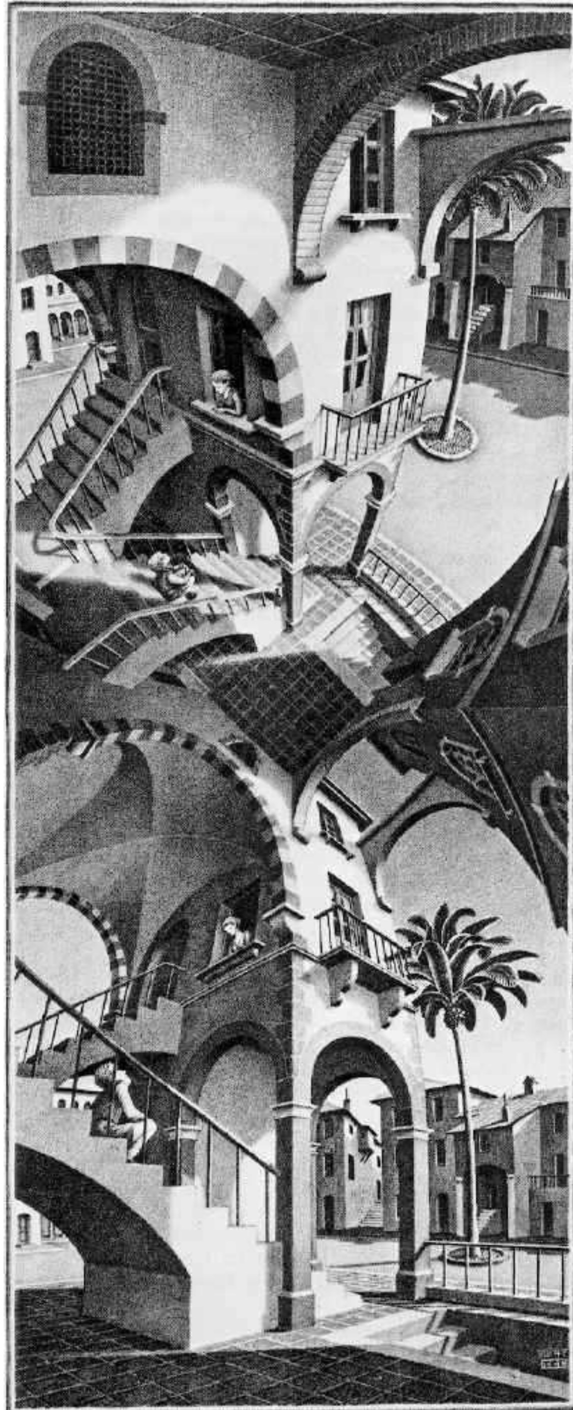


Figure 1.2: “High and Low” by M. C. Escher, an example of a nonlinear projection

Figure 1.3: A strip camera photograph of a man's head [Dav01]

images of urban landscapes. These cameras are similar to virtual strip cameras and are used to re-interpret standard video footage. The moving video footage is indexed by the cross-slit cameras to create a single image that approximates the image a real version of the cross-slit camera would have captured.

1.1.3 Multi-Perspective Images as a Basis for Resynthesis

Hand-drawn and computer-generated panoramas with multiple points of view have been used as the basis of image resynthesis. Cartoon animation from panoramas is an early example of resynthesis that has been adapted to computer-generated images by Wood et al [WFH+97] (see Figure 1.5). Chu and Tai [CT01] use Chinese landscape paintings and computer-generated images as a basis of resynthesis (see Figure 1.4). When a small subsection of the panorama is viewed it approximates a standard single viewpoint. If multiple subsections are taken along a path on the panorama and sequenced into an animation, the effect gives the appearance of motion because the viewpoint shifts continuously in a multi-perspective panorama.

1.1.4 Visualisation with Nonlinear Projection

As a visualisation technique, nonlinear projections can be justified by a more abstract understanding of computer depiction. Durand [Dur02] examined computer depiction more generally as a mapping of scene properties to picture properties. While linear projections are familiar, they may not always capture the relevant properties of the scene succinctly. For example Willats [Wil97] describes a cube drawn by a child (Figure 1.6) that shows the colouring of all sides of the cube in a single image. The effect is reminiscent of Figure 6.9, which shows a nonlinear projection of a cube as seen from the surface of an inward-looking

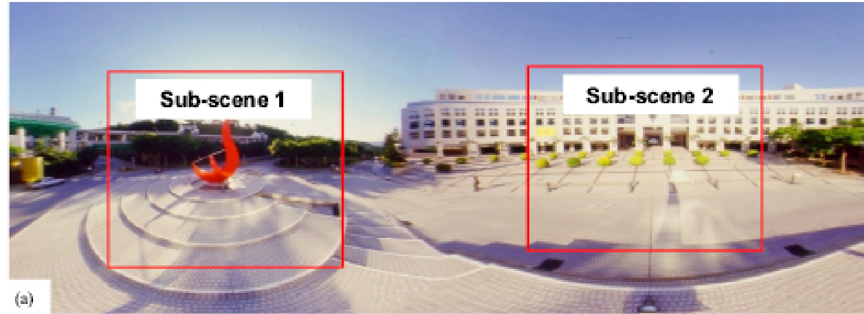


Figure 1.4: A multi-perspective image for use in image resynthesis [Chu01]

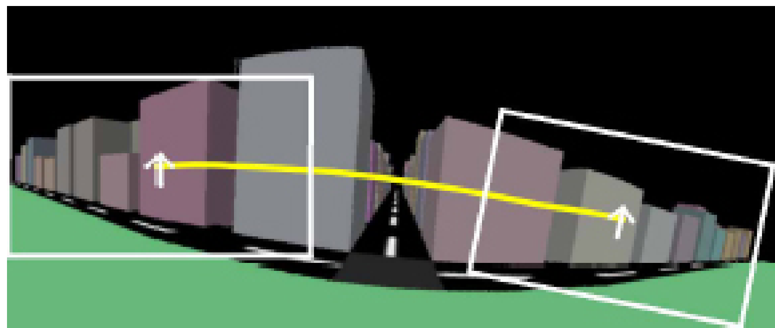


Figure 1.5: A multi-perspective image for use in image resynthesis [WFH+97]

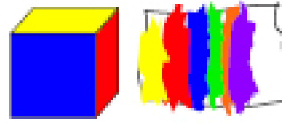


Figure 1.6: A child's depiction of a cube, subsequently redrawn [Wil97]

sphere. All sides of the cube are shown in a single image, similar to the child's depiction.

1.2 Ray tracing and Scanline Rendering

Nonlinear projections have previously been successfully implemented with ray tracing. In contrast, this thesis is fundamentally concerned with a scanline-based alternative to rendering nonlinear projections. Ray tracing and scanline rendering exhibit many fundamental similarities. This is particularly true when tracing primary rays (those which come directly from the viewpoint without reflection or refraction). Consider a point in scene space and its projected equivalent on the image plane. Ray tracing casts a ray from the eye point through the image point to determine the scene point. Scanline rendering projects the scene onto the image plane to determine the image point. Both algorithms determine if a particular image plane point 'sees' a given scene point. Geometrically the algorithms are nearly identical.

Scanline rendering takes advantage of the case where rays are all coincident (they meet at the viewpoint) and the sampling is over a regularly spaced grid (the screen buffer). When these conditions are met, converting a 3D triangle to 2D pixels can be efficiently rendered by computing the projection at only the vertices and then interpolating across the surface of the triangle. Scanline rendering is thus particularly suited to the case where the number of polygons is not large compared with the number of pixels.

Both ray tracing and scanline rendering continue to increase in sophistication and performance, with scanline rendering remaining the common implementation for interactive and non-interactive rendering tasks.

1.3 Thesis Scope

This thesis presents algorithms pertaining to a new form of nonlinear projection called trilinear projection. At its most basic level the rendering method transforms 3D points, lines

or triangles to a 2D representation. However, once transformed many standard graphical features such as lighting and texturing [FDFH90] can be implemented without alteration to the standard algorithms.

The thesis also presents some possible applications of the trilinear projection and explores issues of execution complexity and scalability. However, it does not examine the efficacy of the projection at meeting any production scenario constraints. Performance can vary greatly depending on many factors and the implementation referenced in this thesis only represents a proof of concept.

1.4 Thesis Overview

The remainder of this thesis is organised as follows: Chapter 2 presents trilinear interpolation as a basis for nonlinear projections. The mathematics of trilinear interpolation are detailed. Nonlinear projection surfaces are defined and trilinear interpolation is used to describe a nonlinear projection surface. Ray tracing from a nonlinear projection surface is examined with reference to computational performance.

Chapter 3 details an algorithm to project a 3D point with a trilinear projection. Given a projection triangle whose rays are interpolated across the surface in a trilinear manner, the algorithm computes the rays that intersect a given point. A restatement of the constraints that describe the system leads to a more efficient implementation in certain situations.

Chapter 4 expands the results of the previous chapter to examine the shapes formed by projecting three scene points that define a triangle. The chapter shows that a projected triangle may produce from one to four different shapes of from two to nine vertices. An algorithm to organise these vertices into regular polygons is detailed. Two different tessellation methods to reduce artifacts caused by linear interpolation between vertices are presented. The first method tessellates the scene triangle; the second method samples the scene triangle between vertex solutions. For solutions partially behind the trilinear projection, a view plane clipping algorithm is defined.

Chapter 5 shows how multiple trilinear projections can be joined to form more complex nonlinear projections. Rendering solutions for each trilinear projection are constrained to a triangular section of the screen. Discontinuities arising from naively joining multiple projections are addressed with a clipping algorithm.

Chapter 6 shows how trilinear projections can be used to render nonlinear pro-

jections for visualisation. Visualisation techniques such as distortion-oriented displays and map projections are defined in terms of nonlinear projections. These projections are then used to render an example scene and the results shown.

Chapter 7 details how trilinear projections can be used to approximate reflections and refractions on polygon meshes. Using the vector equations for reflection or refraction and applying them to the normals of the polygon mesh gives a projection surface approximating the reflection or refraction. Simple scenes are demonstrated with different projection surface resolutions. Projection surface resolution increases visual accuracy at the cost of computation time.

Chapter 8 presents an analysis of computation time and visual accuracy for trilinear projection and ray tracing. Parameters such as screen resolution, tessellation factor and projection surface resolution are varied and results averaged over random and specific data sets. These results are discussed and conclusions drawn as to where trilinear projection is likely to be better suited than ray tracing.

Chapter 9 examines previous work in nonlinear projections. Nonlinear projections in art work, visualisations and reflections and refractions and their implementations are reviewed.

Chapter 10 summarises the algorithms presented in this thesis. Further work and directions are detailed. The context of the thesis is analyzed with respect to the results of the work.

Chapter 2

Nonlinear Projection Surfaces

This chapter describes an approach to the problem of defining and rendering from a nonlinear projection. Ray emitting surfaces are introduced as a way of describing projections that extend beyond linear. The triangular mesh is examined as a curved surface approximator and ray emitting surface. Ray tracing from a triangular mesh projection surface is detailed.

2.1 Projection Surfaces

The geometry of standard perspective or orthographic projections when viewed from a ray tracing orthodoxy defines the way in which initial or eye rays are generated. In perspective projection eye rays emanate from a point, and in orthographic projection from a plane. The direction of the eye rays in perspective projection is such that they diverge from the eye point, and in orthographic projection they are parallel. These projections can be seen as linear, because a line in scene space is drawn by a set of rays that define a line in screen space. In this thesis we are concerned with rendering projections that are not linear, so that lines in scene space may map to curves in screen space.

A nonlinear projection can be defined by the nature of its geometry, as seen from a ray emitting basis. The locus of points derived from a projection's ray starting points can define a surface. The directions of these emitted rays can be defined as functions across the surface of starting points. Equally, the mapping between surface and screen positions can be described as a function on the surface of starting points. This approach is described by Löffelmann and Gröller [Löf95], who calls the collection of such surfaces and the functions upon them 'Extended Cameras'. Glassner [Gla00] takes a similar approach and defines 'cu-

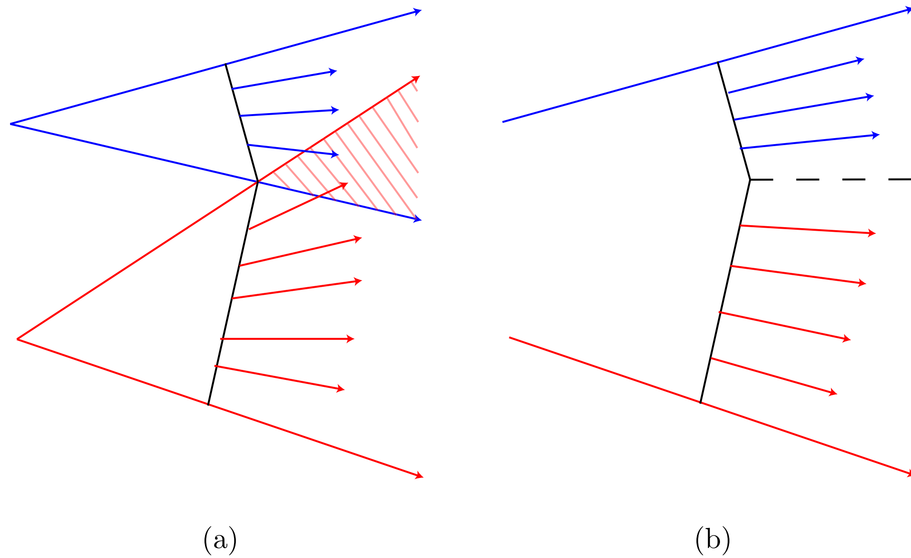


Figure 2.1: Projection surfaces (top view): (a) discontinuity of ray directions on a projection surface defined by perspective projections and (b) continuity of ray directions on a shared-normal interpolated projection surface

bist cameras', which consist of two NURBS (Non-Uniform Rational B-line Splines) surfaces, one for eye ray positions and another for ray directions. Viewing these cameras as surfaces means that the problem of finding sufficiently expressive and efficient nonlinear ray emitting surfaces is analogous to that of finding a representation of a traditional curved surface.

2.2 Mesh Surfaces and Surface Continuity

Complex surfaces can be represented as a mesh of simpler surfaces, which are generally easier to render and manipulate, allowing for rendering performance and modeling flexibility. A mesh of triangles is commonly used to approximate surfaces for display in computer graphics because the triangle is an easily rendered surface. If a triangular mesh is used as a ray emitting surface, this would sufficiently express the starting points of the rays but not necessarily their direction. Each triangular sub-surface could be treated as a perspective projection surface, so that each triangle effectively has an eye point attached to it. The ray directions on a triangular patch converge to the relevant eye point, but start on the triangle's plane.

Figure 2.1 (a) shows a top-down view of a two-section perspective projection. An

unavoidable discontinuity occurs because the rays at the intersection point trace back to two separate eye points. An image rendered by this projection surface would sharply change from one sub section to another. Continuity can be maintained if the ray directions in some way smoothly change from one sub-surface to the next. In Figure 2.1 (b) a diagram of a continuous viewing surface is presented, where the segments share a ray path on their join. Section 2.3 details how such a surface can be constructed.

2.3 Representing a Curved Surface with a Trilinear Interpolated Triangle

In computer graphic scenes composed of triangles the illusion of smoothly curving facets is generated by arbitrary normals that are defined at the vertices of the facets. These normals are not perpendicular to the triangle, but rather to the curved surface that the triangle is approximating. Parameters that rely on the normal of a surface, such as shading, can be calculated on the vertex normals and interpolated across the face of the triangle. A triangle with arbitrary normal vectors defined at its vertices is shown in Figure 2.3.

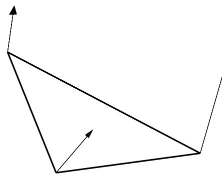


Figure 2.2: Triangle with arbitrary normal vectors

With interpolated normals a mesh with smoothly changing normal directions can be constructed. Using these normals as the rays of a projection surface forms the basis of this thesis. Each single triangular section of the mesh is a trilinear projection.

2.3.1 Interpolation

Interpolation across a triangle can be defined with Barycentric coordinates. A Barycentric point is the weighted sum of the triangle's vertices with weights α_1, α_2 and α_3. This is

shown in Equation 2.1.

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 \tag{2.1}$$

It should also be noted that α_1, α_2 and α_3 must sum to 1. Therefore, one of the three weights is unnecessary.

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{2.2}$$

$$\alpha_1 = 1 - \alpha_2 - \alpha_3 \tag{2.3}$$

Traditionally, in computer graphics α_2 and α_3 are named u and v. So, given a triangle with vertices p_1, p_2 and p_3 and normals n_1, n_2 and n_3, a point and normal on that triangle defined by the parameters u and v is shown in the following equations:

$$tripoint\,(u, v) := (1 - u - v)\,p_1 + up_2 + vp_3 \tag{2.4}$$

$$trinorm\,(u, v) := (1 - u - v)\,n_1 + un_2 + vn_3 \tag{2.5}$$

A triangle with normal vectors interpolated across its surface is shown in Figure 2.3.

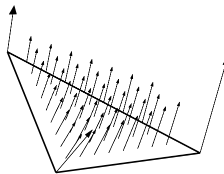


Figure 2.3: Triangle with interpolated normal vectors

2.3.2 Phong Shading

Phong shading uses interpolated normals, computed according to Equation 2.5, to calculate smoothly changing shading values. Figure 2.4 (a) shows a flat shaded cube. The shading values are calculated according to the actual normal to the cube's surface. Figure 2.4 (b) shows a cube shaded according to the Phong algorithm where the normals at each vertex are pointing directly away from the center of the cube. With interpolated normals the shading gives the cube the appearance of being curved, though with an unchanged silhouette.

(a) (b)

Figure 2.4: A Phong shaded cube: (a) normal vectors perpendicular to the faces (b) normals coincident with the cube centre

2.4 Ray Tracing with a Trilinear Projection

Ray tracing with a trilinear projection differs from conventional ray tracing only in that the eye rays do not come from a single point. For each pixel a ray is generated by converting x and y values into u and v and substituting these values into Equations 2.4 and 2.5. With a triangular mesh, each trilinear projection maps to a triangular region of screen space.

Ray tracing in this manner can be accelerated by the application of triangle rasterising algorithms. Triangle rasterising converts a triangle defined by three points into a series of pixels. One way of doing this is to take a horizontal line of pixels at a time. The starting and ending pixels on the horizontal lines increment or decrement by a constant amount between lines. Additionally, properties such as the position on the surface of the triangle change by a constant amount from pixel to pixel. Properties derived from the position on the surface in a linear manner, such as the ray position and direction, can also be reduced to a constant step each pixel.

Figure 2.5 shows a trilinear projection with a section converted to pixels. The interpolated surface rays change by Δn each pixel.

2.5 Summary

While ray tracing provides an obvious way to implement nonlinear projections such as the trilinear projection, ray tracing is not often used for interactive and animated computer graphics because it is expensive to compute. A scanline based method for trilinear projection is potentially more efficient, but requires that the problem of mapping a scene point to

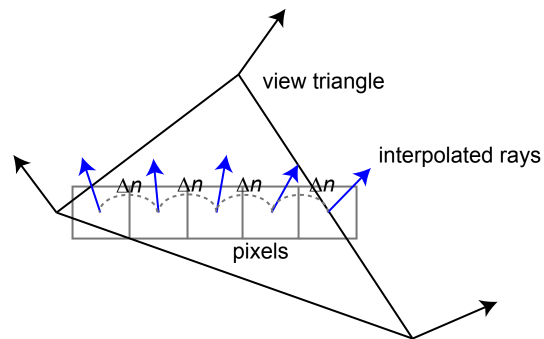


Figure 2.5: Ray tracing a trilinear projection with rasterising

the projection surface must be solved. Chapter 3 examines this problem and presents an analytical solution.

Chapter 3

Projecting a Point with a Trilinear Projection

The key question in projecting to a curved surface is as follows: given a point in the scene, where will it be imaged by the surface? Note that, depending on the geometry of the surface, there may be several images of a given point. If the surface is composed of trilinear triangles, the problem becomes how to map a scene point to a trilinear projection. We wish to find which ray or rays emanating from the triangle intersects the scene point. The problem is illustrated in Figure 3.1.

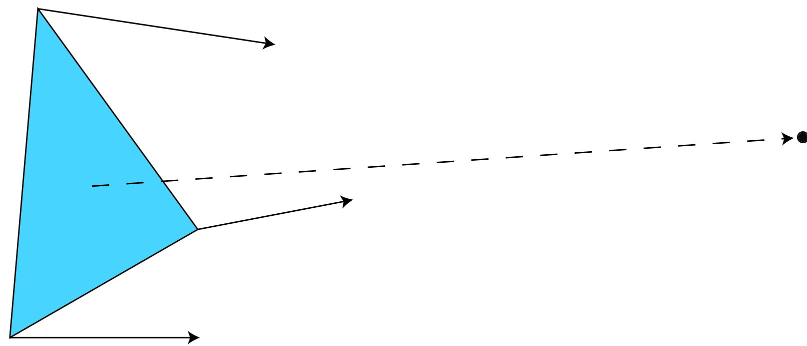


Figure 3.1: Interpolated surface ray intersecting a scene point

3.1 Projecting a Point with a Trilinear Projection

A ray can be defined parametrically by a point, p and a normal n:

$$ray\,(t) \;:=\; p + tn \tag{3.1}$$

Taking the earlier definitions of a point and normal on the surface of a triangle in Equations 2.4 and 2.5, a ray on the surface of the triangle defined by u and v becomes:

$$\begin{aligned} ray\,(u, v, t) &:= tripoint\,(u, v) + t.trinorm\,(u, v) && (3.2)\\ &= (1 - u - v)\,p_1 + up_2 + vp_3 + t\,((1 - u - v)\,n_1 + un_2 + vn_3) && (3.3)\end{aligned}$$

So for a scene point p_s:

$$\begin{aligned} p_s &= ray\,(u, v, t) && (3.4)\\ &= (1 - u - v)\,p_1 + up_2 + vp_3 + t\,((1 - u - v)\,n_1 + un_2 + vn_3) && (3.5)\end{aligned}$$

Solving for u, v and t in 3D means a system of three equations and three unknowns. Because the t and u, v terms are interdependent it is not a linear system. Analytical solutions often do not exist for solving nonlinear systems, and numerical analysis must be used.

3.2 Treating the Trilinear Projection as a Parametric Triangle

To analytically solve this problem, instead of representing the surface as a set of rays, we represent it as a parametric triangle. Each vertex has a point and a normal associated with it. If we treat these as rays we can extend along them according to the parameter t giving three new points, which form the vertices of a triangle as shown in Figure 3.2. The three vertices of the parametric triangle are defined by the equations:

$$\begin{aligned} r_1 &:= p_1 + tn_1 && (3.6)\\ r_2 &:= p_2 + tn_2 && (3.7)\\ r_3 &:= p_3 + tn_3 && (3.8)\end{aligned}$$

A barycentrically defined point on the parametric triangle is:

$$\begin{aligned} paratri\,(u, v, t) &:= (1 - u - v)\,r_1 + ur_2 + vr_3 && (3.9)\\ &= (1 - u - v)\,(p_1 + tn_1) + u\,(p_2 + tn_2) + v\,(p_3 + tn_3) && (3.10)\end{aligned}$$

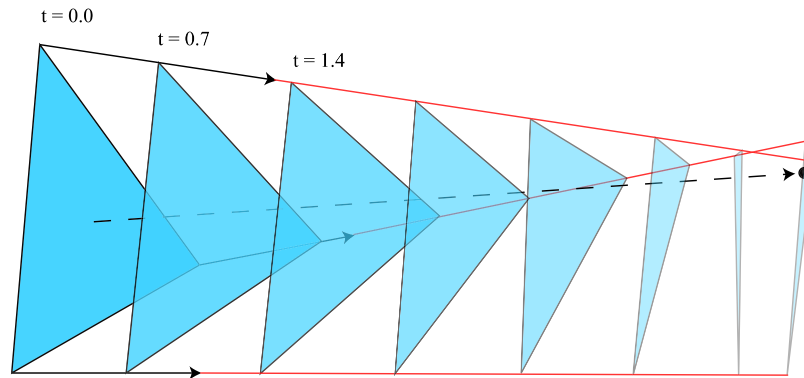


Figure 3.2: A parametric triangle shown at different values of t

The task of projecting a scene point now becomes that of finding a point $paratri\,(u, v, t)$ that coincides with the scene point. The u, v and t satisfying this constraint are the same as those satisfying $p_s = ray\,(u, v, t)$ because the two equations are simply isomorphs, as shown by expanding and then collecting the terms in Equation 3.10:

$$\begin{aligned} paratri\,(u, v, t) &= p_1 + tn_1 - up_1 - utn_1 - vp_1 - vtn_1 + up_2 + utn_2 + vp_3 + vtn_3 \\ &= (n_1 - un_1 - vn_1 + un_2 + vn_3)\,t + p_1 - up_1 - vp_1 + up_2 + vp_3 \\ &= (1 - u - v)\,p_1 + up_2 + vp_3 + t\,((1 - u - v)\,n_1 + un_2 + vn_3) \\ &= ray\,(u, v, t) \end{aligned} \tag{3.11}$$

The advantage in representing the triangle as a parametric triangle is that the parameter t can be determined independently of u and v. First, for the parametric vertices defined in Equation 3.8, find the values of t such that the vertices and the scene point p_s are coplanar, then solve for u and v in the plane of the parametric triangle.

3.3 Determining the Coplanarity of the Parametric Triangle and the Scene Point

Any four points can be considered a tetrahedron; four coplanar points form a tetrahedron whose volume is 0. For a tetrahedron defined by four points A, B, C and D the volume of

the tetrahedron is:

$$volume \quad := \quad \left| \frac{1}{4} \mathbf{AB} \bullet (\mathbf{AC} \times \mathbf{AD}) \right| \tag{3.12}$$

This equation can also be expressed as the magnitude of the determinant of the matrix containing the three vectors.

$$volume \quad := \quad \left| \frac{1}{4} \det \begin{bmatrix} \mathbf{AB} \\ \mathbf{AC} \\ \mathbf{AD} \end{bmatrix} \right| \tag{3.13}$$

Composing the three vectors from the parametric vertices defined in Equation 3.8 with the scene point p_s and substituting into Equation 3.13 gives the volume of the tetrahedron defined by those four points. When these points are coplanar this volume is 0. Removing the unnecessary constant and magnitude gives:

$$\begin{vmatrix} p_1 + tn_1 - p_s \\ p_2 + tn_2 - p_s \\ p_3 + tn_3 - p_s \end{vmatrix} = 0 \tag{3.14}$$

When fully expanded for 3D the equation becomes:

$$\begin{vmatrix} p_{1x} + tn_{1x} - p_{sx} & p_{1y} + tn_{1y} - p_{sy} & p_{1z} + tn_{1z} - p_{sz} \\ p_{2x} + tn_{2x} - p_{sx} & p_{2y} + tn_{2y} - p_{sy} & p_{2z} + tn_{2z} - p_{sz} \\ p_{3x} + tn_{3x} - p_{sx} & p_{3y} + tn_{3y} - p_{sy} & p_{3z} + tn_{1z} - p_{sz} \end{vmatrix} = 0 \tag{3.15}$$

This can be expanded, giving a cubic polynomial in terms of t. The full coefficients for this cubic are shown in Appendix A.1. Either one or three real solutions for t exist, and each value of t defines a potentially different triangle. The roots of this cubic polynomial can be found analytically. Alternatively, numerical techniques such as Newton's method can be used.

3.4 Barycentric Coordinate Conversion

For each triangle coplanar with p_s, values of u and v can be computed by solving the system of three linear equations defined by $tripoint\,(u, v)$ in Equation 2.4, with one equation for each dimension x, y and z. As there are only two unknowns the system can be reduced to a 2x2 linear equation system and solved using Cramer's Rule. This rule states that

for a system of n equations with n unknowns described by $AX = B$ the solutions are $x_{1..n} = \frac{det(A_i)}{det(A)}$ where A_i is formed by replacing the ith column in A with B.

Begin by calculating for the particular value of t the vectors between the triangle vertices $p_{1..3}$ and the scene point p_s:

$$E_1 := p_2 - p_1 \tag{3.16}$$

$$E_2 := p_3 - p_1 \tag{3.17}$$

$$E_3 := p_s - p_1 \tag{3.18}$$

Then calculate the scalar values that are the dot products of vectors E_1 and E_2 by themselves and each other:

$$d_{1,1} := E1 \cdot E1 \tag{3.19}$$

$$d_{2,2} := E2 \cdot E2 \tag{3.20}$$

$$d_{1,2} := E1 \cdot E2 \tag{3.21}$$

Next, calculate the value d, which is equivalent to $\frac{1}{detA}$ in the description of Cramer's Rule above. Its value is important in determining the validity of the solution.

$$d := d_{1,1}d_{2,2} - d_{1,2}d_{1,2} \tag{3.22}$$

Finally, with the help of intermediate scalar values $a_{1..2}$, find the equivalents of $det(A_i)$ for u and v. These are divided by d giving the solutions for u and v.

$$a_1 := E3 \cdot E1 \tag{3.23}$$

$$a_2 := E3 \cdot E2 \tag{3.24}$$

$$u = \frac{a_1 d_{2,2} - a_2 d_{1,2}}{d} \tag{3.25}$$

$$v = \frac{a_2 d_{1,1} - a_1 d_{1,2}}{d} \tag{3.26}$$

If the value of d is close to zero then the triangle determined by $p_{1..3}$ is degenerate, indicating that the points $p_{1..3}$ are collinear or coincident. If $p_{1..3}$ are collinear then the value of t is a valid solution if point p_s lies upon this line. If $p_{1..3}$ are coincident and p_s is sufficiently close to that point, the value of t defines a valid solution. If p_s does not lie on the line or point determined by $p_{1..3}$ the solution is not valid. A triangle is determined to be degenerate if the value of d is below an arbitrary threshold. Automatically calculating an appropriate threshold based upon a particular scene and trilinear projection is left as future work (see Section 10.3).

3.5 Multiple Solutions

There are up to three different solutions for u and v depending on the number of solutions for t. Not all these solutions fit with the higher level interpretation of the task. The solutions may be for negative values of t, meaning that the scene point p_s is behind the surface triangle. Also, the values of u and v may be such that the scene point p_s does not lie within the surface triangle. Interior values for u and v occur when u, v and $u + v$ all lie between 0 and 1:

$$0 \leq u, v, (u + v) \leq 1 \tag{3.27}$$

Exterior value solutions can still aid the rendering process if they are part of a larger scene primitive. Chapter 4 describes how a triangle scene primitive may be rendered. Multiple values of u, v and t for a single point may all be interior and imaged by the trilinear projection. For example, reflections on concave surfaces may image a scene point multiple times. Therefore, a nonlinear projection technique must find and render multiple solutions to be accurate.

3.6 Precalculating Partial Coefficient Values

Computing the coefficients of Equation 3.15 involves substantial calculation not directly dependent on the scene point. These calculations depend only upon the values of the parametric triangle itself and therefore can be reused across scene points. This is useful because in a normal scene situation there are many scene points that need to be projected by each parametric triangle. The most common drawing primitive, the triangle, is comprised of three such scene points. The minor additional memory requirements of storing these values is small in comparison to the reduction in computation.

In Equation 3.13 the volume of a tetrahedron is defined as the determinant of three vectors formed from the four points. This can equally be expressed as the four by four determinant shown in Equation 3.28.

$$\begin{vmatrix} p_{sx} & p_{sy} & p_{sz} & 1 \\ p_{1x} + tn_{1x} & p_{1y} + tn_{1y} & p_{1z} + tn_{1z} & 1 \\ p_{2x} + tn_{2x} & p_{2y} + tn_{2y} & p_{2z} + tn_{2z} & 1 \\ p_{3x} + tn_{3x} & p_{3y} + tn_{3y} & p_{3z} + tn_{1z} & 1 \end{vmatrix} = 0 \tag{3.28}$$

The determinant in Equation 3.28 can be expanded to Equation 3.29 where $E_{1..4}$ are each three by three determinants.

$$p_{sx}E_1 - p_{sy}E_2 + p_{sz}E_3 - E_4 = 0 \tag{3.29}$$

Determinants $E_{1..3}$ are quadratics in t and E_4 is a cubic. This means the coefficient of t^3 depends only upon properties of the parametric triangle and not the scene point. Further, the other coefficients of the cubic defined by the full expansion of Equation 3.28 depend on the scene point in a useful way. Equations $E_{1..4}$ are defined by the general cubic equation, Equation 3.30, where $A_{1..3} = 0$

$$E_i = \quad A_i t^3 + B_i t^2 + C_i t + D_i, \quad i = 1, 2, 3, 4 \tag{3.30}$$

If the coefficients for the cubic equation defined by Equation 3.28 are represented by a, b, c and d, where $at^3 + bt^2 + ct + d = 0$, then the relationship between the coefficients and the properties $(A, B, C, D)_{1..4}$ can be described as:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & A4 \\ B1 & B2 & B3 & B4 \\ C1 & C2 & C3 & C4 \\ D1 & D2 & D3 & D4 \end{bmatrix} \begin{bmatrix} p_{sx} \\ p_{sy} \\ p_{sz} \\ 1 \end{bmatrix} \tag{3.31}$$

All properties $(A, B, C, D)_{1..4}$ can be calculated independently of scene points. The full derivations of $(A, B, C, D)_{1..4}$ are provided in Appendix A.3. Using these pre-calculated partial values allows for faster calculation across multiple scene points.

3.7 Parametric Triangle Sides and Containment

When rendering a scene with many scene points it may improve performance to filter out those points without interior solutions. The earlier this can be determined, the greater the performance benefit. When culling scenes points in a linear projection, standard algorithms test a point by substituting it into the implicit equations of the planes that bound a desired region. The sign of the result after substitution defines which side of the plane the point lies on. If the point lies on the correct side of all bounding planes then it is inside; otherwise it is outside.

Each edge of the trilinear projection sweeps out a surface in space. These surfaces are the bounding surfaces. If a scene point is not on the correct side of the three edge

surfaces it will not have any interior solutions. If the scene point is not part of a larger scene primitive, then it will not be imaged by the trilinear projection and can be culled. Culling by testing against the bounding edge surfaces incurs an additional cost, but that cost is less than fully projecting the scene point. If a sufficient number of scene points are culled performance will improve.

The surface defined by each edge of the trilinear projection is a ruled surface because it is traced out by the motion of a line in space. Equation 3.32 shows the general parametric equation of a surface $s(u, v)$ defined by the interpolation between four points $P_{0..1,0..1}$ by the parameters u and v.

$$s(u, v) = (1 - u)(1 - v)P_{0,0} + (1 - u)vP_{0,1} + u(1 - v)P_{1,0} + uvP_{1,1} \tag{3.32}$$

The four points $P_{0..1,0..1}$ define a skew quadrilateral. In the case of the trilinear projection these four points map to the vertices and normals in the following way:

$$P_{0,0} = p_i \tag{3.33}$$

$$P_{0,1} = p_j \tag{3.34}$$

$$P_{1,0} = p_i + n_i \tag{3.35}$$

$$P_{1,1} = p_j + n_j \tag{3.36}$$

$$where : i, j = \{1, 2, 3\}, i \neq j$$

To use this surface as a bounding surface it must be implicitized and the parameters removed. To make implicitization easier the four points $P_{0..1,0..1}$ can be affinely transformed without loss of generality. Transforming the points in this way means that when testing a point against the surfaces the point must also be transformed. Equation 3.37 shows four points transformed, by rotation, scaling and translation, so that the equations simplify as much as possible.

$$\begin{aligned} P'_{0,0} &= (0, 0, 0) \\ P'_{0,1} &= (0, 0, 1) \\ P'_{1,0} &= (0, 1, A) \\ P'_{1,1} &= (1, B, C) \end{aligned} \tag{3.37}$$

The scalar values A, B and C can be determined by:

1. translating $P_{0..1,0..1}$ by $-P_{0,0}$,

2. rotating $P_{0..1,0..1}$ so that $P_{0,1}$ aligns with the Z-axis,

3. scaling $P_{0..1,0..1}$ along the Z-axis such that $P_{z_{0,1}} = 1$,

4. rotating $P_{0..1,0..1}$ around the Z-axis such that $P_{x_{1,0}} = 0$,

5. scaling along the Y-axis and X-axis such that $P_{y_{1,0}} = 1$ and $P_{x_{1,1}} = 1$.

The surface s is defined for the three dimensions x,y and z. Replacing $s(u, v)$ by these dimensions gives three equations similar on the right hand-side to Equation 3.32:

$$x = (1-u)(1-v)P_{x_{0,0}} + (1-u)vP_{x_{0,1}} + u(1-v)P_{x_{1,0}} + uvP_{x_{1,1}} \tag{3.38}$$

$$y = (1-u)(1-v)P_{y_{0,0}} + (1-u)vP_{y_{0,1}} + u(1-v)P_{y_{1,0}} + uvP_{y_{1,1}} \tag{3.39}$$

$$z = (1-u)(1-v)P_{z_{0,0}} + (1-u)vP_{z_{0,1}} + u(1-v)P_{z_{1,0}} + uvP_{z_{1,1}} \tag{3.40}$$

Values for $P'_{0..1,0..1}$ from Equation 3.37 are substituted to simplify the equations.

$$\begin{aligned} x &= uv \\ y &= u - uv + uvB \\ z &= (1-u)v + u(1-v)A + uvC \end{aligned} \tag{3.41}$$

Finally, substitution and algebraic transformation allows the values of u and v to be eliminated, leaving an equation in terms of x, y and z:

$$z = \frac{x}{y + x - xB} - x(1 + A) + A(y + x - xB) + xC \tag{3.42}$$

Finally this gives an equation for the surface purely in terms of x, y and z as shown in Equation 3.43. This is the implicit equation of the surface defined by an edge of the parametric triangle.

$$\begin{aligned} &(A(B-2) - C - 1)x^2 - Ay^2 + (A(B-3) - C - 1)xy + zx + zy + \\ &\qquad (B(1 - A(B+2) + C) - 1)x + ABy - Bz = 0 \end{aligned} \tag{3.43}$$

Transforming a scene point and substituting its x, y and z into the left hand-side of Equation 3.43 gives a result, the sign of which defines on which side of the surface that point lies. If this is done for each side of the trilinear projection then it can be determined if the scene point has any interior values without fully projecting it.

3.8 Algorithm for Projecting a Scene Point

The algorithm for projecting a scene point, using the techniques described in this chapter, is given in Listing 3.1. The function `uvtSolutions` returns all solutions for a scene point `p` projected by the trilinear projection `tp`.

First the function `tSolutions` calculates the roots of the cubic defined by coefficients `a`, `b`, `c` and `d` which are the values of `t` at which the scene point is coplanar to the parametric triangle. The function `computeCoefficients` returns the values of `a`, `b`, `c` and `d` by using either the equations in Equation A.1 or the optimised version in Equation 3.31. The roots are solved using `analyticCubicSolveReal`, which returns the real-valued roots of the cubic.

Each value of `t` generates a solution for `u` and `v`. These solutions are calculated by `uvFromT` which takes a scene point `p` and a `t` solution and calculates the barycentric coordinates of `p` at `t`. The function `project` returns a triangle defined by the vertices `tp` using Equations 3.6, 3.7 and 3.8. This triangle, `tri`, and the scene point `p` is passed to `toBarycentricCoords`, a function which calculates barycentric coordinates according to the equations in Section 3.4. If the triangle `tri` is degenerate and `t` does not give a valid solution the returned value is marked as invalid.

Each potential solution, `vertex`, is checked by the function `isValidSolution` which determines is the solution has been marked valid or invalid. If it is a valid solution it is appended to the list `vertexlist` which holds the current set of valid solutions. When each solution has been processed, `uvtSolutions` returns `vertexlist` which holds the list of valid solutions.

```
Vertex uvFromT(Point p, TrilinearProjection tp, Real t) {
  Triangle tri = project(tp,t)
  return toBarycentricCoords(p,tri)
}

List tSolutions(TrilinearProjection tp, Point p) {
  {a,b,c,d} = computeCoefficients(tp,p)
  return analyticCubicSolveReal( a, b, c, d )
}

List uvtSolutions(TrilinearProjection tp, Point p) {
  vertexlist = {}
  foreach t in tSolutions(tp,p) {
    vertex = uvFromT(p,tp,t)
    if isValidSolution(vertex) {
```

```
        vertexlist.append(vertex)
      }
    }
    return vertexlist
  }
```

Listing 3.1: Projecting a scene point onto a parametric triangle

3.9 Summary

This chapter describes the equality of surfaces rays and scene points that defines the trilinear projection. To solve this equality analytically, a geometric interpretation of the trilinear projection as a parametric triangle is proposed. This interpretation allows for an analytical solution to the projection problem as a cubic in terms of t, the parameter of the parametric triangle. The location on the trilinear projection that images the scene point can be found by calculating the barycentric coordinates of a scene point in the parametric triangle at t.

To improve performance in the case where a trilinear projection projects many scene points, a reformulation of the cubic in t is detailed and a scene point culling algorithm is defined. The reformulation of the cubic at the heart of the projection allows for partial solutions to be cached, improving performance across multiple scene points. A culling algorithm, based on an implicit equation for the trilinear projection's edges, allows for points that will not be imaged within the bounds of a trilinear projection to be culled.

For completeness, pseudo-code detailing the steps in projecting a scene point with a trilinear projection is provided. Projecting a scene point is an important step in being able to project more complicated primitives.

Chapter 4

Projecting a Triangle with a Trilinear Projection

Apart from projecting the vertices, rendering 3D polygons with linear projection is very similar to rendering 2D polygons. Each vertex of the polygon is projected and the connectivity of the projected vertices is maintained in post-projection space. This means that the projected vertices are linked as before and the 2D post-projection polygon can be drawn using standard 2D polygon-drawing algorithms.

In linear projective rendering of polygons, vertices are translated to screen coordinates with edges and surface data interpolated between the vertices. In a nonlinear projection a linear interpolation between vertices for edges and surface data is no longer accurate. Furthermore, each vertex in the polygon may map to more than one projected vertex, which means that connectivity is not directly transferable.

This chapter outlines how connectivity can be determined for post-projection vertices. Once connected appropriately, these polygons can be rendered with standard 2D polygon-rendering algorithms. Linear interpolation between the vertices is not necessarily accurate so tessellation methods to improve accuracy are presented.

4.1 Drawing a Scene Triangle

Each of the vertices of a scene triangle can be separately projected with the trilinear projection generating either one or three solutions for each vertex. The manner in which the projected vertices are connected can be understood by considering the sweep of the parametric triangle intersected with the scene triangle. At every value of t the parametric triangle

is a standard triangle in scene space. The intersection of this triangle's plane with the scene triangle's plane forms a line. As the value of t changes the intersection line traverses the plane of the scene triangle. The motion of this intersection line is continuous because t is continuous, except where the intersection line is undefined because the scene triangle and parametric triangle are parallel or coplanar.

A straight line intersects at most two sides of a triangle. Furthermore, the locus of a continuously defined line must intersect a vertex on the end of an edge before intersecting the edge itself. The order, from smallest to largest t value, in which projected scene triangle vertices are intersected determines the connectivity of those vertices. Initially the line of intersection crosses the two edges connected with the vertex projected by the smallest t value. Each vertex thereafter toggles which edges are being intersected by the parametric triangle. When all edges are toggled off the line of intersection is no longer traversing the scene triangle and the projected shape is complete.

For strings of vertices, in order of t value, toggling the edge states reveals which vertices group together to form a shape. Even though the scene triangle has three vertices, it does not necessarily project shapes which have three vertices. Moreover, a single scene triangle can produce up to four shapes when projected with a parametric triangle. This particular case happens when the three scene vertices produce three t solutions each, and the nine projected vertices form three two-vertex shapes and one three-vertex shape.

4.2 Determining Shapes

Determining projected shapes for a given scene triangle is a matter of iterating through the projected vertices, in order of t value, and keeping track of which edges of the scene triangle are being intersected. For each scene triangle vertex v_i where $i = 1, 2, 3$, define a tuple expressing which edges that vertex connects to. The tuple corresponds in order to the edges $\langle \overline{v_1v_2}, \overline{v_1v_3}, \overline{v_2v_3} \rangle$ so that for v_1 the tuple, $e(v_1)$, is $\langle 1, 1, 0 \rangle$. Also define a state tuple, `s` that represents which edges are intersected. The state tuple is initialised to $\langle 0, 0, 0 \rangle$ and modified at each iteration through the vertices. Whenever the state becomes $\langle 0, 0, 0 \rangle$ a new shape is started.

If `uvt` is the input list of vertices, sorted in ascending order of t, `L` is the resulting list of shapes, and each element of `L` is a list of vertices in a particular shape, then Listing 4.2 shows code that determines the shapes.

```
ListofLists sortVertices(List uvt)
  e = {<1,1,0>,<1,0,1>,<0,1,1>}
```

```
    s = <0,0,0>
    L = {}
    S = {}

    for each v in uvt {
      if s == <0,0,0> and S != {} {
        L.append(S)
        S = {}
      }

      S.append(v)
      s = xor(s,e[v.sceneVertex()])
    }

    return L
  }
```

Listing 4.2: Sorting vertex lists into shapes

Each vertex in `uvt` stores the scene vertex it corresponds to and the u, v and t are associated with this instance. The list `e` expresses the edges connected to each scene vertex. The first scene triangle vertex is connected to the two first edges so its state is `<1,1,0>`. The function `xor` returns a new list which is the exclusive or of its parameters.

Because of the manner in which the vertex list is generated, the length of the list and the order of vertices are constrained to only certain combinations. Specifically, each vertex will appear either one or three times, which means that the list can contain 3, 5, 7 or 9 vertices. Furthermore, not all vertex lists form possible shapes because the vertex must toggle all edge crossings off to be complete. Table 4.1 shows how various complete shapes can arise from combinations of projected vertices.

4.3 Drawing Shapes

The algorithm in Listing 4.2 is not sufficient to draw the projected shapes as polygons because it does not fully determine the order in which the vertices are connected. However the connectivity can be determined as an extension to the algorithm. Once ordered correctly, the shape can be drawn as a polygon using a graphics API. Occlusion is determined by the t value, which is in effect a post-projection depth value.

Whilst iterating through the vertices, when the state is not `<0,0,0>` two edges will be active. The next vertex will either be joined to one of those edges or, if it is the

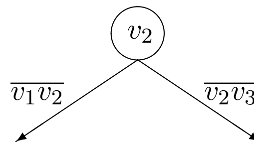
Number of Vertices	Configuration
1	Not possible
2	A single repeated vertex
3	One of each of the vertices
4	Two repeated scene vertices
5	One scene vertex tripled and the other two vertices
6	Three doubled vertices
7	Two tripled vertices and the other single
8	Not possible
9	Each vertex tripled

Table 4.1: Possible shape configurations

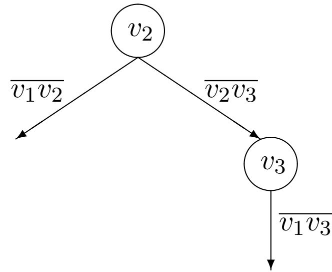
last vertex in a shape, to both. Maintaining two separate lists of vertices, one for each edge within a shape, allows each vertex to be appended after the one it is connected to. These two lists can be named `left` and `right`. Both of these lists correspond to an edge at any point in the process, and this determines upon which list vertices are appended. Once a shape has been determined, one list can be reversed and appended to the end of the other.

For each step of the process two list designators `leftA` and `rightA` are maintained. If the current vertex equals `leftA` or `rightA` it is added to the corresponding list, if it equals neither the new vertex finishes the shape and it is placed in `left` by default. If `leftA` or `rightA` equals the current vertex then the other designator is updated to the vertex not previously contained by `leftA` and `rightA`.

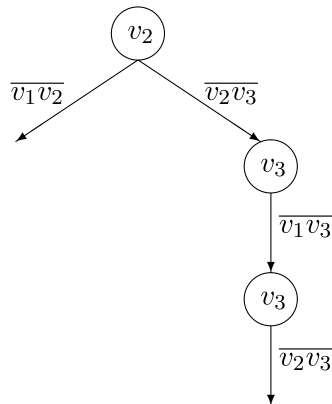
For example, for a list of vertices $V = \langle v_2, v_3, v_3, v_1, v_3 \rangle$ which form a five-vertex shape the following diagrams show how these vertices are formed into a polygon.



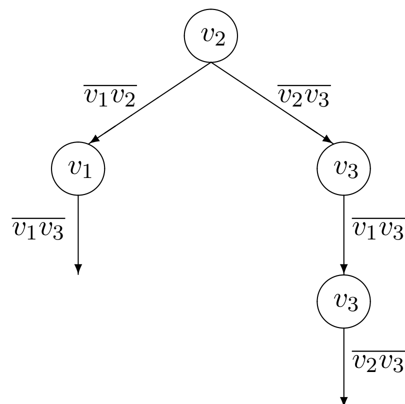
Step 1: Since the first vertex is v_2 the two edges $\overline{v_1v_2}$ and $\overline{v_2v_3}$ become active. The first vertex can be placed in either list; assume it is placed in `left`. If the next vertex was a v_1 then it would be placed in `left` because it has to be connected to the edge $\overline{v_1v_2}$. Similarly if the next vertex was a v_3 it would be placed in `right`. Finally, as a v_2 would be connected to both edges, it would finish the shape, and could be placed in either `left` or `right`. The designators `leftA` and `rightA` are set to v_1 and v_3 respectively.



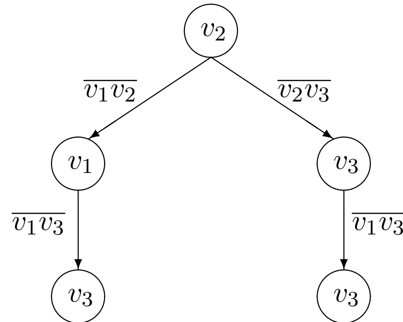
Step 2: The next vertex is in fact v_3 so it is placed in `right`. This toggles the active edges such that $\overline{v_2v_3}$ becomes inactive and $\overline{v_1v_3}$ active. The newly active edge becomes the edge associated with the `right` list. The designator `leftA` is set to v_2.



Step 3: The next vertex is also v_3, indicating a transition to the $\overline{v_2v_3}$ edge. Accordingly the vertex v_3 is placed in the `right` list and the `right` list's edge is changed. The designator `leftA` is updated to v_1.



Step 4: In a similar way, vertex v_1 is placed in the `left` list and the `left` list's edge is changed to $\overline{v_1v_3}$. The designator `rightA` is set to v_2.



Step 5: The final vertex is v_3 which could be placed in either list; it is arbitrarily placed in `left` for convenience. Previously active edges $\overline{v_1v_3}$ and $\overline{v_2v_3}$ are toggled off and the shape is finished.

4.3.1 An Algorithm to Connect Vertex Lists

This algorithms maintains two lists `left` and `right` to store the vertices. The variables `leftA` and `rightA` designate which vertex should be accepted to which list. Both `leftA` and `rightA` are integer values (`1`,`2` or `3`) which correspond to the scene vertices. For each vertex stored in the list `uvt`, which has been sorted in ascending value of t, if its associated scene vertex does not match either `leftA` or `rightA` then that vertex finishes the current shape. If it does match then the designator for the other list (either `leftA` or `rightA`) must be changed. The designator is changed to the vertex not currently indicated by either designator. This can be expressed as sum of all vertex numbers (`6`) minus the total of both current designator values. At the start of each shape the value of `leftA` is initialised to the current vertex's scene vertex, so that the vertex is placed in the `left` list. The function `appendReverse` reverses a list and appends it; other functions are the same as in Listing 4.2.

```
ListofLists connectVertices(List uvt) {
  e       = {<1,1,0>,<1,0,1>,<0,1,1>}
  s       = <0,0,0>
  L       = {}
  S       = {}
  left    = {}
  right   = {}
  leftA   = v.sceneVertex()
```

```
    for each v in uvt {
      if s == <0,0,0> and left != {} {
        S.append(left)
        S.appendReverse(right)
        L.append(S)
        S      = {}
        left   = {}
        right  = {}
        leftA  = v.sceneVertex()
      }

      if v.sceneVertex() != leftA {
        leftA = 6 - (rightA + leftA)
        right.append(v)
      }
      else {
        rightA = 6 - (rightA + leftA)
        left.append(v)
      }

      s = xor(s,e[v.sceneVertex()])
    }

    S.append(left)
    S.appendReverse(right)
    L.append(S)

    return L
  }
```

Listing 4.3: Sorting vertex lists into shapes and ordering by connectivity

4.4 Example Shape Images

Once shapes have been generated and sorted appropriately they can be used to render polygons. Figures 4.1 to 4.7 show example shapes that result from projecting scene triangles with trilinear projection. Scene vertices have been assigned the colours red, green and blue to indicate which scene vertex corresponds to each projected vertex. Colouring on the face has been interpolated trilinearly. Vertices are overlain as circles on the rendering for comparison. The shapes are named according to the number of vertices in a shape. A shape configuration called *(2,2,2,3)* indicates that a scene triangle produces four shapes

when projected by a particular trilinear projection. Three of the shapes have two vertices and the other one has three. Two-vertex shapes are drawn as lines and three-vertex shapes as triangles; with more vertices the shape is drawn as an arbitrary polygon. Scene and view triangle values used to generate these examples are shown in Appendix C.

For comparison the figures also show the result obtained by ray tracing. Depending on the geometry of the triangles, the shapes produced by trilinear projection and ray tracing may differ, although the position of projected vertices corresponds exactly. Section 4.5 discusses techniques to improve the accuracy of drawing shapes between vertices.

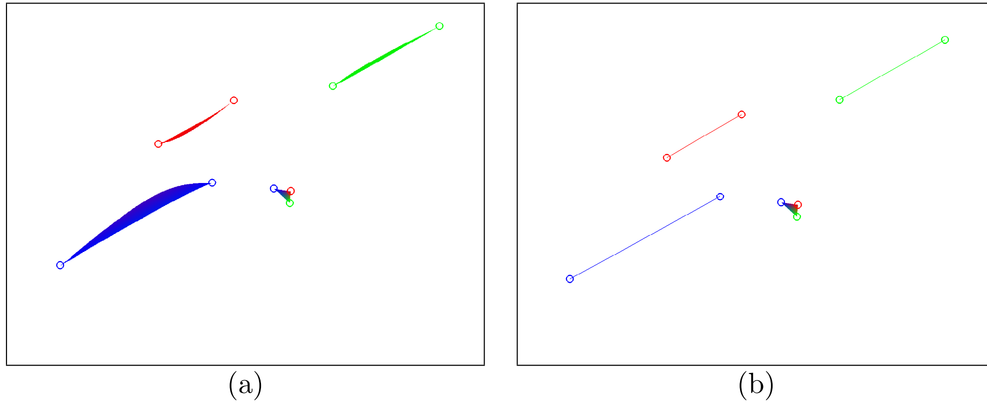


Figure 4.1: A (2,2,2,3) shape configuration: (a) ray trace (b) trilinear projection

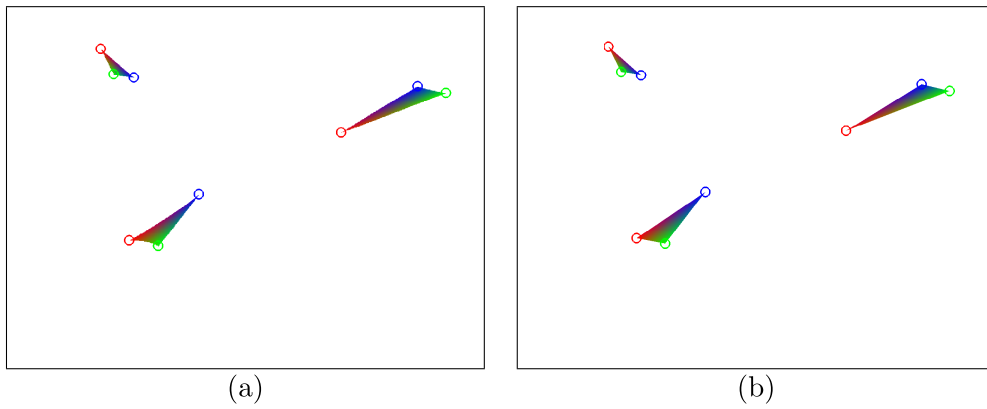


Figure 4.2: A (3,3,3) shape configuration: (a) ray trace (b) trilinear projection

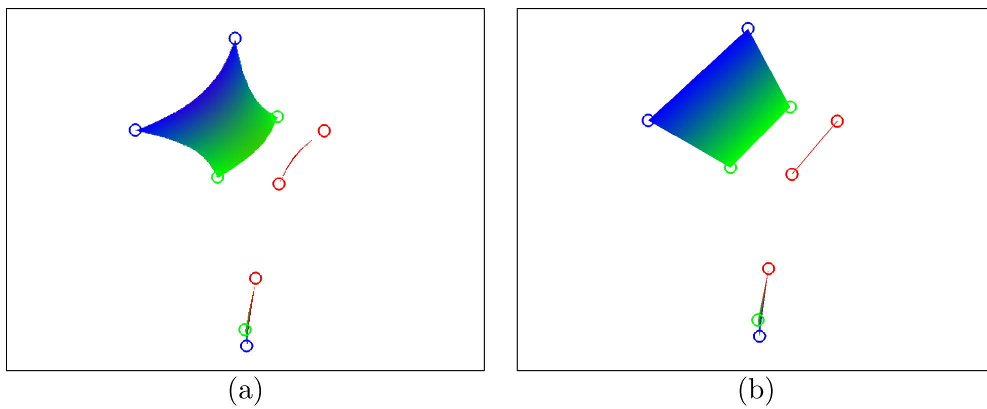


Figure 4.3: A (4,2,3) shape configuration: (a) ray trace (b) trilinear projection

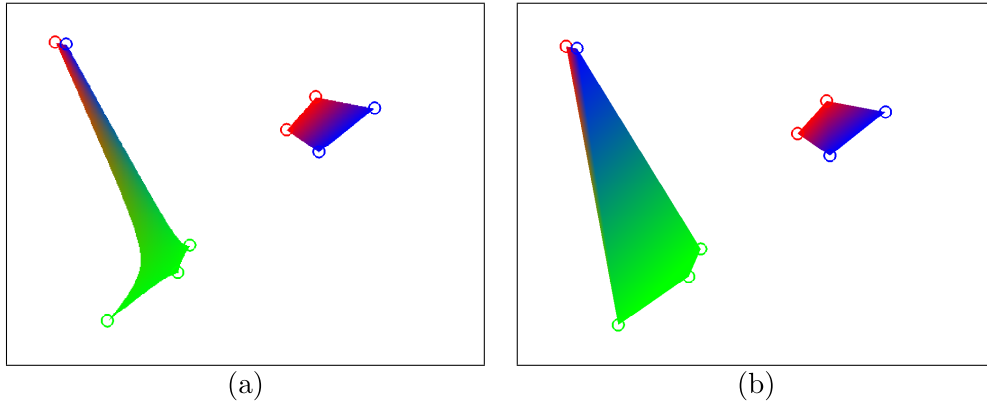


Figure 4.4: A (4,5) shape configuration: (a) ray trace (b) trilinear projection

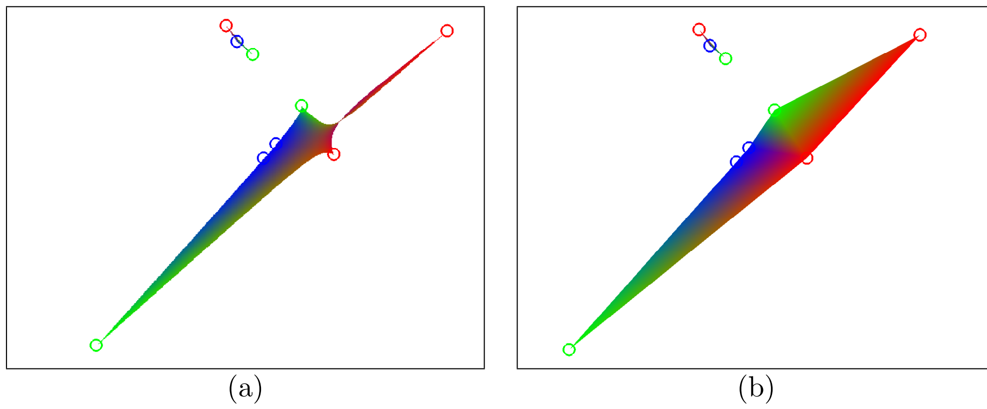


Figure 4.5: A (6,3) shape configuration: (a) ray trace (b) trilinear projection

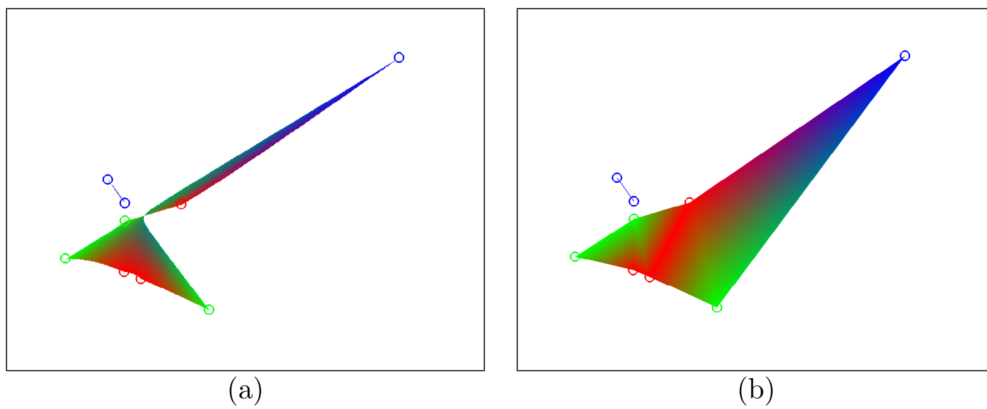


Figure 4.6: A (2,7) shape configuration: (a) ray trace (b) trilinear projection

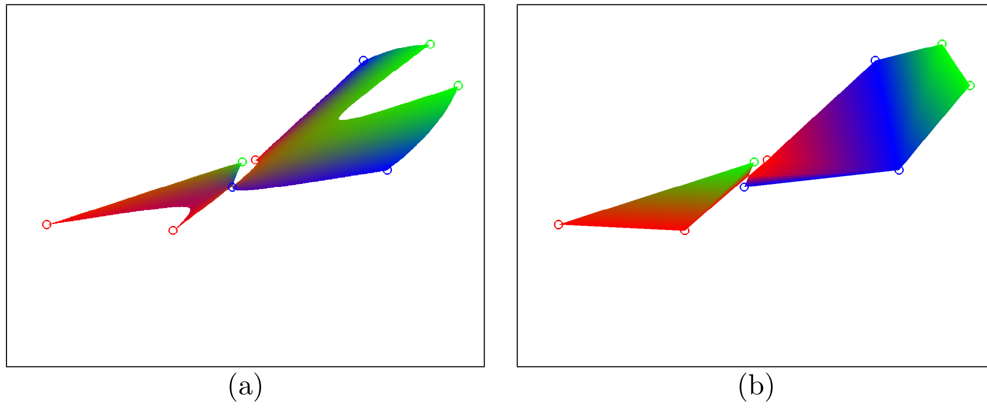


Figure 4.7: A (9) shape configuration: (a) ray trace (b) trilinear projection

4.5 Tessellation

Rendering from a parametric triangle by projecting from the scene triangle vertices does not produce results exactly the same as ray tracing, as illustrated in Figure 4.8. This error is

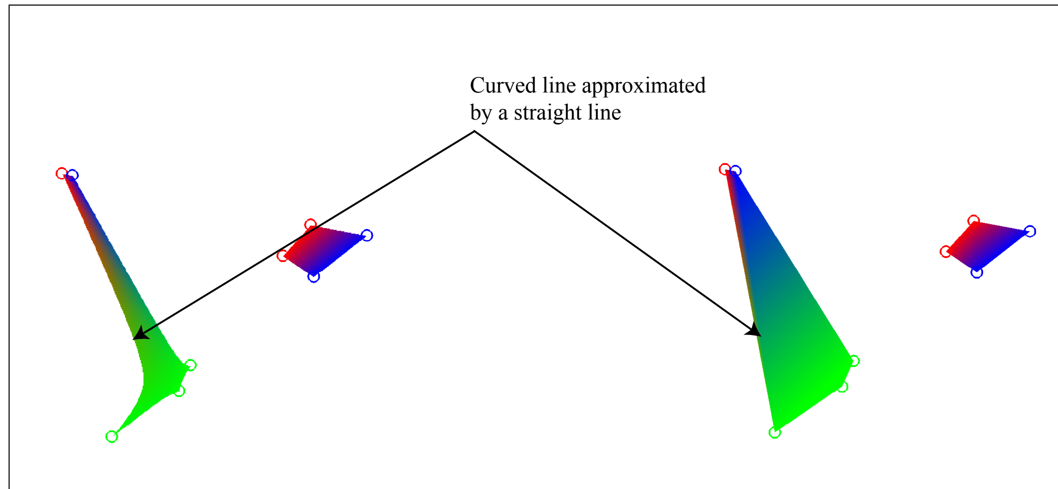


Figure 4.8: Example of the error inherent in the linear approximation of curved shapes

because the shapes are drawn with straight lines between the projected vertices, whereas the correct result would be a curved line. To more accurately render curved lines, tessellation can be used for either the scene triangle or the trilinear projection. Tessellation subdivides the triangle so that the error is lessened. Tessellating scene primitives is a classic way of reducing the error of drawing a curved surface with straight line primitives.

4.5.1 Scene Triangle Tessellation

Many algorithms exist to tessellate triangles; we use a regular sampling with barycentric coordinates. This can be accomplished by stepping through u and v values in Equations 2.4 by a constant incrementor. The values of u and v are restricted to the range $0 \leq u, v, (u + v) \geq 1$ to be within the bounds of the scene triangle. The result of this is a regularly spaced set of points over the triangle. The naive approach to rendering this would then be to join the points into triangles and render each triangle separately. However, some vertices appear on more than one triangle and exploiting this replication results in a more efficient method. After calculating the u, v and t for each point separately, the results can be combined in different ways to render each triangle.

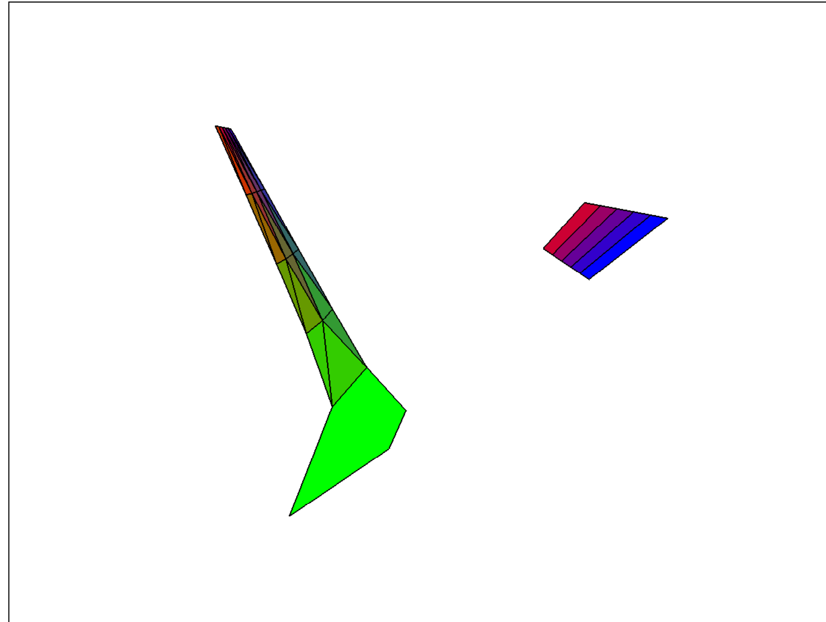


Figure 4.9: Scene triangle tessellated into 25 triangles approximating a (4,5) shape configuration

Figure 4.9 shows the shape previously shown in Figures 4.4 tessellated into 25 triangles and rendered. The triangle edges have been drawn in black to highlight the tessellation. The increased visual accuracy comes at a performance cost, 25 scene points have been projected as compared with 3 in Figure 4.4.

4.5.2 Parametric Triangle Slices

Another tessellation strategy is to sample the scene triangle at discrete t values. For each t value the parametric triangle is a normal triangle. The intersection of this triangle and the scene triangle results in a line, as discussed in Section 4.1. In addition to determining the connectivity of vertices this fact can be used to render scene triangles. For every value of t from the smallest for a particular shape to the largest the intersection line crosses the scene triangle. This line, clipped to the overlapping sections of the scene and parametric triangle (for this particular t value), corresponds directly to a line on the final image. Figure 4.10 shows a scene triangle rendered with 5 extra t samples per shape.

Listing 4.4 shows an implementation to find and add `tessFactor` lines on a particular shape. The minimum and maximum t values of the shape are found (with minimum being set to 0.0 in the case that minimum $t < 0$) with the functions `minimumTValue` and

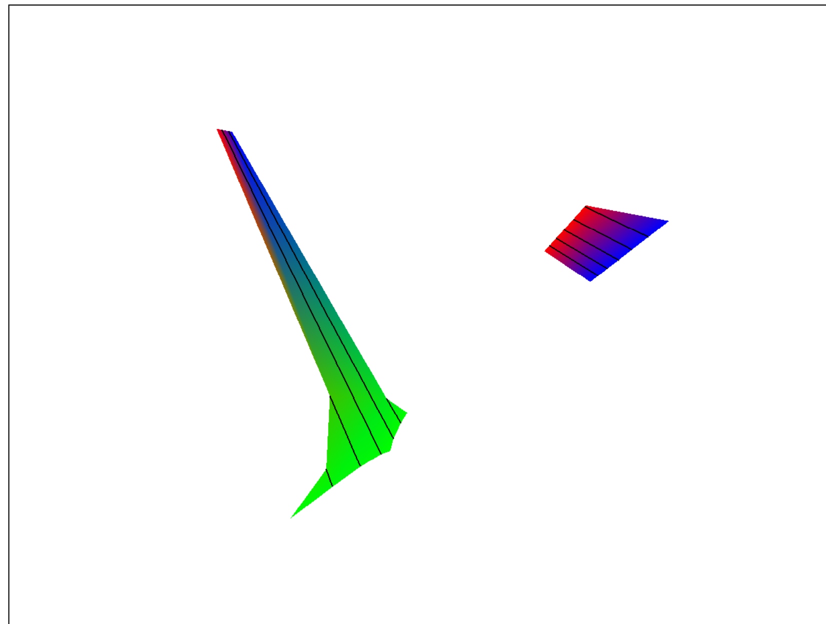


Figure 4.10: Scene triangle sampled at 5 extra t levels per shape approximating a (4,5) shape configuration

`maximumTValue`. Once the minimum and maximum are known the difference between is divided by `tessFactor`. The function `intersectTriangleTriangle` intersects two triangles to form a line segment and returns a list containing the vertices of the line segment in terms of u, v and t.

Once the line segment vertices have been added the shape is drawn incrementally. When drawing, after two tessellated vertices have been processed the current state is drawn. This effectively draws from one tessellation line to the next, including any other vertices between the lines. Drawing the shape incrementally ensures that each tessellation line is drawn explicitly. Drawing the bounding polygon without explicitly drawing each tessellation line would give the same silhouette but the interpolated face values (such as colour or texture coordinates) would not be as accurate.

```
tesselate(TrilinearProjection tP, SceneTriangle sT,
          List uvt, int tessFactor) {
  t    = minimumTValue(uvt)
  maxt = maximumTValue(uvt)

  if t < 0 {
    t = 0
  }
```

```
    if maxt > 0 {
      tstep = (maxt-t)/(tessFactor+1)

      for i = 0..tessFactor {
        t = t + tstep
        uvt.append(intersectTriangleTriangle(project(tP,t),sT))
      }
    }
  }
```

Listing 4.4: Sampling with discrete parametric triangle slices

4.6 Viewing Plane Intersection

Projecting with a parametric triangle means that instead of a view point, as in normal rendering, there is a view plane which corresponds to the plane of the triangle with a t value of 0. In a correct rendering, anything behind this plane should be hidden. In some situations some projected vertices of a scene triangle are in front of this plane and some behind. These shapes must only be partially drawn. The intersection between the scene triangle and the zero t triangle defines a line, which is used to clip shapes whose vertices have mixed positive and negative t values.

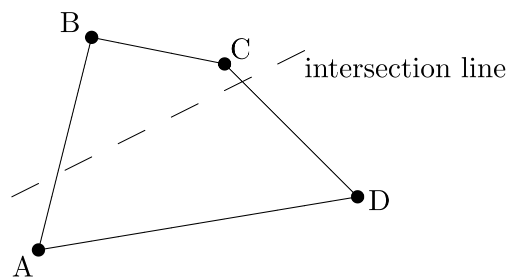


Figure 4.11: A shape partially behind the viewing plane

Figure 4.11 illustrates an examples, where vertices B and C are on the other side of the intersection to A and D. This does not necessarily mean that the view plane and scene triangle intersection must be used to clip the shape. Clipping only occurs if projected vertices have positive and negative t values; a shape that has all vertices with negative t values can be culled.

Clipping can be done in two ways. The line between two projected vertices (for instance A and B) can be clipped against the intersection line. This line between A and B may not be entirely accurate as shown in Section 4.5 and the real intersection may be at a different location. Alternatively the scene vertices that A and B are projected from can also be intersected with the zero t plane giving the exact solution. This second calculation is done in 3D and is hence more computationally expensive than the 2D clipping of the line between A and B.

Listing 4.5 shows an algorithm to clip to the view plane. Function `viewPlaneClip` checks each shape. Function `zeroIntersect` iterates over each edge in a shape (including the last) calling `checkCrossing`. If the first vertex of an edge is below the view plane but the next above, then the intersection point is appended. If both are below the view plane, then nothing is appended to the shape. If the first vertex is above the view plane, it is appended. If the first vertex is above and the second below then the intersection is also appended. The `sceneTriangle` function `edge` returns a line corresponding to the edge between the two specified scene vertices. With the line, `l` in scene space, and the triangle, `tri`, at $t = 0$ the function `intersectLineTriangle` returns their intersection vertex.

```
checkCrossing(Vertex Va, Vertex Vb, List Lt,
                   TrilinearProjection tP, SceneTriangle sT) {
  if Va.tValue() < 0 {
    if Vb.tValue() >= 0 {
      Line l = sT.edge(Va.sceneVertex(),Vb.sceneVertex())
      Triangle tri = project(tp,0)
      Lt.append(intersectLineTriangle(l,tri))
    }
  }
  else {
    if Vb.tValue() >= 0 {
      Lt.append(Va)
    }
    else {
      Lt.append(Va)
      Line l = sT.edge(Va.sceneVertex(),Vb.sceneVertex())
      Triangle tri = project(tp,0)
      Lt.append(intersectLineTriangle(l,tri))
    }
  }
}

zeroIntersect(List V, List Lt, TrilinearProjection tP,
```

```
                SceneTriangle sT) {
  for i = 0..(V.size()-1) {
    checkCrossing(V[i],V[i+1],Lt,tP,sT)
  }
  checkCrossing(V[V.size()-1],V[0],Lt,tP,sT)
}

void viewPlaneClip(ListofLists L, TrilinearProjection tP,
                   SceneTriangle sT) {
  for each L' in L {
    if minimumTValue(L') < 0 {
      Lt = {}
      zeroIntersect(L',Lt,tP,sT)
      copy(L',Lt)
    }
  }
}
```

Listing 4.5: Calculating view-plane intersections

4.7 Summary

Building on the algorithm for projecting a scene point provided in the previous chapter, this chapter examines the problem of projecting a scene triangle. Projecting a scene triangle relies initially on projecting the vertices of the scene triangle separately. Each scene triangle vertice may be projected one or three times and reconnecting the vertices is not trivial because they may now form from one to four shapes of from two to nine vertices. This chapter presents an algorithm to reconnect the vertices into their shapes. A more complicated version of the algorithm constructs ordered polygons, where each vertex is connected to the one before and after it in a list, which may be drawn by standard graphics algorithms.

The polygons drawn by the algorithms in this thesis are not necessarily correct because the lines between vertices should be curved but are represented by straight lines. To alleviate this problem tessellation methods are explored. Tessellating the scene vertex provides more accurate rendering but at a significant performance cost. A new tessellation algorithm is proposed that samples the scene triangle at extra values of t, the parametric triangle parameter, within the shapes. These samples provide extra vertices on the rendered shape at the cost of intersecting two triangles.

Linear projections are able to clip to a view frustum to correctly draw scene primitives partially behind the viewpoint. A trilinear projection does not have a simple view

frustum, so to clip shapes that are partially behind the projection a traversal algorithm is presented. A projected vertex is behind the projection if it has a negative value of t. The clipping algorithm clips scene triangle edges to the trilinear projection plane when the projected vertices change from negative to positive t.

Chapter 5

Multiple Surface Triangles

A projection comprising a single trilinear projection has limited curvature. More complicated projections can be built with a mesh of trilinear triangles. In the same way that a triangular scene mesh can be approximated as smoothly curving by sharing surface normals, so can a projection mesh. Each trilinear triangle projection maps to a unique triangular region of screen space. When projecting with multiple trilinear triangles, issues of continuity arise that can be solved by clipping in scene space.

5.1 Screen Space Clipping

Screen space clipping trims the scene geometry to fit within the bounds of the trilinear projection's triangular region of screen space, after the scene geometry has been projected. In current hardware, clipping planes or stencil buffers can be used to clip geometry to the necessary triangular screen segments. A triangular projection mesh shares normals, which means that the projection should appear continuous at the edges of a triangular section. However, when clipped in screen space, continuity is not necessarily preserved. Scene triangle edges that cross the boundaries between trilinear projections may not join with their corresponding edges in the next trilinear projection. Figure 5.1 illustrates this problem. The red and blue triangles are the projection of one scene triangle from two different trilinear projections. If clipped in screen space, the edges of the red and blue triangles do not meet at the boundary of the trilinear projections as they should. This is a result of the linear approximation of the edges between projected vertices as described in Section 4.5.

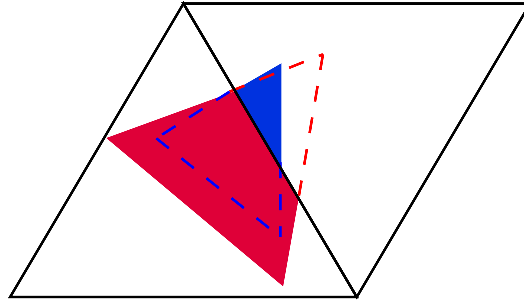


Figure 5.1: A scene triangle spanning two trilinear projections with a discontinuity

5.2 Scene Space Clipping

Discontinuities arising from screen space clipping can be addressed by clipping scene triangles to the volume swept out by projecting the parametric triangle into scene space. The result is equivalent to tessellating the projected scene triangle at the boundary of the next trilinear projection. Each edge of a surface triangle forms a parametric line segment that traces out a surface through space which may intersect with edges in the scene triangles. To correctly clip to a parametric surface triangle region, the three parametric line segments defining the triangle edges must be traced through all the scene triangles, and the triangles clipped according to the intersections.

A line segment of the parametric triangle at t is defined parametrically by:

$$edgesegment\,(s,t) := p_i + tn_i + s\,(p_j + tn_j) \quad i,j = 1,2,3 \quad i \neq j \tag{5.1}$$

where s varies from 0 to 1. Consider the intersection between a parametric edge, defined by the points $p_i + tn_i$ and $p_j + tn_j$, and a scene triangle edge, defined by the points p_{s1} and p_{s2}. The value of t at the intersection as projected out of the trilinear projection, can be determined independently of s because for the two line segments to intersect they must lie on the same plane. According to Equation 3.13 the four points are coplanar when:

$$\begin{vmatrix} p_{s1x} - p_{s2x} & p_{s1y} - p_{s2y} & p_{s1z} - p_{s2z} \\ p_{ix} + tn_{ix} - p_{s2x} & p_{iy} + tn_{iy} - p_{s2y} & p_{iz} + tn_{iz} - p_{s2z} \\ p_{jx} + tn_{jx} - p_{s2x} & p_{jy} + tn_{jy} - p_{s2y} & p_{jz} + tn_{jz} - p_{s2z} \end{vmatrix} = 0 \tag{5.2}$$

This can expanded giving a quadratic in terms of t. This quadratic may have two, or no real solutions. Each value of t defines a triangle and the scene line can be intersected with that triangle using the same algorithm as for ray tracing the intersection of a ray and triangle. Figure 5.2 illustrates the parametrically defined edge, show in blue, intersecting an edge of a scene triangle, shown in red, with the intersection point marked.

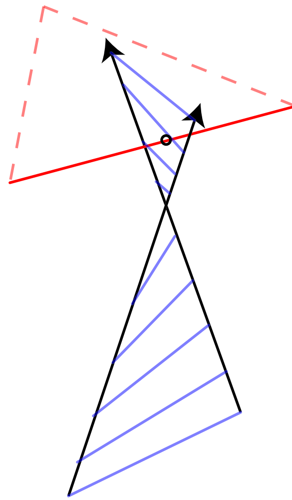


Figure 5.2: A trilinear projection edge swept out into scene space and intersected with a scene triangle

5.2.1 Algorithm for Determining Intersection t Values

The determinant in Equation 5.2 can be described in terms of vector operations. When grouped by terms of t this gives the coefficients of the quadratic of t, whose roots are the projection lengths. Listing 5.6 can calculate the roots and coefficients for any of the edges. In Listing 5.6 the quadratic is assumed to be `coeff[0]`t^2+`coeff[1]`t+`coeff[2]`, vertices `s1` and `s2` describe two points on the line to intersect with the edge, and integers `edge1` and `edge2` are the enumerated numbers of the edge vertices. Other vectors are properties of the trilinear projection as described in Appendix B. The function `quadraticSolveReal` returns a list of the real roots of the specified quadratic. The t values calculated by this listing are converted to intersection points in Listing 5.8.

```
List intersectLineEdge(Vertex s1, Vertex s2,
                       int edge1, int edge2) {
  coeff = {}
```

```
  switch(edge1+edge2) {
  case 3 : intersectLineEdge12(s1,s2,coeff)
  case 4 : intersectLineEdge31(s1,s2,coeff)
  case 5 : intersectLineEdge23(s1,s2,coeff)
  }

  return quadraticSolveReal(coeff[0],coeff[1],coeff[2])
}

void intersectLineEdge12(Vertex s1, Vertex s2, List coeff) {
  coeff[0] = dot(s1,cn1n2) - dot(s2,cn1n2)
  coeff[1] = dot(s1,(s2*en1n2+cn1p2+cp1n2)) - dot(s2,(cn1p2+cp1n2))
  coeff[2] = dot(s1,(s2*ep1p2+cp1p2)) - dot(s2,cp1p2)
}

void intersectLineEdge23(Vertex s1, Vertex s2, List coeff) {
  coeff[0] = dot(s1,cn2n3) - dot(s2,cn2n3)
  coeff[1] = dot(s1,(s2*en2n3+cn2p3+cp2n3)) - dot(s2,(cn2p3+cp2n3))
  coeff[2] = dot(s1,(s2*ep2p3+cp2p3)) - dot(s2,cp2p3)
}

void intersectLineEdge31(Vertex s1, Vertex s2, List coeff) {
  coeff[0] = dot(s1,cn3n1) - dot(s2,cn3n1)
  coeff[1] = dot(s1,(s2*en3n1+cn3p1+cp3n1)) - dot(s2,(cn3p1+cp3n1))
  coeff[2] = dot(s1,(s2*ep3p1+cp3p1)) - dot(s2,cp3p1)
}
```

Listing 5.6: Finding edge intersection t values

5.2.2 Integrating Intersection Points into Drawing Primitives

Each surface and scene triangle comprises three edges and each surface edge must be checked against each scene edge. This means nine tests must be done per surface scene triangle pair. Each test may furnish two valid intersection points. Additionally there is the case of a spanning scene triangle, whose vertices and edges all fall outside the volume of the parametric triangle but whose area does not.

To handle all these cases, intersection vertices are inserted in the shape without culling outliers. The resulting shape is clipped again in screen space, so that spanning and semi-spanning shapes are correctly drawn. Each edge of a projected shape is tested and if either vertex lies outside the trilinear projection then intersecting vertices are added.

Listing 5.7 takes a set of shapes in `viewEdgeClip` and checks each one to see if intersecting vertices need be added by calling `edgeIntersect`. The function `edgeIntersect` checks the edge between each vertex pair of the shape including from the end to start vertices. A vertex is deemed to be outside if its u and v parameters and their sum do not lie between 0 and 1. If either vertex is outside then `spanEdge` is called to insert intersection vertices.

```
void viewEdgeClip(ListofLists L, SceneTriangle sT) {
  for each L' in L
    Lt = {}
    edgeIntersect(L',Lt,sT)
    copy(L',Lt)
  }
}

void edgeIntersect(List V, List Lt, SceneTriangle sT) {
  for i = 0..V.size()-1
    checkEdge(V[i],V[i+1],Lt,sT);
  }
  checkEdge(V[V.size()-1],V[0],Lt,sT);
}

void checkEdge(Vertex Va, Vertex Vb, List Lt, SceneTriangle sT) {
  Lt->append(Va);

  if(!inside(Va) || !inside(Vb)) {
    spanEdge(Va,Vb,Lt,sT);
  }
}

boolean inside(Vertex v) {
  alpha = 1.0 - (v.uValue() + v.vValue())
  return ( alpha > EPSILON_ZERO_MINUS &&
           v.uValue() > EPSILON_ZERO_MINUS &&
           v.vValue() > EPSILON_ZERO_MINUS)
}
```

Listing 5.7: Finding clipping points in shapes

When either vertex lies outside the trilinear projection volume, there may be more than one valid intersection point between them. The quadratic nature of the edge equation means that there are two possible intersections per edge, making a total of six possible intersections for the 3 edges of the trilinear projection. If these intersection points occur between the t values for the starting and ending vertex and lie in the bounds of the trilinear projection,

they are inserted in the shape. Listing 5.8 shows the function `spanEdge` which finds and inserts the valid intersection points.

```
void spanEdge(Vertex Va, Vertex Vb, List Lt,
              TrilinearProjection tP, SceneTriangle sT) {
  sVa = sT.vertex(Va.sceneVertex())
  sVb = sT.vertex(Va.sceneVertexRight())

  t = {}

  t.append(tP.intersectLineEdge(sVa,sVb,1,2))
  t.append(tP.intersectLineEdge(sVa,sVb,2,3))
  t.append(tP.intersectLineEdge(sVa,sVb,3,1))

  if Va.tValue() < Vb.tValue() {
    tLow=Va.tValue()
    tHigh=Vb.tValue()
  }
  else {
    tLow=Vb.tValue()
    tHigh=Va.tValue()
  }

  Ltemp = {}
  for i = 0..t.size() {
    if (t[i] > tLow) and (t[i] < tHigh) {
      p = intersectLineTriangle(sVa,sVb,project(tp,t[i])
      if inside(p) {
        Ltemp.append(p)
      }
    }
  }

  Ltemp.sortByT()
  if Va.tValue() < Vb.tValue() {
    Lt.append(Ltemp)
  } else {
    Lt.appendReverse(Ltemp)
  }
}
```

Listing 5.8: Inserting clipping points into shapes

Inserting extra clipping points into the polygons requires additional computation to find the points and additional complexity in the resulting polygon. Both these factors

reduce performance. If scene triangles are sufficiently small when projected or if enough scene triangles do not span across trilinear projections, then clipping may be unnecessary. Such determinations are application specific.

5.3 Summary

This chapter describes how arbitrarily complicated nonlinear projections can be approximated by composite trilinear projections, with the scene rendered separately and clipped to each projection triangle. In addition the chapter describes a scene-space clipping algorithm to ensure continuity across multiple trilinear projections. The clipping algorithm projects each trilinear projection edge into the scene and finds intersections with scene primitives. These intersections are the solutions to a quadratic equation in terms of the edge's parameter t. Each of these intersections is added as an additional projected vertex to the post projection shapes. When a scene primitive is imaged across multiple trilinear projections the added intersection points align, ensuring continuity.

Chapter 6

Nonlinear Projection for Visualisation

One application of the parametric triangle projection techniques is nonlinear projections for visualisation. In data visualisation the aim is to present a view of the data such that relationships become apparent. Distortions of the data can provide emphasis on particular areas, or show the variation of the data against a base, such as the variation of the earth against a perfectly smooth ellipsoid. Nonlinear projections provide a way of implementing distortions by specifying a suitably shaped viewing surface.

Distortions of the data, while being fundamentally different in implementation than rendering multi-perspective images, highlight the potential and applications for multi-perspective images. Both seek to present the data in a changed way so that previously unseen properties become apparent. This chapter describes some classes of nonlinear projections for visualisation and presents demonstration renderings made using the algorithms put forward in this thesis.

6.1 Detail and Context

A Distortion Orientated Display (DODs) is a general visualisation technique based around the distortion of data. DODs seek to show detail and context simultaneously. The general problem is that when detail is shown, much of the screen is filled with that detail. If the surrounding data is shown at the same level of detail, it would not fit on the screen. To accommodate this, a DOD view shows a region of focus in detail, with a smooth transition to a region of context at a lower detail level.

For example, Smith [Smi97] defines a distortion called a frustum display, which is able to achieve levels of detail sufficient for a city level road map, whilst showing the context of the whole of Australia. Both Carpendale [CCF97] and Winch [WCS00] expanded the idea of distortion orientated displays to allow for focus regions in 3D data sets. These prove useful for highlighting sections of particular detail in a scene without zooming and therefore cutting out periphery data.

To implement a similar concept with nonlinear projection we define a projection surface, shown in Figure 6.1, whose outer rays tends toward a perspective projection but whose inner rays tend toward an orthographic or even inverse perspective projection. Each triangular mesh section in Figure 6.1 is a trilinear projection and Figure 6.2 shows the surface and the scene it used to render. Figure 6.3 shows the cube scene through a perspective projection, and Figure 6.4 shows the result of rendering the same scene with the DOD projection surface. Figure 6.4 (a) is ray traced and (b) is rendered with trilinear projection, showing that the trilinear projection produces results very similar to ray traced image though much faster. The cube scene is composed of 686 scene triangles to simulate the performance characteristics of a more complicated scene, full performance details can be found in Section 8.7. The cube in Figure 6.4 is larger than in the perspective view in Figure 6.3, though the checkerboard is the same size at its edges.

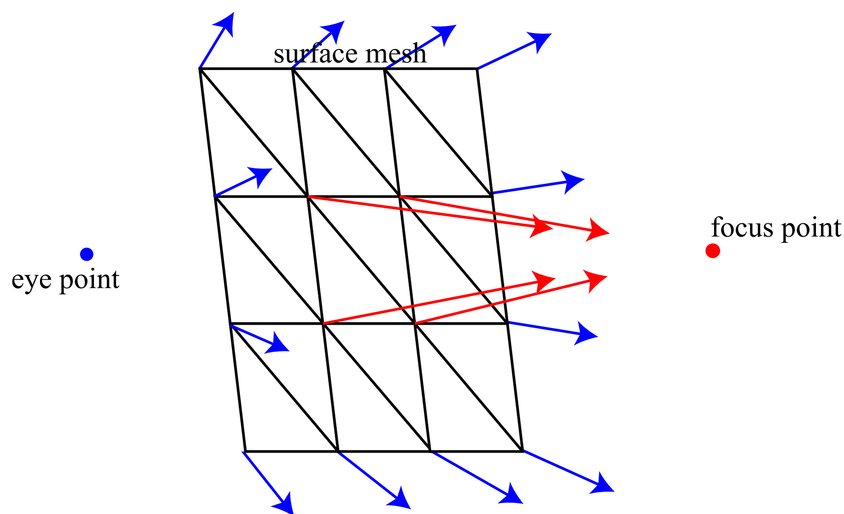


Figure 6.1: A Distortion-Oriented Display mesh projection surface

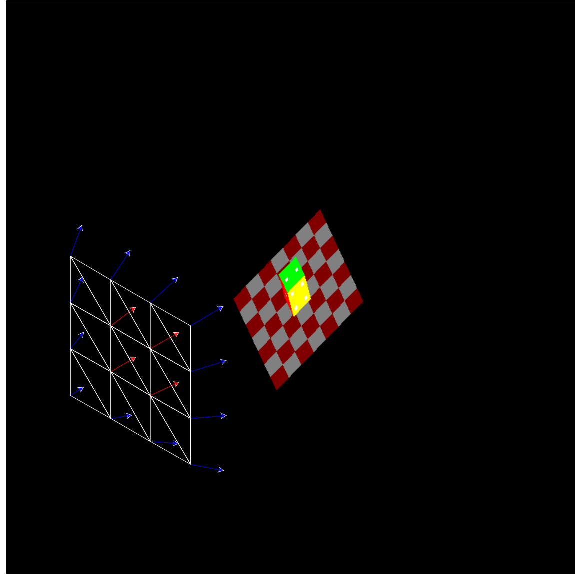


Figure 6.2: A Distortion-Oriented Display projection surface and a cube scene

6.2 Multiple Perspective Views

Multi-perspective views are intended to give a continuous transition between arbitrary views on the data. Expanding on the concept of detail and context, multiple views aim to provide different interpretations seamlessly interwoven. In previous work, [VC01a], see Appendix E, the idea of distorting a mainly planar world onto the inside of cylinder was examined. This was proposed, in the application of virtual maze navigation, so that two different perspectives on the maze could be simultaneously realised: a local view of the undistorted maze walls, and a navigational view of the distant maze perpendicular to the users viewpoint. Figure 6.5 shows an example of the maze distortion.

A trilinear perspective approach to the problem illustrated by Figure 6.5 would attempt to provide a first person, context and transitional view in a single projection. Figure 6.6 shows a side view of such a surface. The perspective projection portion provides detail, the top orthographic provides context and the transition portion ties the two together. An important distinction between the distortion and projection is that scene data may be seen more than once in the projected approach. Figure 6.7 (a) shows a the cube scene ray traced from the surface in Figure 6.6, and Figure 6.7 (b) shows a trilinear projection of the same

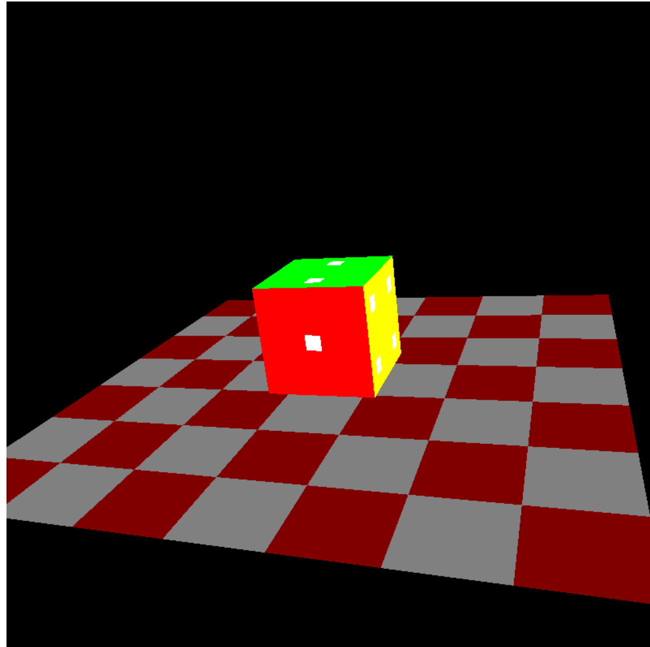
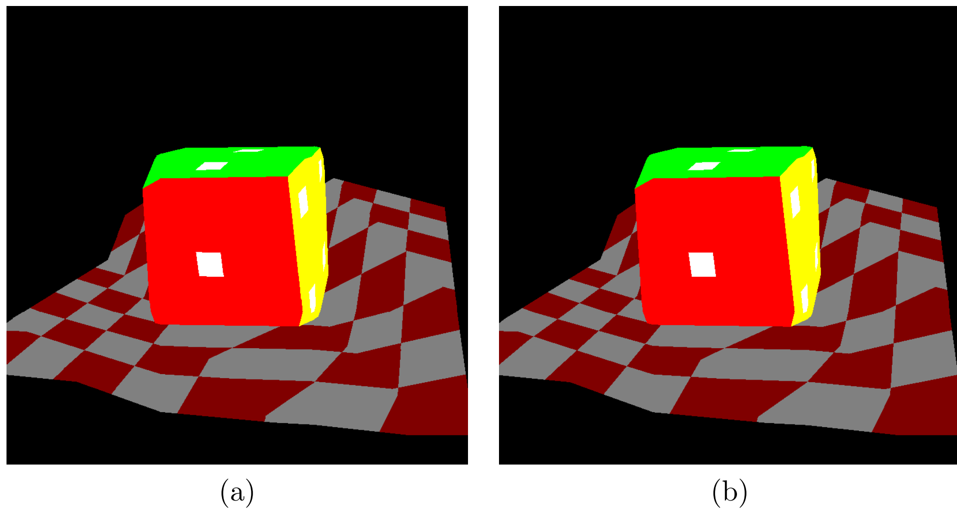


Figure 6.3: A perspective projection of a cube



(a) (b)

Figure 6.4: A Distortion-Orientation projection of a cube (a) ray trace (b) trilinear projection

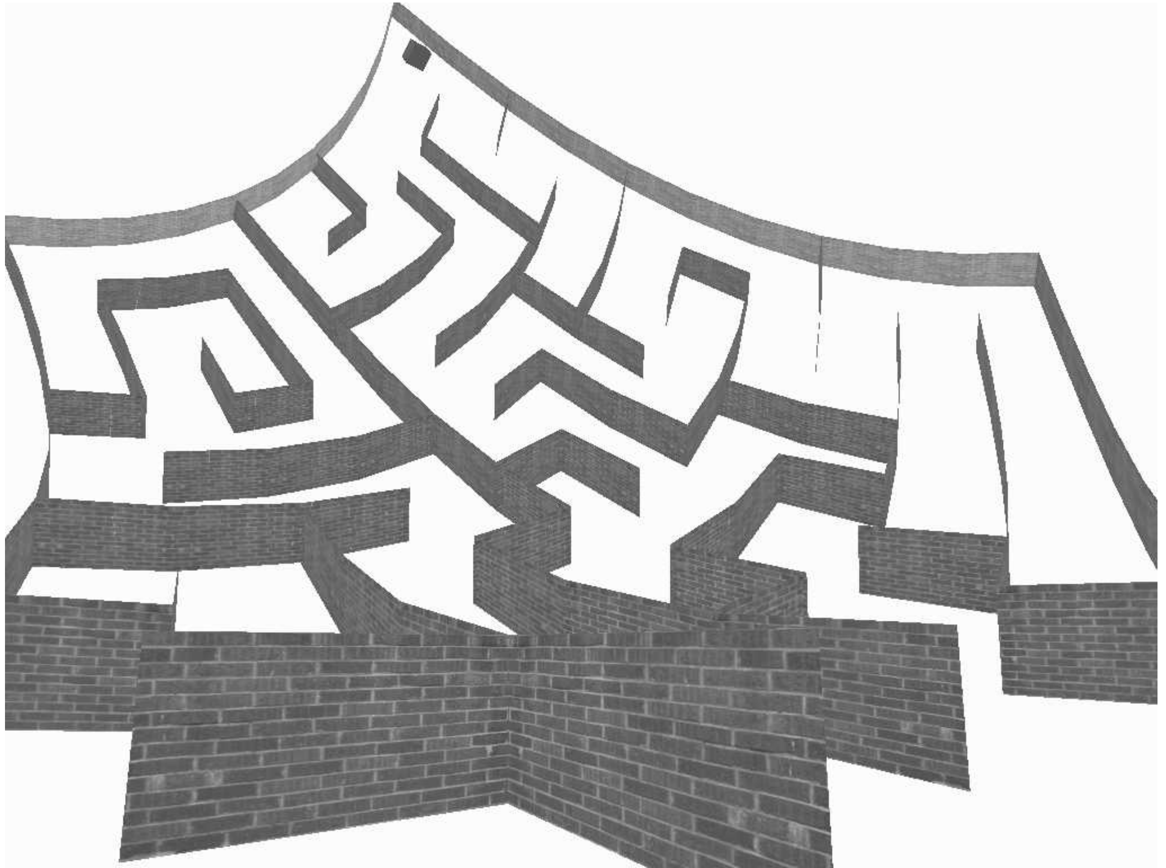


Figure 6.5: A maze distorted in a cylindrical fashion to show context

image. Again the trilinear projected image matches closely with the ray traced version. The visualisation provides two distinct views on the cube, from the left and from the front, and a transitional region in the middle that ties the two views together.

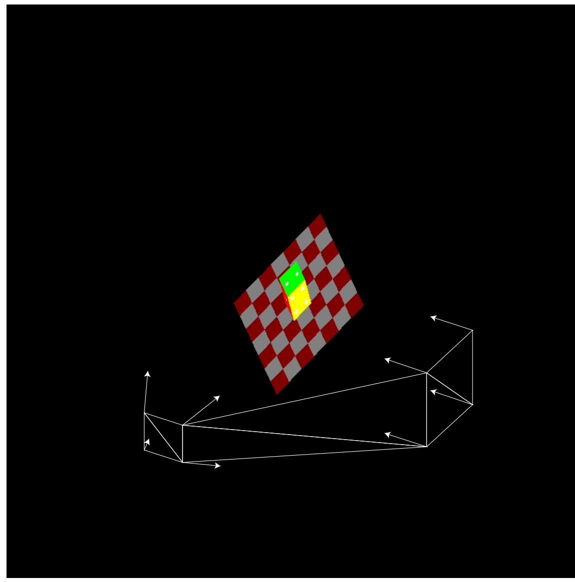
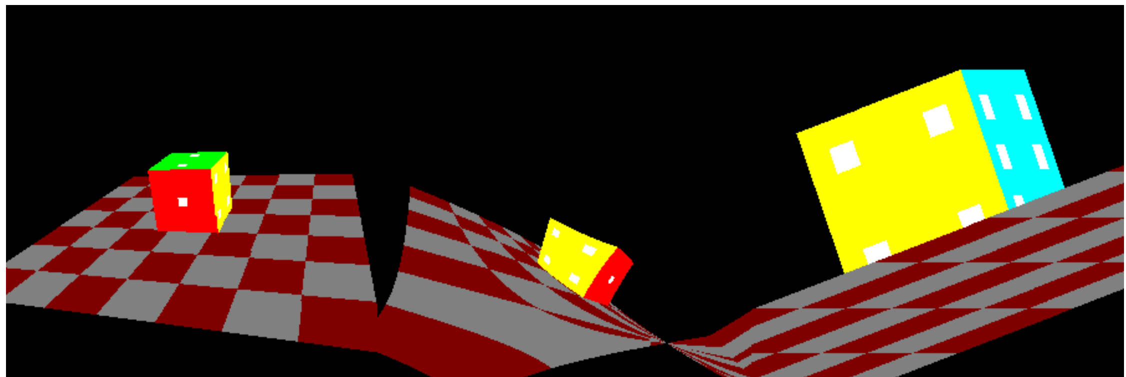


Figure 6.6: A multiple-perspective approach to showing first person detail and side view context

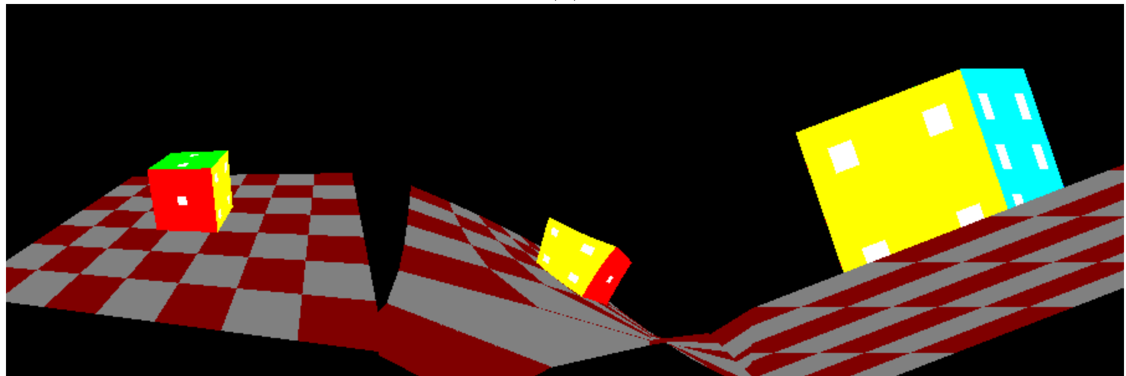
6.3 Mappings

Distorting an object to view it better is most commonly illustrated with map projections, which take a sphere, generally the earth, and transform it to a 2D map. Different projections preserve different features of the data while distorting others. For example in the Mercator projection directions are conserved, although sizes are not, to allow for easy sea navigation. Although the distortion of size is an artifact, it allows for a better understanding of some aspects of the globe.

In general, mappings show the variation of a data set relative to a surface, such as the variation of the earth surface versus a sphere. Figure 6.8 shows a diagram of a spherical mesh viewing a cube data set. The resulting visualisation communicates the nature of a relationship between the data and viewing surface. Figure 6.9 (a) and (b) show a cube



(a)



(b)

Figure 6.7: A cube rendered from a surface derived from Figure 6.6 (a) ray trace (b) trilinear projection

projected by a spherical viewing surface using ray tracing and trilinear projection. The resulting image is related to a map projection, and shows a map of the cube's faces. The ray traced and trilinear projection images are very similar, though the zigzag pattern at the top and bottom of Figure 6.9 (b) is due to degenerate trilinear projections at the poles.

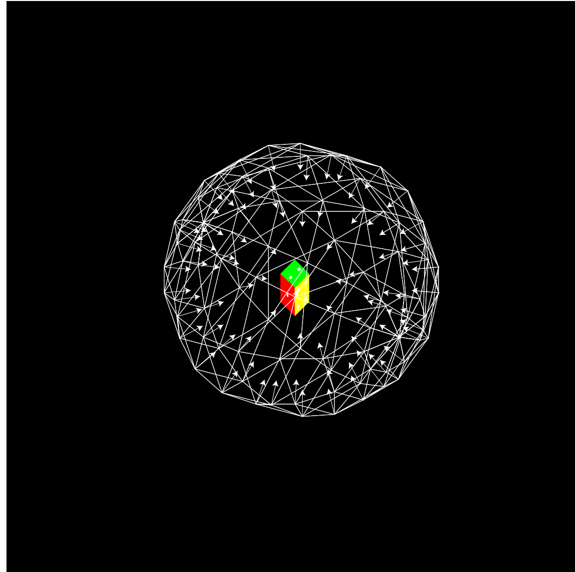


Figure 6.8: An spherical mesh mapping a relation between the scene data and surface

6.4 Summary

Projection surfaces defined by triangular meshes provide sufficiently expressive nonlinear projections to implement many forms of visualisations. These visualisations, similar in nature to data distortions, provide non–traditional views on 3D scenes. These views can be useful for gleaning information from data that is not readily available when viewed through a standard projection. The trilinear projection algorithms in this thesis provide an accurate and fast alternative to implementing these projections with ray tracing as shown in Section 8.7.

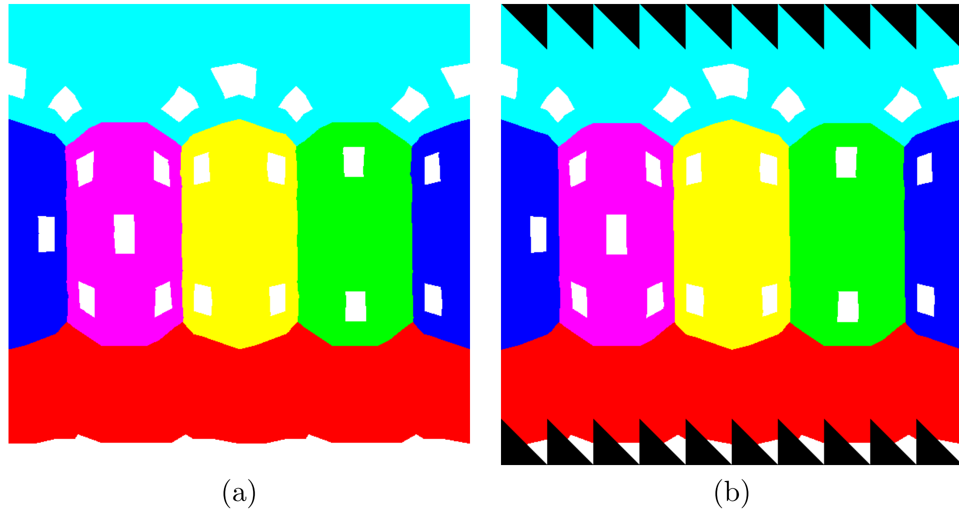


Figure 6.9: A cube rendered by a spherical projection surface (a) ray trace (b) trilinear projection

Chapter 7

Reflections and Refractions

Rendering reflections and refractions is an important problem in generating realistic scenes. The reflection on a plane from a perspective view corresponds to a perspective view through that plane. This can be used to render planar reflections, as shown by Diefenbach [Die96]. Refractions and reflections on non-planar surfaces are not so simple. The ray paths do not change in a linear manner and may be poorly approximated by linear projections. The most common nonplanar surface in computer graphics is the polygon mesh with arbitrary normals. The nonlinear projection methods developed in this thesis can be used to approximate reflections in polygon meshes.

7.1 Integrating Reflection and Refraction Projections into a Scene

Reflections and refractions approximated with trilinear projections can be integrated into the rendering pipeline in the same way planar reflections are integrated by Diefenbach [Die96]. Incident rays on first-hit reflections and refractions depend on the eyepoint and vertex points. With multi-hit reflections and refractions the incident ray to a particular scene vertex can be calculated as a trilinear interpolation of the view surface triangle's normals with the u and v values of the scene vertex's projection.

7.2 Reflections

The specular reflection of a ray is governed by the equation:

$$\theta_r = \theta_i \tag{7.1}$$

where θ_r is the angle of reflection and θ_i is the angle of incidence. This leads to following generally understood equation:

$$r = 2\,(n \cdot i)\,n - i \tag{7.2}$$

where r, n and i are unit vectors for the reflection, surface normal and incidence direction.

7.2.1 First-Hit Reflections on a Polygon Mesh

First-hit reflections refer to reflections of rays that come directly from the eye point to a reflector. Reflections off a curved surface represented by a polygonal mesh can be defined by considering each point on a triangular section of the polygon mesh, the interpolated surface normal and calculating the angle of incidence from the eye point to the surface point.

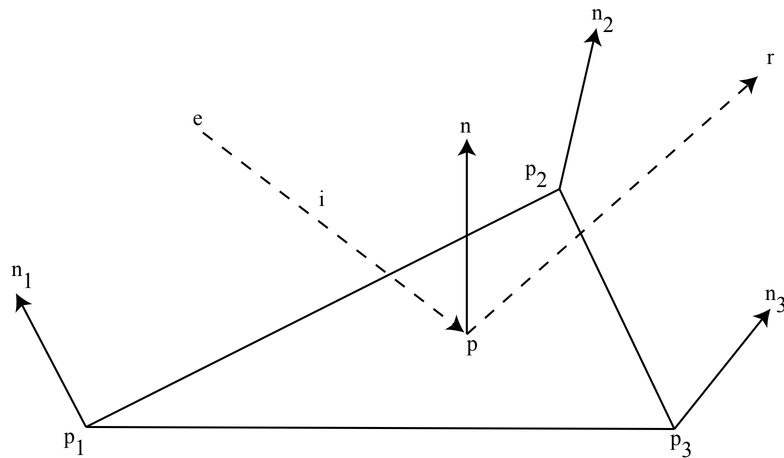


Figure 7.1: A diagram of a first-hit reflection

Figure 7.1 shows a diagram of a first-hit reflection. The incidence ray, i, emanates from the eye point, e, and the ray of reflection, r, reflects off the surface of the scene triangle. The normal vector, n, at the point of reflection, p, is interpolated from the scene triangle's vertex normals, $n_{1..3}$. For a trilinearly interpolated surface point the reflection vector r is

given by:

$$n = (1-u-v)n_1 + un_2 + vn_3 \quad (7.3)$$
$$p = (1-u-v)p_1 + up_2 + vp_3 \quad (7.4)$$
$$i = \frac{p-e}{|p-e|} \quad (7.5)$$
$$r = 2(n \cdot i)n - i \quad (7.6)$$

where $p_{1..3}$ are the triangle vertices, $n_{1..3}$ are the normals at each vertex, and e is the eyepoint.

7.2.2 Approximated First-Hit Reflections

Reflections on trilinear projections can be approximated with the algorithms presented in this thesis. If reflection vectors are calculated at the vertices of the reflecting triangle, this triangle and reflection vectors can form a trilinear projection. The points $p_{1..3}$ and vectors $r_{1..3}$ in Figure 7.2 form a trilinear projection approximating a reflection. The projection is exactly correct at the vertices, but the reflection vectors across the surface will in general only approximate the correct solution. This is because instead of interpolating the surface normal and then calculating the reflection vector, the reflection vector is interpolated from the vertex reflection vectors. However the computed reflection has several desirable characteristics. For example it is nonlinear, which means that the reflection is curved, as is expected, and it is continuous across multiple reflective facets. Figure 7.2 shows how the reflection error arises. Vector r' is interpolated as described in this section, whereas the vector r is the correct reflected vector. The equations that define r' are:

$$i_1 = \frac{p_1 - e}{|p_1 - e|} \quad (7.7)$$
$$i_2 = \frac{p_2 - e}{|p_2 - e|} \quad (7.8)$$
$$i_3 = \frac{p_3 - e}{|p_3 - e|} \quad (7.9)$$
$$r_1 = 2(n_1 \cdot i_1)n_1 - i_1 \quad (7.10)$$
$$r_2 = 2(n_2 \cdot i_2)n_2 - i_2 \quad (7.11)$$
$$r_3 = 2(n_3 \cdot i_3)n_3 - i_3 \quad (7.12)$$
$$r' = (1-u-v)r_1 + ur_2 + vr_3 \quad (7.13)$$

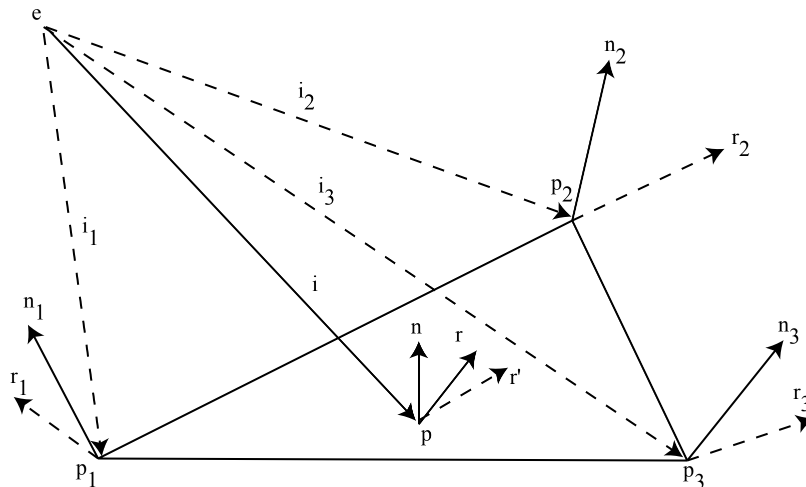


Figure 7.2: Error in an approximation of first-hit reflections

7.2.3 Multi-Hit Reflections on a Polygon Mesh

Multi-hit reflections do not directly trace backwards to the eye point but rather to one or more intermediate reflectors. This means incident rays do not simply merge back to the eye point, but may be arbitrarily more complicated. These reflections can also be approximated by calculating the reflection rays at the vertices and projecting as a trilinear projection, although the error in the computed reflection will be greater than for the first-hit reflection. A more detailed polygon mesh means a more accurate approximation of the reflection, at the cost of increased computation.

7.3 Refraction

Refraction is the bending of light due to the difference in refractive index between two materials. Simple refraction can be described by Snell's law, which shows the relation between angles of incidence, refraction and the refractive index of the materials. Snell's law is described by the following relation:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i} \tag{7.14}$$

where θ_i is the angle of incidence, θ_t is the angle of transmission, η_t is the refractive index of the material the ray is entering and η_i is the refractive index of the material the ray is leaving. This leads to the following equation:

$$q = \eta i - (\cos\theta_t + \eta\cos\theta_i)n \tag{7.15}$$

where q, i and n are the vectors of transmission, incidence and the surface normal, and $\eta = \frac{\eta_t}{\eta_i}$.

7.3.1 First-hit Refraction on a Polygon Mesh

As with first-hit reflection, incident rays all emerge from a view point. Each transmitted ray q across the surface of a triangular section of the mesh, is determined by the following equations:

$$n = (1-u-v)n_1 + un_2 + vn_3 \tag{7.16}$$

$$p = (1-u-v)p_1 + up_2 + vp_3 \tag{7.17}$$

$$i = \frac{p-e}{|p-e|} \tag{7.18}$$

$$q = \eta i - (\cos\theta_t + \eta\cos\theta_i)n \tag{7.19}$$

7.3.2 Approximating Refraction

Refraction can be approximated with trilinear projection in a similar manner to reflection. Transmitted rays are calculated at the vertices and these rays forms a trilinear projection. An approximated transmitted ray q' is described by the following equations:

$$q_1 = \eta i_1 - (\cos\theta_t + \eta\cos\theta_i)n_1 \tag{7.20}$$

$$q_2 = \eta i_2 - (\cos\theta_t + \eta\cos\theta_i)n_2 \tag{7.21}$$

$$q_3 = \eta i_3 - (\cos\theta_t + \eta\cos\theta_i)n_3 \tag{7.22}$$

$$q' = (1-u-v)q_1 + uq_2 + vq_3 \tag{7.23}$$

Again this approximated transmitted ray path is not necessarily equal to the correct ray q.

7.3.3 Multi-Hit Refraction on a Polygon Mesh

Like reflection, multi-hit refraction can mean arbitrarily complicated transmission ray surfaces. Approximation with trilinear projection by can still be used by calculating transmission rays at triangle vertices.

7.4 Example Projections

Figure 7.3 shows a scene with a reflective sphere and a cube that is being rendered from a view point close to the sphere's surface. The rays at the four corners of the view intersect the sphere and are reflected off at various angles. Figure 7.4 (a) shows a ray traced image similar to that which would be seen in the section of reflective sphere shown in Figure 7.3. The cube scene is the same as is introduced in Chapter 6. Figures 7.4 (b) to (f) show a trilinear projection approximation of the image with increasing projection mesh resolution. In the 1x1 surface, Figure 7.4 (b), the trilinear projection is two coplanar triangles whose normals are the reflection vectors show in Figure 7.3. Figures 7.4 (c) to (f) are rendered from projection surfaces that are increasingly accurate approximations of the surface of the sphere and the reflection vectors off the sphere's surface. The increasing mesh resolution means more trilinear projections are used to approximate the reflection, resulting in more accurate rendering.

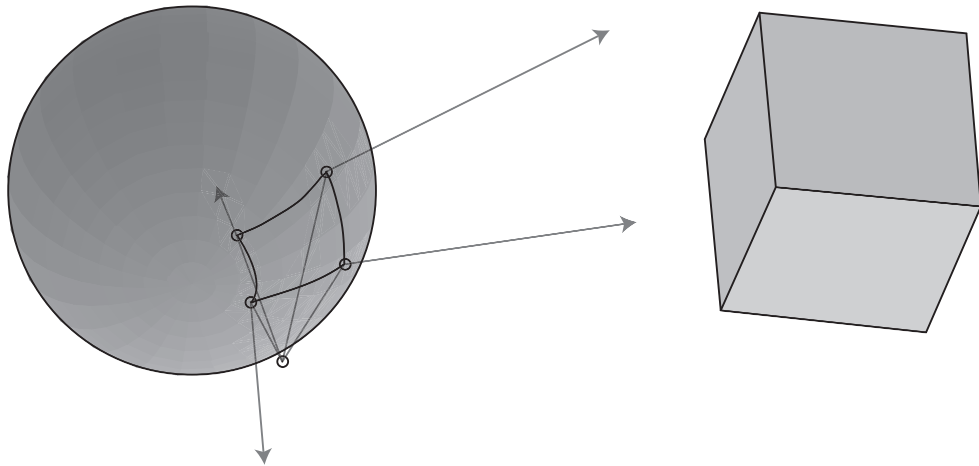


Figure 7.3: A cube reflected in a sphere

Figures 7.5 show a reflection of a cube scene on a flat plane 5x5 mesh, with normal vectors randomly perturbed at the mesh points. This is similar to the sort of image amusement park funny mirrors generate. Figure 7.5 (a) is the correct ray traced reflection and Figure 7.5(b) is the reflection approximated by trilinear projection.

Figures 7.5 and 7.4 show the trilinear projection generated images are a good ap-

proximation to the ray traced images, although the trilinear projection images were rendered much more quickly, see Section 8.7 for details of the rendering performance.

Figures 7.6 (a) to (f) show a refraction of a cube scene through a plane whose normals are coincident. This simulates the effect of viewing an object through a convex lens. Figure 7.6 (a) is the correct ray traced solution and figures 7.6 (b) through (f) show trilinear projections to approximate this using 1x1, 2x2, 3x3, 4x4 and 5x5 resolution meshes respectively.

7.5 Summary

Trilinear projection provides a nonlinear projection with which one can approximate reflections and refractions. Although ray directions are only exactly correct at the vertices, the ray directions are continuous across the projecting surface. This continuity provides a smoother approximation than using linear projection. Accuracy can be increased by tessellating viewing surfaces, though this comes at the cost of performance. The particular performance/accuracy trade off can be varied to suit the scenario.

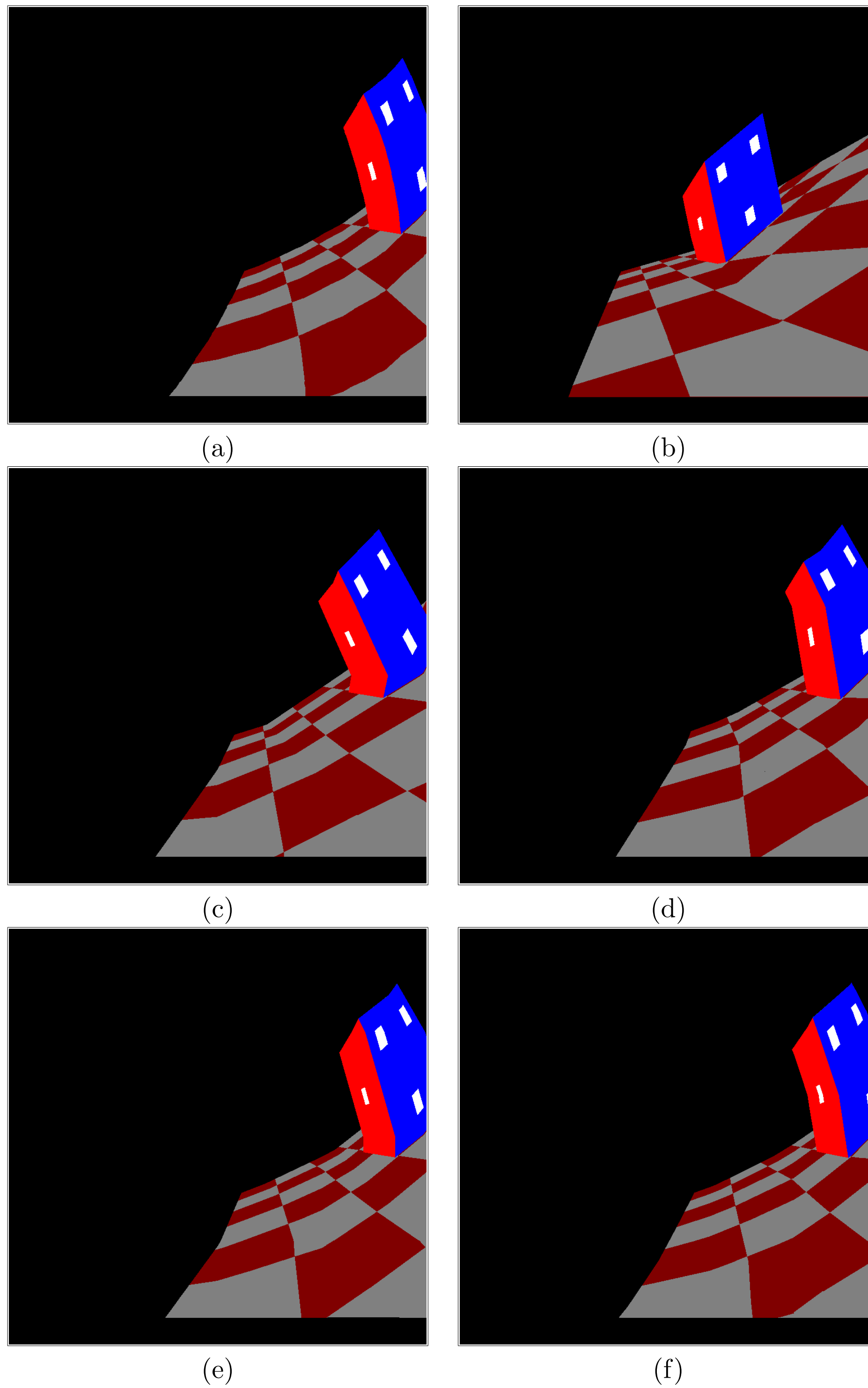
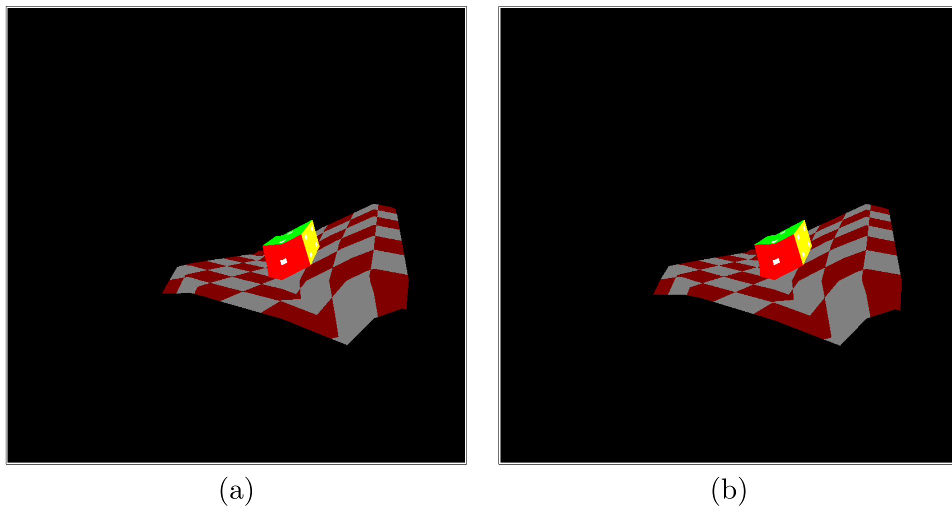


Figure 7.4: A cube reflected on a sphere: (a) ray traced, (b) 1x1 surface, (c) 2x2 surface, (d) 3x3 surface, (e) 4x4 surface, (f) 5x5 surface



(a) (b)

Figure 7.5: A cube reflected on a plane with perturbed normals, implemented as a 5x5 trilinear projection surface: (a) ray traced (b) trilinear projection

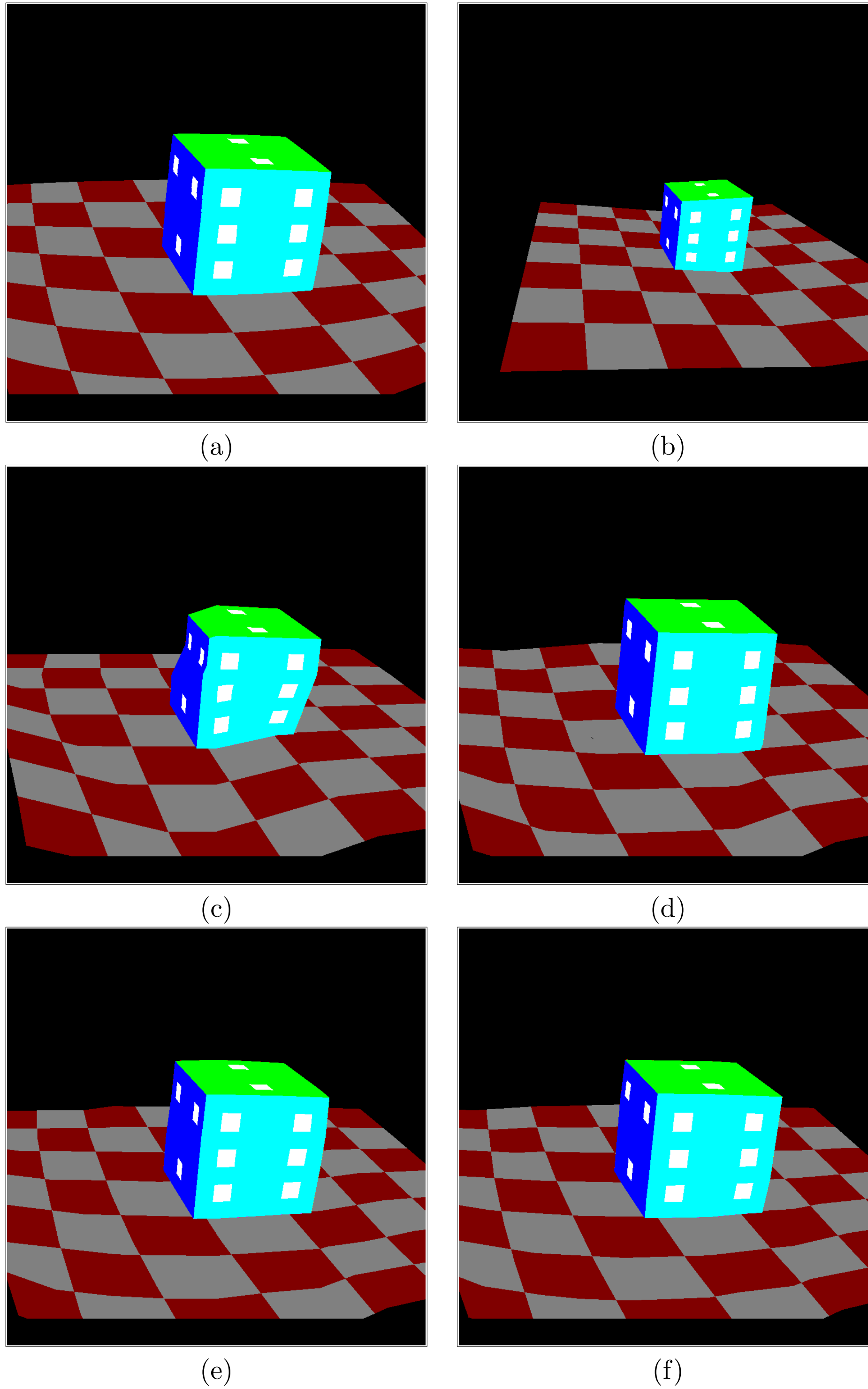


Figure 7.6: A cube refracted through a plane with spherical normals: (a) ray traced, (b) 1x1 surface, (c) 2x2 surface, (d) 3x3 surface, (e) 4x4 surface, (f) 5x5 surface

Chapter 8

Performance Evaluation

This chapter compares the performance of the algorithms developed in this thesis against a ray tracing implementation. Ray tracing provides the simplest method of implementing nonlinear projections, and is considered as the benchmark of visual accuracy, even though ray tracing can suffer from aliasing problems. Most importantly, the measurements presented in this chapter provide an estimate of the algorithm's performance in terms of speed for a given image size, scene and view surface complexity.

8.1 Experimental Conditions

A prototype implementation of the trilinear projection algorithms and a ray tracer was developed using C++ and OpenGL. The prototype can draw single and multiple trilinear projections, with or without clipping and tessellation. The ray tracing algorithm renders trilinear projection surfaces in the manner described in Section 2.4, and some ray-triangle intersection code is adapted from that presented by Buss [Bus03]. Reflective or refractive projection surfaces can be generated automatically, and the ray tracing implementation can render correct reflection and refraction solutions for comparison.

In the prototype code, ray tracing intersections are performed in software and then pixel data is transferred to graphics hardware. With trilinear projection the projection happens in software and then polygons are transferred to graphics hardware for rasterisation.

Execution speeds were averaged across multiple runs. Each run consisted of a render including drawing the requisite pixels to hardware pixel buffers, but did not include swapping buffers to screen. The results here were obtained on a 800 MHz Athlon with a GeForce 2mx graphics card.

8.1.1 Data

Scenes used to measure the algorithm's performance consisted of disconnected triangles with arbitrarily specified vertex colours. No lighting or texture mapping was applied. The results in Figures 8.1, 8.4 and 8.9 to 8.12 were obtained with single scene triangles of different configurations as show in Figures 4.1 to 4.7. These scenes were designed to test the performance of the algorithms over the different possible projected shapes, and to measure the visual accuracy of the tessellation methods.

Results in the other figures in this chapter, except those in Section 8.7, were obtained with random triangles constrained such that some of the scene triangle should be rendered on the screen. Trilinear projection meshes for the random triangle scenes were composed of random points and normals corresponding to a regular mesh on screen space. These scenes were designed to test the performance characteristics of the algorithms over differing scene and trilinear projection mesh complexity. Section 8.7 shows timing results for various example scenes shown in Chapters 6 and 7 of this thesis.

8.1.2 Caveats

The ray tracing implementation used to gather the results in this chapter is naive with respect to its scene organisation. Scene triangles were tested in sequence with no hierarchical organisation. A more sophisticated ray tracing implementation would organise the scene hierarchically so that rays need only be intersected with a subset of the scene. Such techniques can considerably speed up ray tracing so that it is sub-linear in speed with respect to scene complexity. Although not implemented in this prototype, hierarchical scene organisation can reasonably be expected to benefit trilinear projection as well. With multiple trilinear projections each trilinear projection may see only a portion of the scene. Hierarchical organisation of the scene would speed up rendering by making it easier to test if a group of triangles is seen by a particular trilinear projection.

The amount of processing done by the standard central processing unit versus the graphics unit could be varied by different implementations. Ray tracing implementations have been made for current graphics hardware though they are not yet prominent. Although not explored in this thesis, it should be noted that the trilinear projection algorithms could most likely also be fully implemented in current graphics hardware.

8.2 Ray Tracing

Figure 8.1 shows average execution time to render a single scene triangle against image size in square pixels. Each line on the graph corresponds to a single scene triangle rendered from a single trilinear projection resulting in a different configuration as in Figures 4.1 to 4.7. As expected, rendering time increased linearly with resolution. Rendering time was approximately equal for different resulting shape configurations.

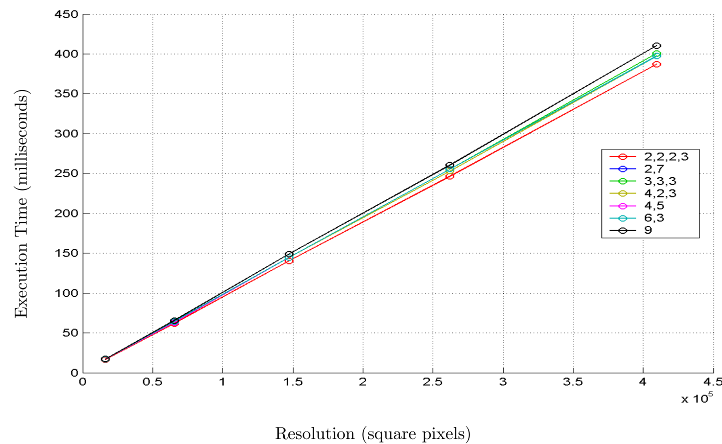


Figure 8.1: Execution time versus resolution for ray tracing different configurations

For scenes with random scene triangles, Figure 8.2 shows that execution time is still linear with resolution. Each line on the graph corresponds to a scene with a different number of random scene triangles.

Figure 8.3 plots the same data set as Figure 8.2 but as execution time versus scene triangles for different resolutions. Each line on the graph corresponds to a different resolution. Execution increased mainly linearly against the number of scene triangles except for the largest scenes, which were slightly slower. The decrease in speed for complex scenes could have been due to a number of factors but most likely was caused by cache misses between memory and processor across the multiple runs.

As expected, this naive ray tracing implementation was linear in execution time against both image size and number of scene triangles. Because ray starting position and normal direction were determined as described in Section 2.4, rendering trilinear projection surfaces with ray tracing should have similar performance to the ray tracing of perspective projections.

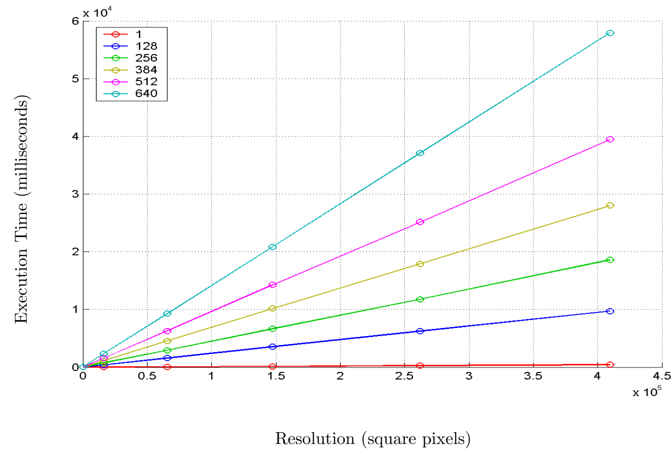


Figure 8.2: Execution time versus resolution for ray tracing across different complexity scenes

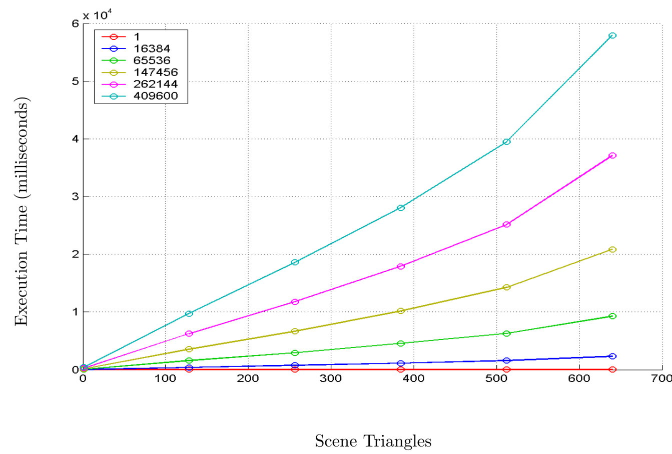


Figure 8.3: Execution time versus number of scene triangles for ray tracing across different resolutions

8.3 Trilinear Projection

Figure 8.4 shows the performance the trilinear projection algorithm under the same conditions as the ray tracing algorithm in Figure 8.1. Configurations with shapes of three or less vertices were rendered faster than shapes with four or more vertices because rendering complex shapes involve potential tessellation as the shapes may self intersect and be non-convex. Most shape configurations were rendered in a constant time against resolution, because the rasterisation of the shapes is quick in comparison to projecting the shapes. The (9) vertex shape configuration became slower with increasing resolution after a certain point because the graphics hardware reached its fill-rate limit.

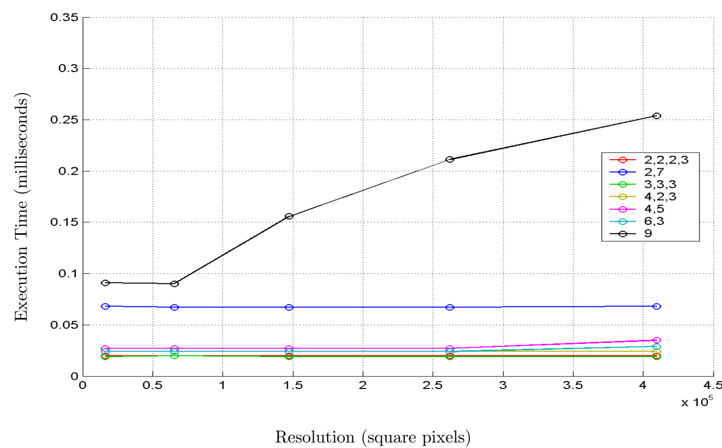


Figure 8.4: Execution time versus resolution for trilinear projecting different configurations

Figure 8.5 shows that execution time versus resolution is constant with trilinear projection until a certain point. Each line on the graph represents a scene of different complexity. At resolutions greater than 1.5×10^5 the fill rate of the graphics hardware is important and execution time became linear with respect to resolution.

Figure 8.6 plots the same data set as Figure 8.5 but with execution time versus the number of scene triangles across different resolutions. Each line in the figure represents a different square scene resolution. The execution time was linear against the number of scene triangles because each scene triangle requires a constant computation time to render.

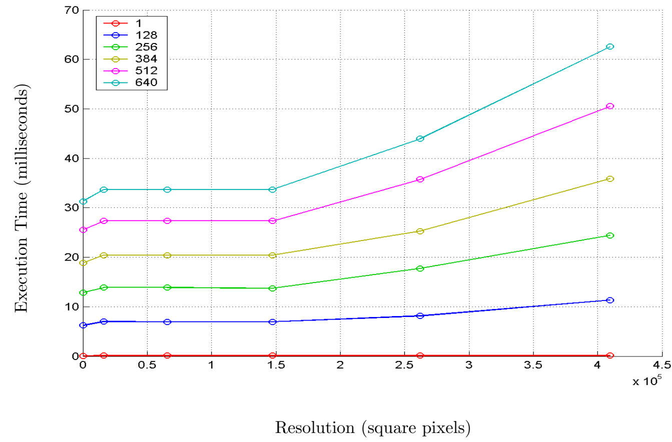


Figure 8.5: Execution time versus resolution for trilinear projection across different complexity scenes

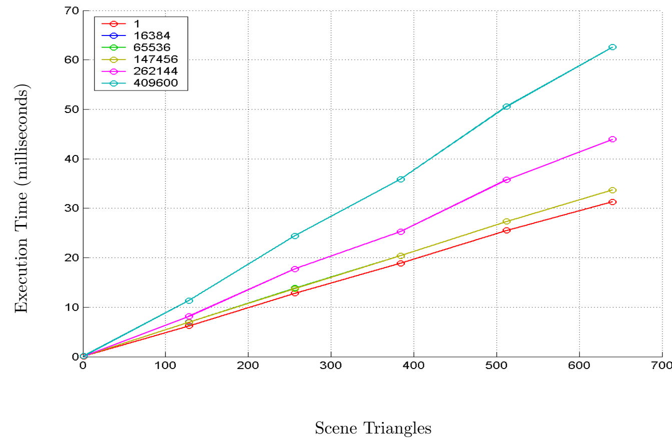


Figure 8.6: Execution time versus number of scene triangles for trilinear projection across different resolutions

8.4 Speed up

The speed up in execution of trilinear projection over ray tracing was calculated as the time to render a given scene using ray tracing divided by the time taken to render the same scene with trilinear projection. Figure 8.7 shows a graph of speed up versus resolution for different single triangle scenes. Each line in the figure represents a different configuration as per Figures 4.1 to 4.7. The more complicated shapes exhibit less speedup than the simpler shapes because the execution time for trilinear projection is less for simpler shapes. Moreover, because the trilinear projection rendering time for a simple configuration was nearly constant, and the rendering time of ray tracing was linear, the speedup versus resolution is linear.

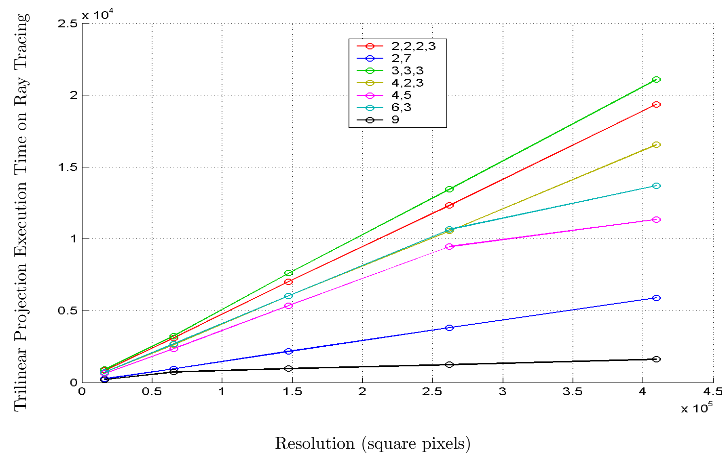


Figure 8.7: Relative speedup versus resolution for trilinear projection across different configurations

Figure 8.8 shows the speed up of trilinear projection over ray tracing across different complexity scenes, for a 1x1 trilinear projection surface. A single triangle scene exhibits a large speedup, but all complex scenes tend toward a speed up of between 500 and 1000 times with increasing resolution.

8.5 Tessellation Methods

The two tessellation methods described in Section 4.5 were evaluated over the different configurations for accuracy and execution time versus tessellation factor.

Visual accuracy was measured by subtracting RGB pixel values of the test image

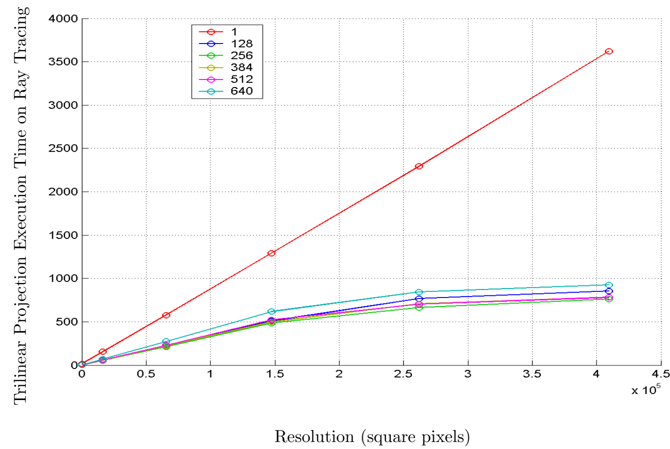


Figure 8.8: Relative speedup versus resolution for trilinear projection across different complexity scenes

from the baseline image, generated by ray tracing, then computing the average absolute value of the error per pixel. Because the RGB values are interpolated across the face of the scene triangles, this technique allows errors to be estimated for more than just the silhouette. The relative error is defined as the average RGB value of the difference mask divided by the average RGB value of the ray traced image.

Figures 8.9 and 8.10 shows that tessellation factor versus error was similar for scene triangle tessellation and parametric triangle slicing techniques over the different configurations. Each line in the plots represents a different configuration.

Figure 8.11 shows the averaged results over the different configurations for both tessellation techniques. Apart from initial conditions that randomly favoured one technique over the other, the error quickly converges to the same value.

However, the two techniques have very different execution time versus tessellation factor graphs as can be seen in Figure 8.12. Scene triangle tessellation takes a super-linear amount of execution time whereas parametric triangle slicing is linear. This follows from the fact scene triangle tessellation exponentially increases the number of vertices with a linear increase in tessellation factor, whereas parametric triangle slicing increases the number of vertices by two times the increase in tessellation factor.

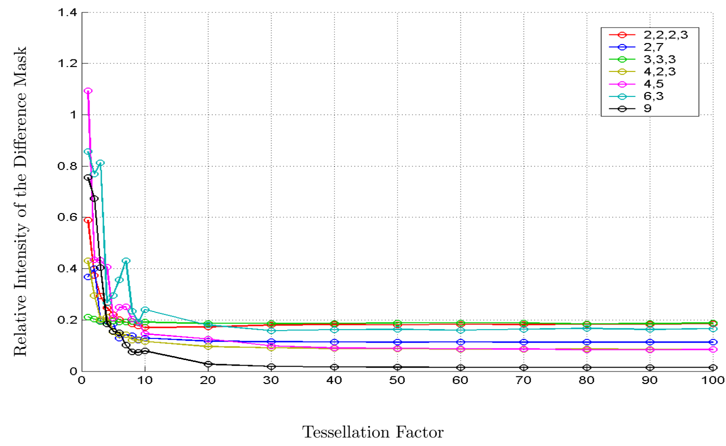


Figure 8.9: Relative intensity of the difference mask versus tessellation factor for parametric triangle slicing over different configurations

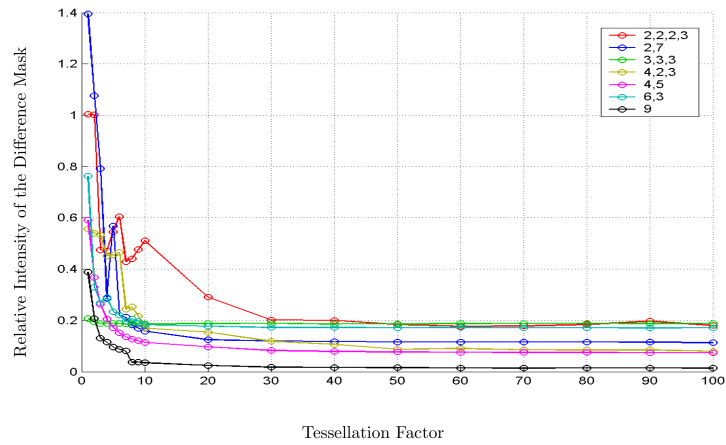


Figure 8.10: Relative intensity of the difference mask versus tessellation factor for scene triangle tessellation over different configurations

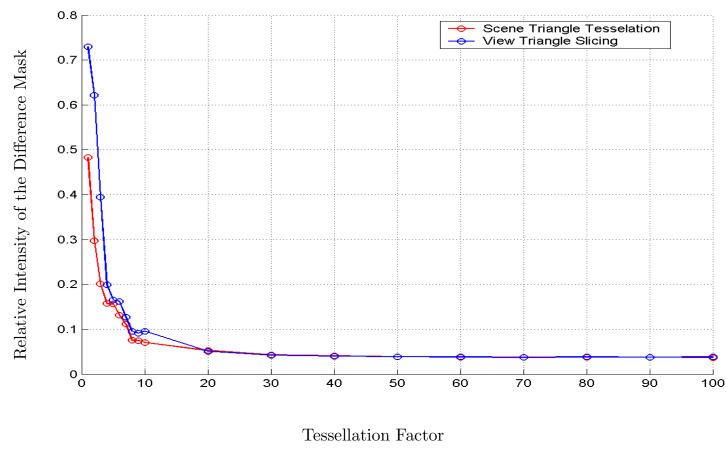


Figure 8.11: Relative intensity of the difference mask versus tessellation factor averaged over each configurations for parametric triangle slicing and scene triangle tessellation

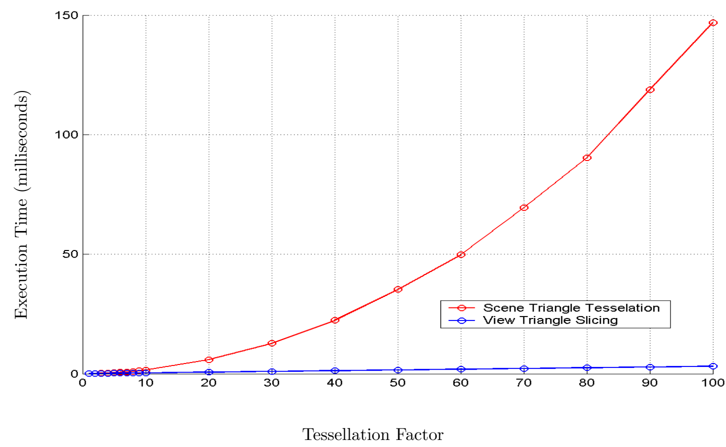


Figure 8.12: Execution time versus tessellation factor averaged over each configurations for parametric triangle slicing and scene triangle tessellation

8.6 Multiple Trilinear Projections

This section details the performance of trilinear projection and ray tracing for multiple trilinear projections. Figure 8.13 shows ray tracing execution time versus the number of trilinear projections. Each line in the plot represents a different resolution for a scene with 640 scene triangles. The figure shows that ray tracing time is essentially constant versus the number of trilinear projections, as each triangle introduces only a very small overhead into the rendering process.

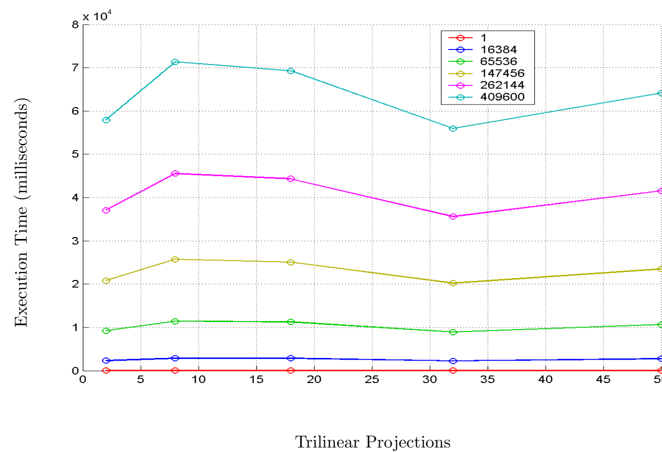


Figure 8.13: Execution time versus number of trilinear projections for ray tracing across different resolutions

Figure 8.14 shows a graph of execution time for rendering the same images with trilinear projection versus the number of trilinear projections. Each line in the plot represents a different resolution for rendering a scene with 640 scene triangles. With trilinear projection each additional trilinear projection effectively requires a re-rendering of the scene. The graph shows that trilinear projection technique is linear versus the number of view triangles, which means the additional amount of work per trilinear projection is constant.

Figure 8.15 shows the speed up of trilinear projection over ray tracing with increasing numbers of view triangles across different resolutions with a 640 scene triangle scene. As expected, the speed up decreases with more view triangles as the resolution per view triangle decreases.

Trilinear projection requires clipping of the view for each view triangle. Figure 8.16 shows the relative cost of the scene-space clipping algorithm described in Section 4.5.2 against the simpler screen-space clipping. This figure shows the slow-down versus number

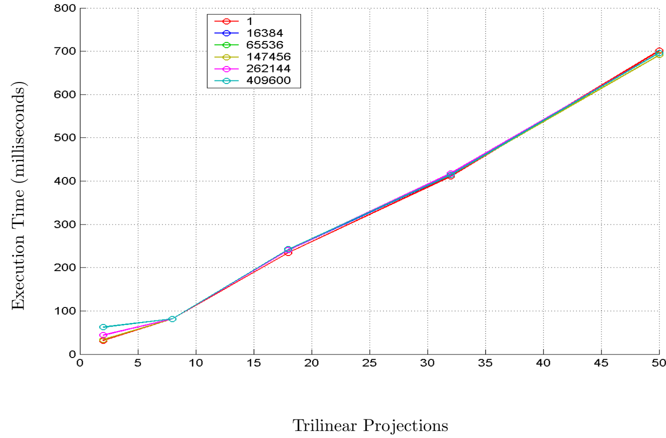


Figure 8.14: Execution time versus number of trilinear projections across different resolutions

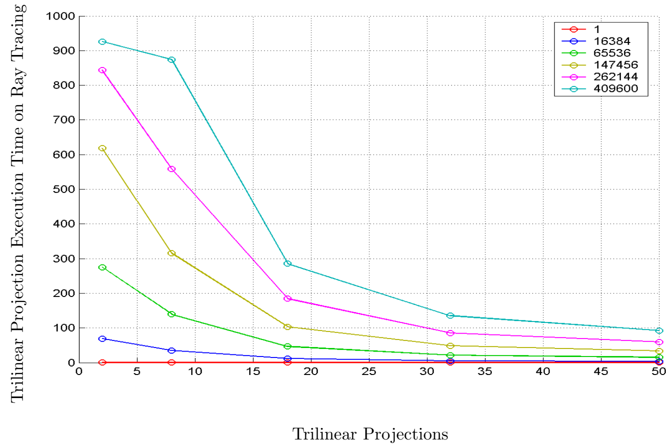


Figure 8.15: Execution time versus number of trilinear projections across different resolutions

of trilinear projections. Slow-down is the execution time for scene-space clipping divided by simple screen-space clipping. Each line on the figure represents a different complexity scene. Apart from the single scene triangle case, the slow-down does not depend on the number of view triangles. The slow-down is initially low with a small number of trilinear projections, then climbs before being linear with respect to trilinear projections. Overall scene-space clipping reduces performance around two times, and the further performance deterioration with increasing numbers of trilinear projections is only slight.

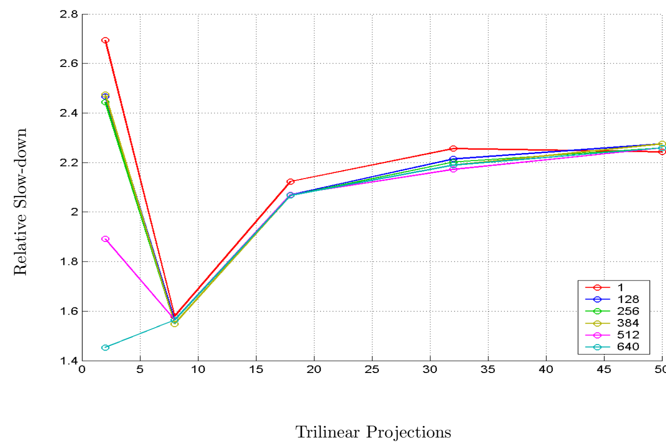


Figure 8.16: Relative execution time for clipped and non-clipped trilinear projection versus number of trilinear projections across different complexity scenes

8.7 Nonlinear Projection for Visualisation, and Reflections and Refractions

For the examples in Chapters 6 and 7 the performance results can be seen in Table 8.1. For all the examples trilinear projection performed many times faster than ray tracing, often with little visual difference between the resulting images.

Figure	Resolution	Trilinear Projections	Scene Triangles	Ray(ms)	Trilinear(ms)	Speed up
6.4	800×800	18	686	226906.0	164.6	1379
6.7	900×300	6	686	92163.0	49.6	1858
6.9	600×600	200	588	40518.0	1242.0	37
7.4 (a)	800×800		686	177926.0		
7.4 (b)	800×800	2	686		24.4	7292
7.4 (c)	800×800	8	686		70.1	2538
7.4 (d)	800×800	18	686		150.1	1185
7.4 (e)	800×800	32	686		230.2	773
7.4 (f)	800×800	50	686		360.7	493
7.5	800×800	50	686	254066.0	410.7	619
7.6 (a)	800×800		686	247576.0		
7.6 (b)	800×800	2	686		18.5	13382
7.6 (c)	800×800	8	686		72.3	3424
7.6 (d)	800×800	18	686		167.0	1482
7.6 (e)	800×800	32	686		287.8	860
7.6 (f)	800×800	50	686		467.3	530

Table 8.1: Execution time for rendering examples in this thesis

8.8 Summary

Trilinear projection provides a method of rendering nonlinear projections with performance characteristics similar to scanline rendering. For a single trilinear projection, drawing scene triangles involves two main additional complications over linear scanline projection: solving the equations in Chapter 3 for each point and turning the results into its constituent polygons, and drawing the resulting polygons, which are often more complicated. For scenes rendered with a single view triangle, trilinear projection offers performance with similar characteristics to linear scanline rendering. For multiple trilinear projections, performance depends on the number of view triangles.

Chapter 9

Related Work

This work on which this research is based is in general split between the two applications of nonlinear projections: reflections on curved surfaces and as a visualisation technique. The two most prevalent rendering algorithms, ray tracing and scanline rendering, each present different ways of realizing such projections. Research pertinent to the two algorithms and particularly to rendering reflections on curved surfaces is presented and examined.

9.1 Ray Tracing

Classical ray tracing is first clearly defined by Whitted [Whi80]. The physics of seeing can be described simply as the process of emitted light interacting with a scene before exciting receptors in our eyes. In its simplest form, ray tracing shoots imaginary light rays out of the eye through the image plane into the scene. The closest intersection between this ray and an object determines the colour the ray contributes to its associated pixel.

The similarity of ray tracing to the physical process of illumination makes it suitable to model many complicated effects. Amongst these effects are shadows, surface properties, reflection and refraction. The disadvantage of ray tracing is generally a larger rendering time than scanline methods.

Recursive ray tracing, as set out in Whitted's work, generates one ray (at least) for each screen pixel. This ray then intersects the scene geometry. From this intersection a ray is shot to each light source, and if the surface is reflective or refractive a reflected or refracted ray is also spawned. The rays shot toward the light source determine if the point from whence they originated is in shadow; if the ray intersects scene geometry before reaching

the source then the point is in shadow from that source. Reflected and refracted rays carry on through the scene (themselves potentially being refracted or reflected) acquiring surface interactions that ultimately affect the end colour of the pixel with which they are associated. In this way ray tracing handles reflections on curved surfaces in exactly the same way as does tracing of eye rays. Once a ray starting position and direction have been determined, be it eye or reflection/refraction ray, its interaction with the scene is resolved in the same manner.

Ray tracing can easily simulate complicated properties of light-surface interaction, reducing them to problems of multiple vectors. At any ray-surface interaction, there are vectors representing input, light source and surface normal directions. Along with surface properties, such as colour and texture, these are the inputs to the function that determines a portion of the colour to be attributed to the ray's originating pixel. This makes a convenient and flexible representation for describing illumination phenomena.

A naive ray tracing implementation makes no attempt to exploit the coherence between rays in a scene. Since the eye rays (which originate from the viewpoint in the scene) all intersect at a common point, this commonality can be exploited as in scanline rendering. For a scene primitive it can be determined which eye ray will intersect it by projecting the scene geometry onto the viewing plane.

Naive ray tracing also suffers from aliasing problems. Rays are infinitely thin, which means sampling edges is very jagged. To overcome this problem, multiple rays per pixel can be cast, or rays with volume (beams, cones or pencils) can be used. In cone tracing, rays are replaced by cones. Each cone can stand for multiple rays (which exploits the coherence between them) and is intersected against the primitives in the scene to determine what it images. Cones have volume so aliasing caused by the point-sampled nature of ray tracing can be eliminated.

9.1.1 Beam Tracing

Heckbert and Hanrahan [HH84] introduced the idea of using beams which are shot into the scene instead of infinitely thin rays. These beams travel into the scene and are clipped against scene objects. Each beam is effectively a perspective transformation, and a beam frustum is a perspective view defined by a starting point and a rectangular frustum. The scene objects are depth sorted and traversed in order. If an object is "seen" by the beam it is subtracted from the beam's 2D scene representation. The representation is initially blank and accepts any object. Additional objects are added in accordance with polygon set

rules, hence hiding objects that are behind the currently held set.

When a beam intersects a reflective or refractive object a new beam is created in a manner similar to ray tracing. The reflected or refracted beam represents the transformation of the incident beam according to the physical laws as much as possible. The generated beam accurately represents the optical path of a reflection on planar surfaces. However, the nonlinear nature of refractions means that the generated beam does not necessarily exactly represent the optical path of all light refracted from the incident beam. Reflections and refractions on curved surfaces cannot be represented accurately by beams.

9.1.2 Spatial Subdivision

The cost of testing intersections between rays and surfaces can be reduced by the prudent arrangement of surface geometry within hierarchical structures. The path through the hierarchical structure for the ray is computed and intersections are only checked for objects in the cells through which the ray passes. These cells can be checked in order, so that when the nearest intersection in the first cell is found, no further cells need be investigated. In Havran *et al's* work [HPP00] different hierarchical subdivision schemes were tested on a standard testbed of geometry from the *Standard Procedural Database* described by Haines [Hai87].

9.1.3 Hardware Ray Tracing

Scanline graphics hardware is currently popular and widely distributed, whereas ray tracing has lacked widely available hardware solutions. As scanline hardware has become increasingly general, ray tracers have been implemented in scanline hardware. The highly parallel nature of ray tracing means that it should be well suited to implementation of Single Instruction Multiple Data (SIMD) hardware. Each different ray path can be computed independently of any other ray path if desired. Purcell [PBMH02] and Carr [CHH02] provide implementations that exploit this coherence on graphics hardware. Though performance in both papers was at best equal to CPU-based implementations, the expectation is that the rate of growth of Graphics Processor Unit (GPU) performance will exceed that of CPU performance in the near future. Historically this has proved to be the case, due in large part to the more parallel nature of the operations performed by the GPU hardware.

Figure 9.1: A conventionally rendered set of columns

9.2 Ray Tracing Nonlinear Projections

Nonlinear projections can be rendered by ray tracing if a nonlinear projection can be expressed as a set of rays. The way in which a nonlinear projection defines a set of rays has been considered by several researchers and some approaches are examined in this section.

9.2.1 Ray Tracing with Extended Cameras

In Löffelmann and Gröller's work [Löf95] the idea of rendering from multiple viewpoints with ray tracing is examined. By developing an extended camera for ray tracing, the authors present multi-perspective images with visualisation as an application. Essentially, an extended camera is a set of 3D rays. Each ray has a starting position and a direction, and these rays are used by a conventional ray tracer to draw the scene. Unlike a conventional ray tracer, these rays do not necessarily originate from the same point.

Figures 9.1 and 9.2, both taken from Löffelmann and Gröller's work, show the extended camera ray tracing in operation. The image in Figure 9.1 is rendered normally; the image in Figure 9.2 is rendered out of a torus camera, where rays all start on the surface of the torus and point in the direction of the surface normal. The extended camera is comprised of three sections: an object space transformer, a picture space transformer and a parameter space transformer. The object space transformer is a function that returns the 3D start position of a ray given its corresponding index on the 2D screen. This function defines a surface in 3D, and the scene is rendered "out of" this surface.

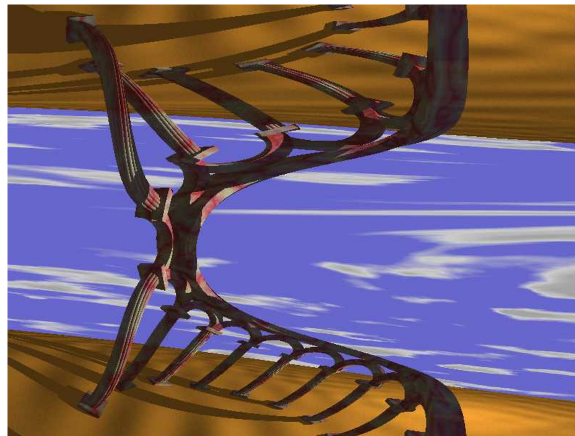


Figure 9.2: Columns rendered from a torus surface

The extended camera generates rays which are rendered with POVray a widely available ray tracing implementation [oVPL04]. This means the implementation handles a wide range of geometry and lighting conditions and the performance is that of a standard ray tracer.

Löffelmann and Gröller's extended cameras are similar to trilinear projection. In their terms, the trilinear projection would be described as an extended camera corresponding to a triangle with trilinearly interpolated normals.

9.2.2 Cubism and Cameras: Free-form Optics for Computer Graphics

Glassner [Gla00] examines nonlinear projections and their expressive power. Nonlinear projections provide storytelling tools and can compose images that are not as limited by the geometry of the scene. As an example, Glassner suggests a scene where there are three characters of interest: a boy and girl in separate houses and a brother in the road between them. These three characters can be imaged with nonlinear projection without juxtaposing disjoint images together; a hand-drawn version is shown in Figure 9.3.

Glassner proposes that the geometry of the nonlinear projection can be defined by two surfaces: an eye surface and a lens surface. These surfaces are NURBS (Non-Uniform Rational B-line Spine) patches that define ray starting positions and direction vectors. Each pixel maps to a point on both surfaces and rays are shot from the eye surface through the corresponding point on the lens surface. Generating eye and lens surfaces to suit particular artistic goals is not an intuitive process and a collage approach is suggested as an alternative.

Figure 9.3: A hand-drawn nonlinear projection of a street scene [Gla00]

The collage approach ties section of the nonlinear projection to standard rendered views, and intermediate areas are interpolated with ray tracing. This approach is similar the trilinear projection technique examined in Section 6.2.

9.2.3 Multi-Perspective Images for Visualisation

In earlier work [VC01b], see Appendix F, we present an interface for specifying nonlinear perspectives based upon the OpenGL library [SA98]. Nonlinear surfaces are specified by defining vertex and normal directions for each ray in a nonlinear projection or by specifying other common scene primitives such as triangles, and rendered using a ray tracing implementation. Figure 9.4 shows a cube rendered from a hemisphere surface above the scene, with the ray positions and directions defined by a Bezier surface. Specifying a nonlinear projection using a common scene primitive such as a triangle was the inspiration for the trilinear projection described in this thesis.

9.2.4 General Linear Cameras

Yu and McMillan [YM04] introduce General Linear Cameras (GLC) as a mathematical description for a class of nonlinear projections defined by three rays passing through two parallel planes. The authors define and name various special cases of these cameras, and implement them with ray tracing and a light field rendering system.

The trilinear projection defined in this thesis is less constrained than a GLC because it is not defined by the intersection with two parallel planes. A trilinear projection is a GLC if the trilinear projection vectors have magnitude in the direction normal to the view-

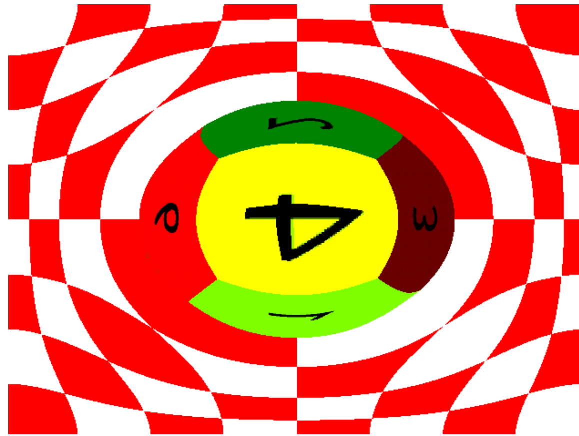


Figure 9.4: A cube rendered from a hemisphere surface [VC01b]

ing triangle. Therefore, the trilinear is a superset of the GLC. However, if the lengths of the three normal vectors of a trilinear projection are varied it results in a different projection.

9.3 Scanline Rendering

In its simplest terms, scanline rendering involves a perspective transformation distorting the scene data to represent the effect of the virtual camera, and the sampling and quantization of the scene data to a 3D grid (the screen buffer and the z-buffer).

Like ray tracing, scanline rendering can take advantage of techniques that remove hidden geometry early. Occlusion culling techniques have been developed for scanline methods which remove unseen geometry before it is drawn to the z-buffer. These techniques usually involve spatially subdividing the scene and organising it into a hierarchy. However, the benefit is not so great as in ray tracing, because each ray must traverse into the scene, whereas in scanline rendering the scene is traversed only once.

9.3.1 Multi-Pass Rendering

In Diefenbach's thesis [Die96] multi-pass methods for increased graphical realism are presented. Specular reflections and reflections, shadows, and global illumination are incorporated into a rasterising framework. Reflections on planar surfaces are rendered from a virtual viewpoint (the viewpoint as reflected by the plane of reflection) and incorporated into the image as an additional pass. This is very similar to beam tracing as described in Section

9.1.1, although using z-buffer hidden surface algorithms rather than set intersections.

While correctly handling reflections on planar surfaces the technique does not handle reflections on curved surfaces. However it does provide a general framework by which reflections and refractions can be drawn by different projections and incorporated into the frame.

9.3.2 Reflections on Spheres and Cylinders of Revolution

Glaeser [Gla99] presents equations for calculating the reflection of a space point on a sphere or cylinder of revolution. The solution to the reflex is an algebraic equation of order four, which comes from the fact that both cases can be reduced to the 2D reflection on a circle.

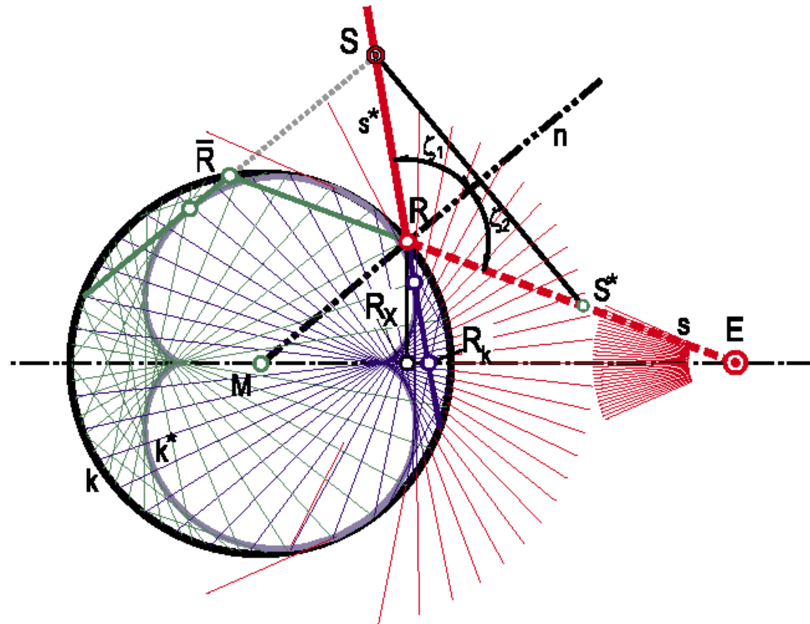


Figure 9.5: Catacaustic of the reflection congruence [Gla99]

Geometrically the problem can be seen in Figure 9.5, taken from Glaeser's work [Gla99]. S is the point that is reflected by the circle (centre at M), as seen from viewpoint E. A ray shot from the eye at E reflects off the circle at point R. Calculating the roots of an order four polynomial gives four values of E. Each value of E may be valid. Lines and polygons reflected on spheres and cylinders of revolution can be drawn by subdividing the line or polygon sufficiently. Higher order reflections, such as reflections on spheres reflected on spheres for instance, are not handled by this work. For a limited subset of geometry this

work provides an exact solution to projecting geometry onto reflective objects.

9.3.3 Multiple-Center-of-Projection Images

Rademacher and Bishop [RB98] presents generalised multi-perspective images as the basis for resynthesis, the advantage being a variable level of sampling without multiple separate images. The paper calls these Multiple-Centre-Of-Projection images. One section of the panorama can be from close to a portion of the image, giving a high sampling for that area, while others are further away and capture more of the object. Moving a virtual camera through the scene generates the multi-perspective panorama. At regular intervals the camera captures a single line of pixels for the final panorama. These lines, either rows or columns, are placed next to each other, so that the viewpoint smoothly changes from one to the next. This is effectively a virtual strip camera, as detailed in Section 1.1.2. Figure 9.6 shows a Multiple-Centre-Of-Projection image of an elephant. The virtual camera path goes from one side of the elephant to the other, so that the image simultaneously shows both sides and the front.

Generating a Multiple-Centre-Of-Projection is not designed to be done dynamically. The generation of such images takes time of the order of the time taken to scanline render the scene multiplied by the number of single-pixel-width images captured. The description of camera path closely matches the model of extended camera detailed in Section 9.2.1. The idea extends beyond the simple idea of a strip to that of a full surface, though automatic generation of images with such surfaces is not examined.

9.4 Object Distortion for Nonlinear Projections

Object distortion methods are similar to perspective distortion techniques, except that they deform geometry to approximate arbitrary nonlinear projections. Once deformed, the geometry can be rendered by standard techniques, usually scan line rendering. Previous research in distortion methods for visualisation do not describe distortions in terms of nonlinear projections, but do give general motivation for the distorted viewing of data.

9.4.1 Distortion Methods for Visualisation

The fundamental problem distortion-oriented techniques solve is the “detail and context” problem. This problem occurs when visualisations need to present information-rich detailed views concurrently with sparser context views. A classic problem in the domain is that of a

Figure 9.6: A nonlinear projection of an elephant [RB98]

street map. Seeing street-level information in sufficient detail to traverse streets precludes seeing enough of the street map to navigate larger distances.

Leung [LA94] presents a review and taxonomy of distortion-oriented techniques particularly for text and 2D graphics. He proposes that various continuous 2D distortions for visualisation can be subsumed under the technique of 'rubber sheet' warping. This warping stretches data in a manner analogous to stretching a rubber sheet, and can express continuous 2D deformations.

Distortion-oriented display ideas are extended to 3D data by Winch *et al* [WCS00] and Carpendale [CCF97]. In Winch *et al's* work, regions of interest can be zoomed in 3D. These regions expand space around a focal point, distorting close data, and gradually falling off as distance increases. Winch *et al* also explores [WCS01] occlusion around regions of interest and transparency of occluding data.

Carpendale [CCF97] explores techniques for displaying both detail and context within 3D graphs. This work is particularly concerned with the issue of occlusion, where a particular graph node may be obscured by the nodes in front of it. The authors propose techniques for nonlinear expansion of space around nodes of interest and for moving occluding nodes. Nodes that occlude a node of interest are moved out of the way by a nonlinear region of expansion along the line between the eye point and node of interest.

9.4.2 Interactive Reflections on Curved Objects

The most intriguing approximation of rendering reflections on curved surfaces is described by Ofek and Rappoport [OR99]. In this method, objects are transformed by the reflective surface so that they may be rendered from a single viewpoint. In essence the data is distorted to an approximation of how it will look after being viewed from a reflective surface, and then rendered. The technique requires an appropriate tessellation of both reflective surface and scene object to approximate the curvature of lines reflected with curved surfaces. The performance of this technique is sufficient for real-time rendering of moderate scenes. The technique works on standard polygon scenes making it suited for visualisation tasks, and allowing easy integration into current applications.

The correspondence between a point in space and the reflective surface is approximated by two "explosion maps", one close to the reflective surface and one distant, both centred on the reflective surface. An explosive map is a projection of the reflective triangles out along their normals onto the surface of a sphere. A point in space is then projected onto both spheres, through the centre, and averaged. The reflective triangle on which the point falls is assumed to be the reflective triangle which 'sees' the point. The coordinate at which the projected space point intersects the reflective triangle on the explosion map is used to distort the space point to its projected position.

An explosion map is an approximation of the geometry of the reflection mapping. The mapping of scene point to the explosion map through the centre is only accurate if the reflection vectors of the surface happen to pass through the centre. This approximation back to a centre is similar to the way in which environment mapping approximates all reflection rays back to a centre. With two explosion maps, one near and the other far, the error is probably lessened but the extent of the improvement is unclear. The explosion map approximation generates a single corresponding point for each scene point, even if the that point would be reflected by multiple places on that surface.

9.4.3 Specular Path Perturbation

Specular path perturbation, detailed by Chen and Arvo [CA00], approximates reflections with perturbation methods. Given a scene point and view point the technique determines which point on the reflective surface images that scene point. From a set of known reflection, view, and scene point correspondences, similar reflection correspondences can be approximated with second-order derivatives. For scene objects imaged multiple times by the reflective surfaces, the technique does not necessarily connect the constituent reflected

vertices correctly.

The approximate nature of the calculation makes it fast without losing too much accuracy compared with ray tracing. However, the set of known reflections needs to be pre-calculated, making the technique unsuited to dynamic scenes, though dynamic shading can be added to the reflected images.

9.4.4 Region of Influence Cameras

Singh [Sin02] present an algorithm and interactive tool for creating nonlinear perspectives. The concept is based around the idea of multiple cameras. Each camera has a position and direction as well as a center of interest. Any given scene point is projected by a weighted sum of the individual camera transformation matrices and their view-port parameters. This is an efficient way to distort the data based on certain constraints. The weights are mainly determined by a combination of two factors: positional influence and directional influence. Positional influence biases weights higher for cameras whose center of interest is near the scene point. Directional influence biases weights higher for cameras whose central line of projection is near to the scene point.

The interactive and intuitive nature of the camera specification makes it suitable for certain artistic goals. Fundamentally this technique forms a distortion of the data. However, no scene point may be imaged by multiple cameras whereas general nonlinear projections may involve imaging the same point multiple times, especially when implementing reflections and refractions.

9.5 Approximating Reflections on Curved Objects with Image Based Rendering

Reflections on curved surfaces are a natural form of nonlinear projection. Computer graphics has long been interested in realistically rendering scenes including reflections. However the problem of computing reflections on curved objects for artistic purposes is simpler then that of computing true nonlinear projections. In particular, artistic reflections need not be entirely accurate, and often they only reflect static geometry, and hence can make use of pre-rendered data. Additionally they are often assumed to be small in relation to the rest of the screen, lessening accuracy requirements and making per-pixel costs less important. In contrast, nonlinear projections for visualisation need to be dynamic, geometrically accurate and generally fill the screen. Nevertheless, research into generating reflections on

curved objects, especially for real-time graphics, shows how nonlinear projection can be implemented.

Image-based rendering techniques for reflections and refractions often render to textures that are used in a scan-line rendering system to draw the images on the reflective or refractive objects.

9.5.1 Environment Mapping

Environment mapping was initially suggested by Blinn and Newell [BN76], and is often used to approximate curved reflections for real-time graphics. When a scene is rendered from a particular viewpoint, the light coming into that point is sampled. With enough samples for a particular viewpoint, the technique can approximate the colour of any ray shot from that point. A popular implementation of environment mapping involves rendering to the six faces of a cube centred on a point. When used for generating reflections, this centre point is the centre of the reflective object. For each ray that bounces off the reflective object, a ray is shot from the centre point in the direction of the reflected ray into the cube. In effect, this approximation moves each reflected ray to the centre of the cube. For scenes where the reflected objects are far away from the reflective object the technique works well; otherwise the approximation is obvious. For general nonlinear projections, environment maps may not be sufficiently accurate. The light is sampled at only one point, making it unsuited for nonlinear projection such as described in Chapter 6.

In other work [VC03], see Appendix G, we describe a novel extension of environment mapping that renders images from the outside of an object toward its centre. The technique relies on an anti-perspective projection, which reverses the standard depth buffer test, rendering closer objects as smaller than distant ones. These anti-perspective projections can be combined into a single view when used as an environment map, and can render images as if from a sphere surrounding the centre point.

9.5.2 Extended Environment Mapping

An extension to the concept of environment mapping is proposed by Cho [Cho00] as a way to provide more accurate images. Instead of simply sampling the light coming into a point, a depth-mapped image is calculated for each of the six cube faces of the environment map. Reflection rays can then be calculated from the 3D depth map, without needing to approximate the start point of the ray. This technique amounts to rendering the scene through a modified representation of the geometry (the depth map) that provides a

significant performance enhancement in some cases.

In the case that the rays being traced are distant from the point at which the extended environment map was generated, artifacts can be introduced. The depth map representation of the scene does not contain full information about objects that are at least partially occluded from the point of view that the environment map is made. If a ray is likely to be incorrect, the technique defaults to a ray tracing implementation. If a significant number of rays are likely to be correct this may still improve performance by reducing the amount of ray tracing needed.

For dynamic scenes the environment map needs to be generated each frame and then used to approximate the reflection. This adds the additional performance cost of rendering the scene using scanline rendering for each of the faces of the extended environment mapped cube.

9.5.3 Parameterized Environment Maps

Another technique based on environment mapping is presented by Hakura *et al* [HSL01]. In this technique layers of environment maps are used. Different environment maps may be used and layered according to the viewer's location and direction of view, to avoid the limitations of standard environment maps. For instance, close objects and self reflections could be captured by different maps to distant objects.

The environment maps are generated by ray tracing from a particular view point. Distant and local geometry is captured in two distinct maps. The maps themselves are decomposed into separate diffuse, Fresnel modulation and incident specular layers for a more accurate representation of the scene. The final parameterized environment map is created by inferring an environment map using a least-squares best-fit match between the image when rendered with the environment map and the ray-traced reference image. The parameterized environment maps do not simply equate to a projection from a single centre point, making them more able to capture reflections accurately. The environment map can be parameterized across any variable, such as the position of objects and lights.

The algorithm presents good results for static scenes with a changing view point. It is unclear how many samples would be needed to capture a dynamic scene, even if the motion in the scene is pre-determinable. Each view point sample requires a full ray-tracing pass and inference, making dynamic generation unhelpful over simply ray tracing the view. The work presents an image-based rendering technique that is particularly suited to rendering reflections and integrates well with current graphics hardware, but it has a

significant offline rendering cost. The technique does not easily lend itself to rendering general nonlinear projections.

9.5.4 Light Field Rendering

Image-based light sampling techniques such as the Lumigraph [GGSC96] can be useful for rendering reflections. These techniques represent the light leaving a convex hull in a scene, and are sampled with 2D slices to generate a particular view. Essentially, light sampling techniques map incoming ray directions and positions to colour values. The application of this technique for reflections and refractions is explored by Heidrich *et al* [HLCS99]. In this work a mapping between incoming rays and outgoing reflected or refracted rays is represented in a light-field-style structure. The outgoing rays can then be used to index another light field or environment map to determine final colour values. The densely sampled nature of the light field techniques means they are unsuited to dynamic scenes and have large memory overheads.

9.6 Summary

This chapter presents related research in ray tracing, scanline rendering and image-based rendering as it relates to the implementation of nonlinear projections. Ray tracing has previously been used to implement nonlinear projections, both explicitly and as reflections and refractions on curved surfaces. The manner in which a nonlinear projection defines the set of rays to trace differs across the literature and is usually directed by how the nonlinear projection is created.

Scanline methods can use multi-pass rendering to include effects such as planar reflections into a single image, and this technique can be expanded to incorporate nonlinear projections that approximate reflections and refractions on curved surfaces. Object distortions have been explored previously as a visualisation tool and as a way of approximating nonlinear projections. As a visualisation concept "detail and context" provides a framework for the nonlinear viewing of data to better suit user interactions. Methods to approximate nonlinear projections by distortion are similar to the algorithms developed in this thesis, except that they do not necessarily define a clear solution to how a multiply imaged scene point should be projected.

Image-based rendering techniques are the current standard for real-time approximation of reflections and refractions on curved objects. The simplest solution, environment

mapping, is both the fastest and the least geometrically accurate. Reflection or refraction rays are approximated as coincident. Variations to the theme of environment mapping provide increased geometric accuracy at the cost of performance. The image-based methods all rely on extensive pre-calculation and so do not lend themselves to dynamic scenes.

Chapter 10

Conclusion

This chapter summarises the findings of the thesis and views the main contributions of this work, the trilinear projection, in context. Further work directions are briefly discussed, with the main focus on how to take the algorithms presented here to a production environment.

10.1 Summary

In summary this thesis has developed algorithms to project scene points and scene triangles with a trilinear projection surface. The trilinear projection as a basis for rendering nonlinear projections is congruent with rendering curved surfaces in current computer graphics. It offers continuity across curved projection surfaces and handles scenes where the data is close to the surface as well as distant. Supporting algorithms to tessellate solutions for greater accuracy and to cull scene data behind the view plane are provided. Multiple projection surfaces can be used to build more complicated projection surfaces, and algorithms to integrate solutions across these projection surfaces are developed. Trilinear projection surfaces are demonstrated for visualisation and for reflections and refractions. Preliminary performance characteristics are presented to gauge when trilinear projections are likely to offer a computational advantage over ray tracing solutions.

10.2 Contributions

The key contributions of this thesis are four algorithms for trilinear projection of 3D scene data: projecting a point with a trilinear projection, reconnecting a scene triangle's vertices

once projected, parametric triangle slicing to reduce error, and edge clipping in scene space. The applications of the technique are examined in two areas. First, trilinear projection is demonstrated as a method for implementing nonlinear projections for visualisation. Second, reflections and refractions on curved surfaces are approximated by the trilinear projection.

10.2.1 Projecting a Scene Point with Trilinear Projection

In implementing nonlinear projections the mapping between scene points and the screen becomes more complex. In many nonlinear projections this means that analytical solutions to the problem of determining how a scene point is imaged do not exist. A naive expression of the trilinear projection geometry makes for a nonlinear system of equations in solving the mapping problem. After restating the geometry of the trilinear projection in terms of a parametric triangle, Chapter 3 shows how the problem can be reduced to finding the roots of a cubic polynomial, which can be found analytically. The algorithm calculates a value for the parameter such that the generated triangle is coplanar with the scene point. As it is likely that there will be many scene points to be imaged by each trilinear projection, an optimisation is presented for this case. This optimisation calculates partial values of the cubic's coefficients for a certain trilinear projection and then reuses them over multiple scene points. The analytical solution to the problem of projecting a scene point with a trilinear projection makes a scanline style algorithm possible for rendering nonlinear projections.

10.2.2 Projecting a Scene Triangle with Trilinear Projection

A nonlinear projection such as the trilinear projection can image a single scene point more than once. For a scene triangle each of the three vertices may be imaged up to three times by a trilinear projection. Reconnecting these projected vertices may generate from one to four shapes of from two to nine vertices. Chapter 4 details an algorithm to connect these vertices based upon the order of their traversal by the parametric triangle. The algorithm iterates through a list of sorted vertices and at each vertex toggles a state variable representing which two edges of the scene triangle are active. An extension to the algorithm sorts the vertices into shapes that are edge ordered. Each vertex in an edge ordered shape is connected to the vertices before and after it. These edge ordered shapes can be rendered with standard rasterising algorithms.

10.2.3 Parametric Triangle Slicing

A nonlinear projection does not necessarily map straight lines in scene space to straight lines in screen space. Projecting, connecting and rendering scene primitives is inherently an approximation of how a shape should be drawn by a nonlinear projection, because the lines between the projected vertices are assumed to be straight. To alleviate this error, scene primitives may be tessellated into smaller shapes. This tessellation can lead to an significant increase in the amount of computation needed for each primitive. Section 4.5.2 presents an alternative approach that samples the scene primitive with additional values of the parameter of the parametric triangle between vertex solutions. For each of these in between parameter values a line across the scene primitive is sampled, giving two extra shape vertices. The shapes are then drawn incrementally so the sampled line is drawn as a line in the projected shape. The algorithm provides an equivalent increase in rendering quality with a much smaller performance cost than tessellating the scene primitives.

10.2.4 Scene Space Clipping

Meshing trilinear projections together gives a way to approximate more complicated nonlinear projections in a manner similar to approximating a curved surface with a polygonal mesh. Projected shapes may not be continuous across trilinear projections because of the linear approximation between projected shape vertices. To obviate this problem a scene-space clipping algorithm is developed in Chapter 5. The edges of the trilinear projection are swept out into the scene and the intersection between these edges and the scene primitives calculated. These intersection points can then be used to either clip or augment the projected shape so that continuity across composite trilinear projections is preserved.

10.2.5 The Application of Trilinear Projection in Visualisation

Chapter 6 also briefly explores how trilinear projection can be used for visualisation. Previous work in distortions and projections including detail and context with distortion-oriented displays are adapted to nonlinear projections and implemented with trilinear projection. A nonlinear projection that includes views from multiple perspectives and seamlessly transitions between is implemented with trilinear projection. The mapping of a sphere to a flat screen is examined with a nonlinear projection, itself a sphere facing inwards, and implemented with trilinear projection.

10.2.6 The Application of Trilinear Projection in Rendering Reflections and Refractions on Curved Surfaces

Reflections and refractions on curved surfaces form natural nonlinear projections. The geometry of reflections and refractions allows the mapping of incoming vectors into reflected or transmitted vectors. Chapter 7 shows how calculating these transmitted or transformed vectors at the vertices of the reflective or refractive object generates a nonlinear projection that approximates the reflection or refraction. For a reflective or refractive object represented by a triangular mesh, a trilinear projection can be generated to approximate the reflection or refraction from each facet. Rendering from these trilinear projections is an approximation that can be improved by increasing the number of trilinear projections per reflective or refractive object. Reflections and refractions of a simple scene are demonstrated with trilinear projection.

10.3 Further Work

The algorithms in this thesis have not been fully applied to a real world problem. Applying them to a production system would require analysis and balancing of issues such as which methods to use and when. Even for a single trilinear projection tessellation can be used to improve visual accuracy at the cost of performance, but when and how much to tessellate remains an unsolved problem.

The number of trilinear projections required to sufficiently approximate arbitrary curved projection surfaces is unexplored, but is an important factor in the rendering time of the algorithms in this thesis. This is especially pertinent for reflections and refractions on curved surfaces because each trilinear projection is itself an approximation of the light interaction.

Numerical instability in converting to barycentric coordinates means that an appropriate tolerance level depends on the data. Linear projections require that data lie in a frustum with near and far values such that scene data can be adequately scaled to avoid instabilities. Scaling could be used to avoid instabilities in a trilinear projection, but a mechanism analogous to a view frustum is not readily apparent.

The concept of projecting a surface out along a parameter, in this thesis a parametric triangle, to resolve a projection can be expanded to more complicated surfaces. For instance, the reflection or refraction of light on a curved surface can be characterised by the sweep of a parameterised surface. Computing the value of the parameter which satisfies

the implicit equation for the surface at a particular point leads to the reflected location of that point. Determining the implicit equation for the reflection or refraction off a particular surface is an unsolved problem, and it is unclear how the value of the parameter could be determined for complex surfaces.

The applications of nonlinear projections for visualisation is a relatively unexplored field. Unusual views of data, though they can be initially confusing, allow for more viewing constraints to be satisfied in a single image. Detail and context, multiple viewing directions, and other combinations of viewing constraints can be continuously joined but the efficacy of such a view for achieving any particular visual task is unexamined.

10.4 Conclusion

The problem of rendering nonlinear projections with scanline style algorithms has not previously been thoroughly examined. This thesis presents a trilinear projection algorithm that projects scene points and triangles in a manner compatible with scanline rendering algorithms. The performance characteristics of scanline rendering have made it very useful in interactive and animated computer graphics, and algorithms compatible with scanline rendering have a substantial base of hardware and software to draw upon. A limitation of scanline rendering has been its difficulty in modeling certain complicated optical interactions. The trilinear projection algorithm provides a new technique for rendering optical interactions that presents developers with more tools to render unusual visualisations and optical phenomena with acceptable performance.

Appendix A

Expanded Equations

A.1 Parametric Triangle and Scene Point Coplanarity Test

The coefficients of the cubic used to test a scene point and parametric triangle coplanarity are a, b, c and d where $at^3 + bt^2 + ct + d = 0$. The full expansion of these coefficients in terms of the triangle projections points $p_{1..3}$ and normals $n_{1..3}$, and the scene point p_s is as follows:

$$a = n_{1x}n_{2y}n_{3z} - n_{1x}n_{2z}n_{3y} - n_{2x}n_{1y}n_{3z} + n_{3x}n_{1y}n_{2z} - n_{3x}n_{1z}n_{2y} + n_{2x}n_{1z}n_{3y}$$

$$\begin{aligned} b = {} & n_{2x}p_yn_{3z} - n_{3x}n_{1y}p_z + n_{3x}n_{1y}p_{2z} - n_{1x}p_{2z}n_{3y} + n_{1x}n_{2z}p_y - n_{1x}n_{2z}p_{3y} + \\ & p_{1x}n_{2y}n_{3z} + n_{1x}p_zn_{3y} - p_{1x}n_{2z}n_{3y} - n_{3x}n_{2z}p_y + n_{3x}p_{1y}n_{2z} + p_xn_{1z}n_{2y} - \\ & p_xn_{1y}n_{2z} - p_{3x}n_{1z}n_{2y} + p_{3x}n_{1y}n_{2z} + n_{3x}n_{2y}p_z - p_{2x}n_{1y}n_{3z} - n_{2x}p_zn_{3y} + \\ & n_{2x}p_{1z}n_{3y} - n_{3x}p_{1z}n_{2y} + n_{3x}n_{1z}p_y - n_{3x}n_{1z}p_{2y} - p_xn_{1z}n_{3y} + p_xn_{1y}n_{3z} + \\ & p_{2x}n_{1z}n_{3y} - n_{2x}n_{1z}p_y + n_{2x}n_{1z}p_{3y} - p_xn_{2y}n_{3z} + p_xn_{2z}n_{3y} - n_{2x}n_{1y}p_{3z} - \\ & n_{2x}p_{1y}n_{3z} + n_{2x}n_{1y}p_z - n_{1x}p_yn_{3z} + n_{1x}p_{2y}n_{3z} - n_{1x}n_{2y}p_z + n_{1x}n_{2y}p_{3z} \end{aligned}$$

$$
\begin{aligned}
c \;=\; & -p_x p_{1z} n_{3y} - p_{1x} n_{2y} p_z + p_{1x} n_{2y} p_{3z} + n_{1x} p_z p_{3y} + n_{1x} p_{2z} p_y - n_{1x} p_{2z} p_{3y} + \\
& p_x p_{1z} n_{2y} + p_{1x} n_{2z} p_y - p_{1x} n_{2z} p_{3y} - p_{1x} p_y n_{3z} + p_{1x} p_{2y} n_{3z} - n_{3x} p_{2z} p_y - \\
& n_{3x} p_{1y} p_z + n_{3x} p_{1y} p_{2z} + p_x n_{1z} p_{2y} - p_x p_{1y} n_{2z} - p_x n_{1y} p_{2z} + p_{3x} n_{2y} p_z - \\
& p_{3x} p_{1z} n_{2y} + p_{3x} n_{1z} p_y - p_{3x} n_{1z} p_{2y} - p_{3x} n_{2z} p_y + p_{3x} p_{1y} n_{2z} - p_{3x} n_{1y} p_z + \\
& p_{3x} n_{1y} p_{2z} + n_{3x} p_{2y} p_z + n_{3x} p_{1z} p_y - n_{2x} p_z p_{3y} - n_{2x} p_{1z} p_y + n_{2x} p_{1z} p_{3y} - \\
& n_{3x} p_{1z} p_{2y} - p_x n_{1z} p_{3y} + p_x p_{1y} n_{3z} + p_x n_{1y} p_{3z} - p_{2x} p_z n_{3y} + p_{2x} p_{1z} n_{3y} - \\
& p_{2x} n_{1z} p_y + p_{2x} n_{1z} p_{3y} + p_{2x} p_y n_{3z} - p_{2x} p_{1y} n_{3z} + p_{2x} n_{1y} p_z - p_{2x} n_{1y} p_{3z} + \\
& p_{1x} p_z n_{3y} - p_{1x} p_{2z} n_{3y} + n_{2x} p_y p_{3z} - p_x n_{2y} p_{3z} + p_x p_{2z} n_{3y} + p_x n_{2z} p_{3y} - \\
& p_x p_{2y} n_{3z} - n_{1x} p_{2y} p_z - n_{1x} p_y p_{3z} + n_{2x} p_{1y} p_z - n_{2x} p_{1y} p_{3z} + n_{1x} p_{2y} p_{3z}
\end{aligned}
$$

$$
\begin{aligned}
d \;=\; & -p_x p_{1z} p_{3y} + p_{2x} p_{1y} p_z + p_x p_{1z} p_{2y} - p_{3x} p_{1y} p_z + p_x p_{2z} p_{3y} + p_{3x} p_{1y} p_{2z} + \\
& p_{1x} p_z p_{3y} + p_{1x} p_{2z} p_y - p_{2x} p_{1y} p_{3z} - p_x p_{2y} p_{3z} + p_{3x} p_{2y} p_z + p_{3x} p_{1z} p_y - \\
& p3x p_{1z} p_{2y} + p_x p_{1y} p_{3z} - p_{2x} p_z p_{3y} - p_{2x} p_{1z} p_y + p_{2x} p_{1z} p_{3y} + p_{2x} p_y p_{3z} - \\
& p_{1x} p_{2y} p_z - p_{1x} p_{2z} p_{3y} - p_{1x} p_y p_{3z} + p_{1x} p_{2y} p_{3z} - p_x p_{1y} p_{2z} - p_{3x} p_{2z} p_y
\end{aligned}
$$

A.2 Line Segment Intersection Coplanarity Cubic

The intersection between two scene points, defining a scene line, and two trilinear projection points and normals, defining an edge of the trilinear projection, where the scene points are p_{s1} and p_{s2} and the trilinear projection points are p_i and p_j and the normals are n_i and n_j $(i, j = 1, 2, 3, i \neq j)$, gives a quadratic equation in terms of t. The coefficients of this quadratic are a,b and c where $at^2 + bt + c = 0$, the full expansion of these coefficients is as follows:

$$
\begin{aligned}
a \;=\; & -p_{s1x} n_{jy} n_{iz} + p_{s1x} n_{jz} n_{iy} - n_{jx} p_{s2y} n_{iz} + n_{ix} p_{s2y} n_{jz} + n_{ix} p_{s1z} n_{jy} - n_{ix} p_{s2z} n_{jy} - \\
& n_{ix} p_{s1y} n_{jz} + n_{jx} p_{s2z} n_{iy} + p_{s2x} n_{jy} n_{iz} - n_{jx} p_{s1z} n_{iy} - p_{s2x} n_{jz} n_{iy} + n_{jx} p_{s1y} n_{iz}
\end{aligned}
$$

$$
\begin{aligned}
b &= n_{ix}p_{s2y}p_{jz} - n_{ix}p_{s2z}p_{jy} + p_{s2x}p_{s1y}n_{jz} - p_{s2x}p_{s1z}n_{jy} - p_{s1x}p_{s2y}n_{jz} + p_{s1x}p_{s2z}n_{jy} + \\
&\quad n_{jx}p_{s2y}p_{s1z} - n_{jx}p_{s2z}p_{s1y} + p_{s2x}p_{jy}n_{iz} + p_{s2x}n_{jy}p_{iz} - p_{s2x}p_{jz}n_{iy} - p_{s2x}n_{jz}p_{iy} - \\
&\quad n_{jx}p_{s1z}p_{iy} + p_{jx}p_{s1y}n_{iz} - p_{jx}p_{s1z}n_{iy} - n_{ix}p_{s1y}p_{jz} + n_{ix}p_{s1z}p_{jy} - p_{ix}p_{s1y}n_{jz} + \\
&\quad p_{ix}p_{s1z}n_{jy} + p_{ix}p_{s2y}n_{jz} - p_{ix}p_{s2z}n_{jy} + p_{s2x}p_{s1z}n_{iy} - p_{s2x}p_{s1y}n_{iz} - n_{ix}p_{s2y}p_{s1z} + \\
&\quad n_{ix}p_{s2z}p_{s1y} + p_{s1x}p_{s2y}n_{iz} - p_{s1x}p_{s2z}n_{iy} - p_{s1x}p_{jy}n_{iz} - p_{s1x}n_{jy}p_{iz} + p_{s1x}p_{jz}n_{iy} + \\
&\quad p_{s1x}n_{jz}p_{iy} + n_{jx}p_{s1y}p_{iz} - n_{jx}p_{s2y}p_{iz} + n_{jx}p_{s2z}p_{iy} - p_{jx}p_{s2y}n_{iz} + p_{jx}p_{s2z}n_{iy}
\end{aligned}
$$

$$
\begin{aligned}
c &= p_{jx}p_{s2y}p_{s1z} + p_{s1x}p_{s2z}p_{jy} - p_{s1x}p_{jy}p_{iz} - p_{jx}p_{s2z}p_{s1y} + p_{s2x}p_{s1y}p_{jz} + p_{ix}p_{s2z}p_{s1y} - \\
&\quad p_{s1x}p_{s2y}p_{jz} - p_{s2x}p_{s1z}p_{jy} - p_{ix}p_{s2z}p_{jy} + p_{ix}p_{s2y}p_{jz} - p_{s2x}p_{s1y}p_{iz} - p_{jx}p_{s2y}p_{iz} - \\
&\quad p_{s2x}p_{jz}p_{iy} + p_{s2x}p_{jy}p_{iz} + p_{ix}p_{s1z}p_{jy} - p_{jx}p_{s1z}p_{iy} + p_{jx}p_{s1y}p_{iz} + p_{s1x}p_{jz}p_{iy} - \\
&\quad p_{s1x}p_{s2z}p_{iy} + p_{s1x}p_{s2y}p_{iz} + p_{s2x}p_{s1z}p_{iy} - p_{ix}p_{s1y}p_{jz} - p_{ix}p_{s2y}p_{s1z} + p_{jx}p_{s2z}p_{iy}
\end{aligned}
$$

A.3 Precalculation for Cubic Coefficients

$$
\begin{aligned}
A_4 &= -n_{1x}(n_{2y}n_{3z} - n_{3y}n_{2z}) - n_{2x}(n_{3y}n_{1z} - n_{1y}n_{3z}) - n_{3x}(n_{1y}n_{2z} - n_{2y}n_{1z}) \\
B_1 &= (n_{2x}(n_{3y} - n_{1y}) + n_{1x}(n_{2y} - n_{3y}) + n_{3x}(n_{1y} - n_{2y}) \\
B_2 &= (n_{2y}(n_{3z} - n_{1z}) + n_{1y}(n_{2z} - n_{3z}) + n_{3y}(n_{1z} - n_{2z}) \\
B_3 &= (n_{2z}(n_{3x} - n_{1x}) + n_{1z}(n_{2x} - n_{3x}) + n_{3z}(n_{1x} - n_{2x}) \\
B_4 &= -p_{1x}(n_{2y}n_{3z} - n_{3y}n_{2z}) - n_{1x}(p_{2y}n_{3z} + n_{2y}p_{3z} - p_{3y}n_{2z} - n_{3y}p_{2z}) - \\
&\quad p_{2x}(n_{3y}n_{1z} - n_{1y}n_{3z}) - n_{2x}(p_{3y}n_{1z} + n_{3y}p_{1z} - p_{1y}n_{3z} - n_{1y}p_{3z}) - \\
&\quad p_{3x}(n_{1y}n_{2z} - n_{2y}n_{1z}) - n_{3x}(p_{1y}n_{2z} + n_{1y}p_{2z} - p_{2y}n_{1z} - n_{2y}p_{1z}) \\
C_1 &= p_{1x}(n_{2y} - n_{3y}) + n_{1x}(p_{2y} - p_{3y}) + p_{2x}(n_{3y} - n_{1y}) + n_{2x}(p_{3y} - p_{1y}) + \\
&\quad p_{3x}(n_{1y} - n_{2y}) + n_{3x}(p_{1y} - p_{2y})) \\
C_2 &= p_{1y}(n_{2z} - n_{3z}) + n_{1y}(p_{2z} - p_{3z}) + p_{2y}(n_{3z} - n_{1z}) + n_{2y}(p_{3z} - p_{1z}) + \\
&\quad p_{3y}(n_{1z} - n_{2z}) + n_{3y}(p_{1z} - p_{2z})) \\
C_3 &= p_{1z}(n_{2x} - n_{3x}) + n_{1z}(p_{2x} - p_{3x}) + p_{2z}(n_{3x} - n_{1x}) + n_{2z}(p_{3x} - p_{1x}) + \\
&\quad p_{3z}(n_{1x} - n_{2x}) + n_{3z}(p_{1x} - p_{2x})) \\
C_4 &= -p_{1x}(p_{2y}n_{3z} + n_{2y}p_{3z} - p_{3y}n_{2z} - n_{3y}p_{2z}) - n_{1x}(p_{2y}p_{3z} - p_{2z}p_{3y}) - \\
&\quad p_{2x}(p_{3y}n_{1z} + n_{3y}p_{1z} - p_{1y}n_{3z} - n_{1y}p_{3z}) - n_{2x}(p_{1z}p_{3y} - p_{1y}p_{3z}) - \\
&\quad p_{3x}(p_{1y}n_{2z} + n_{1y}p_{2z} - p_{2y}n_{1z} - n_{2y}p_{1z}) - n_{3x}(p_{1y}p_{2z} - p_{1z}p_{2y}) \\
D_1 &= p_{1y}(p_{2z} - p_{3z}) + p_{3y}(p_{1z} - p_{2z}) + p_{2y}(p_{3z} - p_{1z}) \\
D_2 &= p_{1x}(p_{2y} - p_{3y}) + p_{3x}(p_{1y} - p_{2y}) + p_{2x}(p_{3y} - p_{1y}) \\
D_3 &= p_{1z}(p_{2x} - p_{3x}) + p_{3z}(p_{1x} - p_{2x}) + p_{2z}(p_{3x} - p_{1x}) \\
D_4 &= p_{1x}(p_{2y}p_{3z} - p_{2z}p_{3y})) + (p_{3x}(p_{1y}p_{2z} - p_{1z}p_{2y}) + p_{2x}(p_{1z}p_{3y} - p_{1y}p_{3z})
\end{aligned}
$$

Appendix B

Vector Properties

Various vector properties of the trilinear projections points and normals can be stored for the efficient calculation of trilinear projection edge intersections. These properties are either the cross product or subtraction of two points or normals, and a full listing is show in Table B.1.

Variable Name	Description
`cp1p2`	$p_1 \times p_2$
`cp2p3`	$p_2 \times p_3$
`cp3p1`	$p_3 \times p_1$
`cn1n2`	$n_1 \times n_2$
`cn2n3`	$n_2 \times n_3$
`cn3n1`	$n_3 \times n_1$
`cp1n2`	$p_1 \times n_2$
`cp2n3`	$p_2 \times n_3$
`cp3n1`	$p_3 \times n_1$
`cn1p2`	$n_1 \times p_2$
`cn2p3`	$n_2 \times p_3$
`cn3p1`	$n_3 \times p_1$
`ep1p2`	$p_1 - p_2$
`ep2p3`	$p_2 - p_3$
`ep3p1`	$p_3 - p_1$
`en1n2`	$n_1 - n_2$
`en2n3`	$n_2 - n_3$
`en3n1`	$n_3 - n_1$

Table B.1: Vector properties of the trilinear propjection

Appendix C

View and Scene Data

This appendix shows the data used to generate the examples 4.1 to 4.7. Table C.1 shows vertex coordinates for the examples. Each scene is composed of one scene triangle with three vertices whose coordinates are (v.x, v.y, v.z). Table C.2 shows the view triangle parameters for the examples, each view triangle vertex has a position (v.x, v.y, v.z) and a vector ¡n.x, n.y, n.z¿. All values in these tables were randomly generated.

Scene	v.x	v.y	v.z
2,2,2,3			
	1.58e-001	-3.19e-001	5.22e-002
	-2.17e-001	-4.18e-001	-4.91e-002
	2.50e-002	-2.30e-001	-2.02e-001
3,3,3			
	9.62e-002	-4.57e-002	-1.10e-002
	5.35e-002	-1.44e-001	-3.84e-002
	7.26e-002	-1.48e-001	-1.07e-001
4,2,3			
	1.71e-001	9.28e-002	2.21e-001
	1.52e-002	9.31e-002	2.44e-001
	-2.13e-001	-7.65e-002	4.45e-001
4,5			
	-2.36e-001	-1.43e-001	1.30e-001
	-2.42e-001	7.43e-001	1.84e-001
	-3.35e-001	-1.31e-001	1.66e-001
6,3			
	3.13e-001	2.99e-001	5.95e-002
	5.15-001	6.30e-001	3.60e-001
	3.91e-001	3.82e-001	2.10e-001
2,7			
	-6.70e-001	-8.24e-001	3.47e-001
	-7.29-001	-7.87e-001	3.78e-001
	-1.17e-001	-3.88e-001	2.65e-001
9			
	7.04e-002	4.79e-001	1.62e-001
	-1.70e-001	-3.64e-001	-3.52e-001
	9.62e-002	-4.57e-002	-1.10e-002

Table C.1:

View	v.x	v.y	v.z	n.x	n.y	n.z
2,2,2,3						
	0.51	-0.96	0.74	0.81	0.26	-0.52
	-0.73	0.82	0.06	0.06	-0.83	-0.56
	-0.44	-0.70	-0.53	-0.11	0.70	0.70
3,3,3						
	0.82	0.46	-0.80	-0.43	-0.04	0.90
	-0.25	0.14	0.67	0.36	-0.62	-0.70
	0.12	-0.20	-0.09	-0.47	-0.61	-0.64
4,2,3						
	-0.85	-0.64	0.77	-0.14	-0.02	0.99
	0.55	0.60	-0.83	-0.87	0.38	0.31
	0.44	0.72	-0.08	0.70	-0.70	-0.16
4,5						
	0.97	-0.37	-0.39	-0.85	-0.42	0.31
	-0.05	0.45	0.76	-0.20	0.80	-0.57
	-0.83	-0.71	-0.38	0.55	-0.19	0.82
6,3						
	-0.07	-0.91	0.77	-0.32	-0.67	-0.67
	-0.13	0.26	0.49	0.73	0.68	0.08
	0.28	-0.25	-0.98	0.08	0.68	0.73
2,7						
	0.68	-0.00	-0.39	0.26	0.71	0.65
	-0.43	-0.85	0.77	-0.77	-0.56	-0.32
	-0.61	0.95	-0.39	0.59	-0.74	0.33
9						
	-0.45	0.44	-0.43	0.34	0.54	-0.77
	0.58	-0.75	-0.58	-0.74	-0.52	-0.43
	0.00	0.81	-0.26	0.37	0.52	0.77

Table C.2:

Appendix D

Tabulated Performance Results

The following sections present the full tabulated data points for the graphs in Chapter 8 that pertain to results on random scene data.

D.1 Ray Tracing Results

Resolution (square pixels)	Scene Triangles	Trilinear Projections	Execution Time (ms)
1	1	2	0.89
1	1	8	1.04
1	1	18	1.08
1	1	32	1.14
1	1	50	1.23
16384	1	2	17.43
16384	1	8	16.57
16384	1	18	17.88
16384	1	32	17.14
16384	1	50	18.54
65536	1	2	65.69
65536	1	8	61.82
65536	1	18	65.06
65536	1	32	63.19
65536	1	50	66.73
147456	1	2	147.29
147456	1	8	137.63
147456	1	18	143.00
147456	1	32	138.88
147456	1	50	143.00
262144	1	2	260.50
262144	1	8	244.40
262144	1	18	253.00
262144	1	32	244.40
262144	1	50	250.25
409600	1	2	410.67
409600	1	8	380.67
409600	1	18	390.67
409600	1	32	380.67
409600	1	50	384.00
1	128	2	1.08
1	128	8	1.36

1	128	18	1.65
1	128	32	1.96
1	128	50	2.46
16384	128	2	390.67
16384	128	8	400.67
16384	128	18	420.67
16384	128	32	364.00
16384	128	50	427.33
65536	128	2	1572.00
65536	128	8	1622.00
65536	128	18	1652.00
65536	128	32	1482.00
65536	128	50	1672.00
147456	128	2	3545.00
147456	128	8	3605.00
147456	128	18	3635.00
147456	128	32	3284.00
147456	128	50	3635.00
262144	128	2	6259.00
262144	128	8	6369.00
262144	128	18	6429.00
262144	128	32	5768.00
262144	128	50	6409.00
409600	128	2	9714.00
409600	128	8	9955.00
409600	128	18	10015.00
409600	128	32	9083.00
409600	128	50	9995.00
1	256	2	1.33
1	256	8	1.66
1	256	18	2.21
1	256	32	2.78
1	256	50	3.72
16384	256	2	751.00
16384	256	8	816.00
16384	256	18	836.00
16384	256	32	726.00
16384	256	50	846.00
65536	256	2	2944.00
65536	256	8	3205.00
65536	256	18	3255.00
65536	256	32	2834.00
65536	256	50	3235.00
147456	256	2	6639.00
147456	256	8	7241.00
147456	256	18	7211.00
147456	256	32	6439.00
147456	256	50	7110.00
262144	256	2	11787.00
262144	256	8	12789.00
262144	256	18	12729.00
262144	256	32	11316.00
262144	256	50	12568.00
409600	256	2	18587.00
409600	256	8	20029.00
409600	256	18	19898.00
409600	256	32	17786.00
409600	256	50	19398.00
1	384	2	1.60
1	384	8	2.01
1	384	18	2.84
1	384	32	3.74

1	384	50	5.16
16384	384	2	1142.00
16384	384	8	1301.00
16384	384	18	1321.00
16384	384	32	1132.00
16384	384	50	1332.00
65536	384	2	4547.00
65536	384	8	5077.00
65536	384	18	5097.00
65536	384	32	4437.00
65536	384	50	5067.00
147456	384	2	10155.00
147456	384	8	11477.00
147456	384	18	11367.00
147456	384	32	10065.00
147456	384	50	11166.00
262144	384	2	17905.00
262144	384	8	20299.00
262144	384	18	20069.00
262144	384	32	17695.00
262144	384	50	19769.00
409600	384	2	28031.00
409600	384	8	31785.00
409600	384	18	31365.00
409600	384	32	27820.00
409600	384	50	30514.00
1	512	2	1.94
1	512	8	2.50
1	512	18	3.52
1	512	32	4.74
1	512	50	6.81
16384	512	2	1592.00
16384	512	8	1833.00
16384	512	18	1843.00
16384	512	32	1602.00
16384	512	50	1853.00
65536	512	2	6269.00
65536	512	8	7221.00
65536	512	18	7221.00
65536	512	32	6329.00
65536	512	50	7180.00
147456	512	2	14271.00
147456	512	8	16314.00
147456	512	18	16103.00
147456	512	32	14450.00
147456	512	50	15802.00
262144	512	2	25136.00
262144	512	8	28852.00
262144	512	18	28461.00
262144	512	32	25277.00
262144	512	50	27970.00
409600	512	2	39467.00
409600	512	8	45185.00
409600	512	18	44464.00
409600	512	32	39717.00
409600	512	50	43172.00
1	640	2	2.25
1	640	8	3.07
1	640	18	4.70
1	640	32	6.30
1	640	50	9.36
16384	640	2	2324.00

16384	640	8	2864.00
16384	640	18	2874.00
16384	640	32	2243.00
16384	640	50	2754.00
65536	640	2	9263.00
65536	640	8	11407.00
65536	640	18	11266.00
65536	640	32	8923.00
65536	640	50	10666.00
147456	640	2	20840.00
147456	640	8	25757.00
147456	640	18	25096.00
147456	640	32	20239.00
147456	640	50	23473.00
262144	640	2	37103.00
262144	640	8	45546.00
262144	640	18	44314.00
262144	640	32	35601.00
262144	640	50	41540.00
409600	640	2	57913.00
409600	640	8	71352.00
409600	640	18	69250.00
409600	640	32	55970.00
409600	640	50	64122.00

Table D.1: Ray tracing results over random scene data

D.2 Trilinear Projection Results

Resolution (square pixels)	Scene Triangles	Trilinear Projections	Execution Time (ms)
1	1	2	0.06
1	1	8	0.21
1	1	18	0.66
1	1	32	0.99
1	1	50	1.95
16384	1	2	0.11
16384	1	8	0.20
16384	1	18	0.66
16384	1	32	1.08
16384	1	50	1.86
65536	1	2	0.11
65536	1	8	0.21
65536	1	18	0.67
65536	1	32	1.09
65536	1	50	1.86
147456	1	2	0.11
147456	1	8	0.21
147456	1	18	0.67
147456	1	32	1.09
147456	1	50	1.90
262144	1	2	0.11
262144	1	8	0.20
262144	1	18	0.66
262144	1	32	1.09
262144	1	50	1.90
409600	1	2	0.11
409600	1	8	0.20
409600	1	18	0.66
409600	1	32	1.08
409600	1	50	1.89
1	128	2	6.22
1	128	8	16.15
1	128	18	47.67
1	128	32	82.38
1	128	50	137.63
16384	128	2	7.00
16384	128	8	16.15
16384	128	18	48.62
16384	128	32	83.42
16384	128	50	136.38
65536	128	2	6.95
65536	128	8	16.15
65536	128	18	49.10
65536	128	32	83.42
65536	128	50	137.63
147456	128	2	6.95
147456	128	8	16.15
147456	128	18	49.10
147456	128	32	83.42
147456	128	50	137.63
262144	128	2	8.14
262144	128	8	16.31
262144	128	18	48.62
262144	128	32	83.42
262144	128	50	136.38
409600	128	2	11.36
409600	128	8	16.15
409600	128	18	48.62
409600	128	32	83.42

409600	128	50	137.75
1	256	2	12.83
1	256	8	32.61
1	256	18	94.64
1	256	32	164.57
1	256	50	275.50
16384	256	2	13.85
16384	256	8	32.61
16384	256	18	96.45
16384	256	32	166.83
16384	256	50	273.00
65536	256	2	13.85
65536	256	8	32.61
65536	256	18	97.36
65536	256	32	166.83
65536	256	50	273.00
147456	256	2	13.71
147456	256	8	32.61
147456	256	18	98.27
147456	256	32	166.83
147456	256	50	273.00
262144	256	2	17.74
262144	256	8	32.61
262144	256	18	97.36
262144	256	32	166.83
262144	256	50	273.00
409600	256	2	24.41
409600	256	8	32.29
409600	256	18	97.36
409600	256	32	166.83
409600	256	50	273.00
1	384	2	18.89
1	384	8	49.57
1	384	18	141.50
1	384	32	246.40
1	384	50	414.00
16384	384	2	20.43
16384	384	8	48.62
16384	384	18	145.86
16384	384	32	250.50
16384	384	50	407.33
65536	384	2	20.43
65536	384	8	48.62
65536	384	18	145.86
65536	384	32	250.50
65536	384	50	407.33
147456	384	2	20.43
147456	384	8	48.62
147456	384	18	145.86
147456	384	32	250.50
147456	384	50	407.33
262144	384	2	25.27
262144	384	8	49.10
262144	384	18	144.43
262144	384	32	250.50
262144	384	50	410.67
409600	384	2	35.90
409600	384	8	49.10
409600	384	18	144.43
409600	384	32	248.40
409600	384	50	410.67
1	512	2	25.52

1	512	8	65.06
1	512	18	188.67
1	512	32	330.50
1	512	50	555.50
16384	512	2	27.32
16384	512	8	65.06
16384	512	18	193.67
16384	512	32	334.00
16384	512	50	545.50
65536	512	2	27.32
65536	512	8	65.06
65536	512	18	193.67
65536	512	32	334.00
65536	512	50	545.50
147456	512	2	27.32
147456	512	8	65.06
147456	512	18	193.67
147456	512	32	334.00
147456	512	50	545.50
262144	512	2	35.75
262144	512	8	65.06
262144	512	18	193.67
262144	512	32	333.00
262144	512	50	550.50
409600	512	2	50.55
409600	512	8	65.06
409600	512	18	193.67
409600	512	32	333.00
409600	512	50	550.50
1	640	2	31.28
1	640	8	81.62
1	640	18	234.40
1	640	32	410.67
1	640	50	701.00
16384	640	2	33.70
16384	640	8	81.62
16384	640	18	240.40
16384	640	32	414.00
16384	640	50	691.00
65536	640	2	33.70
65536	640	8	81.62
65536	640	18	242.40
65536	640	32	414.00
65536	640	50	696.00
147456	640	2	33.70
147456	640	8	81.62
147456	640	18	242.40
147456	640	32	414.00
147456	640	50	691.00
262144	640	2	43.96
262144	640	8	81.62
262144	640	18	240.40
262144	640	32	417.33
262144	640	50	696.00
409600	640	2	62.56
409600	640	8	81.62
409600	640	18	242.40
409600	640	32	414.00
409600	640	50	696.00

Table D.2: Trilinear projection results over random scene data

D.3 Trilinear Projection with Scene Space Clipping Results

Resolution (square pixels)	Scene Triangles	Trilinear Projections	Execution Time (ms)
1	1	2	0.16
1	1	8	0.26
1	1	18	1.20
1	1	32	1.92
1	1	50	3.20
16384	1	2	0.15
16384	1	8	0.26
16384	1	18	1.19
16384	1	32	1.93
16384	1	50	3.14
65536	1	2	0.15
65536	1	8	0.26
65536	1	18	1.15
65536	1	32	1.93
65536	1	50	3.14
147456	1	2	0.15
147456	1	8	0.26
147456	1	18	1.15
147456	1	32	1.95
147456	1	50	3.20
262144	1	2	0.18
262144	1	8	0.26
262144	1	18	1.14
262144	1	32	1.93
262144	1	50	3.20
409600	1	2	0.26
409600	1	8	0.26
409600	1	18	1.15
409600	1	32	1.93
409600	1	50	3.22
1	128	2	17.43
1	128	8	25.92
1	128	18	98.27
1	128	32	185.33
1	128	50	315.50
16384	128	2	17.74
16384	128	8	25.27
16384	128	18	99.18
16384	128	32	182.00
16384	128	50	313.00
65536	128	2	17.74
65536	128	8	25.27
65536	128	18	100.10
65536	128	32	180.33
65536	128	50	310.50
147456	128	2	17.74
147456	128	8	25.27
147456	128	18	100.10
147456	128	32	182.00
147456	128	50	310.50
262144	128	2	17.56
262144	128	8	25.27
262144	128	18	100.10
262144	128	32	180.33
262144	128	50	313.00
409600	128	2	18.05
409600	128	8	25.27
409600	128	18	100.10
409600	128	32	180.33

409600	128	50	313.00
1	256	2	34.03
1	256	8	52.68
1	256	18	197.00
1	256	32	364.00
1	256	50	645.50
16384	256	2	34.52
16384	256	8	51.55
16384	256	18	198.67
16384	256	32	360.67
16384	256	50	630.50
65536	256	2	34.52
65536	256	8	51.05
65536	256	18	198.67
65536	256	32	357.33
65536	256	50	635.50
147456	256	2	34.37
147456	256	8	51.05
147456	256	18	198.67
147456	256	32	360.67
147456	256	50	640.50
262144	256	2	34.52
262144	256	8	51.55
262144	256	18	198.67
262144	256	32	357.33
262144	256	50	640.50
409600	256	2	35.55
409600	256	8	51.55
409600	256	18	200.20
409600	256	32	357.33
409600	256	50	635.50
1	384	2	51.05
1	384	8	77.77
1	384	18	295.50
1	384	32	551.00
1	384	50	956.50
16384	384	2	50.55
16384	384	8	77.00
16384	384	18	295.50
16384	384	32	546.00
16384	384	50	946.50
65536	384	2	50.05
65536	384	8	76.50
65536	384	18	295.50
65536	384	32	541.00
65536	384	50	946.50
147456	384	2	50.05
147456	384	8	77.00
147456	384	18	295.50
147456	384	32	540.50
147456	384	50	946.50
262144	384	2	50.55
262144	384	8	77.00
262144	384	18	295.50
262144	384	32	540.50
262144	384	50	941.50
409600	384	2	52.55
409600	384	8	77.00
409600	384	18	295.50
409600	384	32	540.50
409600	384	50	946.50
1	512	2	68.73

1	512	8	103.10
1	512	18	397.33
1	512	32	746.00
1	512	50	1281.00
16384	512	2	67.40
16384	512	8	102.10
16384	512	18	397.33
16384	512	32	736.00
16384	512	50	1281.00
65536	512	2	67.40
65536	512	8	102.10
65536	512	18	397.33
65536	512	32	731.00
65536	512	50	1271.00
147456	512	2	67.40
147456	512	8	102.10
147456	512	18	397.33
147456	512	32	731.00
147456	512	50	1271.00
262144	512	2	68.07
262144	512	8	101.10
262144	512	18	397.33
262144	512	32	736.00
262144	512	50	1271.00
409600	512	2	74.36
409600	512	8	101.10
409600	512	18	397.33
409600	512	32	731.00
409600	512	50	1271.00
1	640	2	84.25
1	640	8	128.88
1	640	18	497.67
1	640	32	926.50
1	640	50	1572.00
16384	640	2	83.15
16384	640	8	127.63
16384	640	18	497.67
16384	640	32	916.50
16384	640	50	1572.00
65536	640	2	82.38
65536	640	8	126.38
65536	640	18	501.00
65536	640	32	911.50
65536	640	50	1572.00
147456	640	2	83.42
147456	640	8	126.38
147456	640	18	501.00
147456	640	32	906.50
147456	640	50	1572.00
262144	640	2	83.15
262144	640	8	127.63
262144	640	18	497.67
262144	640	32	906.50
262144	640	50	1572.00
409600	640	2	90.92
409600	640	8	127.63
409600	640	18	501.00
409600	640	32	906.50
409600	640	50	1572.00

Table D.3: Trilinear projection with clipping results over random scene data

D.4 Ray Tracing on Different Configurations

Resolution (square pixels)	Execution Time (ms)
16384	16.41
65536	61.82
147456	140.25
262144	246.40
409600	387.33

Table D.4: Ray tracing results on a 2,2,2,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	16.97
65536	63.19
147456	144.57
262144	255.50
409600	400.67

Table D.5: Ray tracing results on a 2,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	16.97
65536	64.44
147456	144.57
262144	255.50
409600	400.67

Table D.6: Ray tracing results on a 3,3,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	16.85
65536	63.19
147456	144.57
262144	253.00
409600	397.33

Table D.7: Ray tracing results on a 4,2,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	16.85
65536	63.19
147456	144.57
262144	255.50
409600	397.33

Table D.8: Ray tracing results on a 4,5 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	16.97
65536	64.44
147456	144.57
262144	255.50
409600	397.33

Table D.9: Ray tracing results on a 6,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	17.26
65536	65.69
147456	148.86
262144	260.50
409600	410.67

Table D.10: Ray tracing results on a 9 configuration example

D.5 Trilinear Projection on Different Configurations

Resolution (square pixels)	Execution Time (ms)
16384	0.02
65536	0.02
147456	0.02
262144	0.02
409600	0.02

Table D.11: Trilinear projection results on a 2,2,2,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	0.07
65536	0.07
147456	0.07
262144	0.07
409600	0.07

Table D.12: Trilinear projection results on a 2,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	0.02
65536	0.02
147456	0.02
262144	0.02
409600	0.02

Table D.13: Trilinear projection results on a 3,3,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	0.02
65536	0.02
147456	0.02
262144	0.02
409600	0.02

Table D.14: Trilinear projection results on a 4,2,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	0.03
65536	0.03
147456	0.03
262144	0.03
409600	0.04

Table D.15: Trilinear projection results on a 4,5 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	0.02
65536	0.02
147456	0.02
262144	0.02
409600	0.03

Table D.16: Trilinear projection results on a 6,3 configuration example

Resolution (square pixels)	Execution Time (ms)
16384	0.09
65536	0.09
147456	0.16
262144	0.21
409600	0.25

Table D.17: Trilinear projection results on a 9 configuration example

D.6 Trilinear Projection with Scene Triangle Tessellation Results on Different Configurations

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.05	1.00424
409600	2	0.09	1.00228
409600	3	0.15	0.47538
409600	4	0.21	0.47147
409600	5	0.29	0.54483
409600	6	0.46	0.60483
409600	7	0.60	0.42941
409600	8	0.78	0.44050
409600	9	0.97	0.47669
409600	10	1.17	0.51157
409600	20	4.57	0.29116
409600	30	9.63	0.20215
409600	40	18.20	0.19987
409600	50	27.81	0.18422
409600	60	40.84	0.17672
409600	70	60.65	0.17770
409600	80	79.31	0.18389
409600	90	102.10	0.19694
409600	100	122.44	0.17998

Table D.18: Trilinear projection scene triangle tessellation results on a 2,2,2,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.11	1.39485
409600	2	0.11	1.07603
409600	3	0.19	0.79223
409600	4	0.29	0.28525
409600	5	0.38	0.56777
409600	6	0.60	0.21992
409600	7	0.77	0.21212
409600	8	0.99	0.18218
409600	9	1.29	0.16966
409600	10	1.52	0.15787
409600	20	5.79	0.12484
409600	30	12.67	0.12012
409600	40	21.76	0.11776
409600	50	34.37	0.11595
409600	60	46.41	0.11631
409600	70	64.50	0.11631
409600	80	87.58	0.11649
409600	90	121.33	0.11522
409600	100	147.43	0.11341

Table D.19: Trilinear projection scene triangle tessellation results on a 2,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.07	0.20702
409600	2	0.13	0.19347
409600	3	0.22	0.18854
409600	4	0.34	0.18854
409600	5	0.48	0.18854
409600	6	0.72	0.18854

409600	7	0.93	0.18608
409600	8	1.22	0.18731
409600	9	1.55	0.18854
409600	10	1.89	0.18608
409600	20	7.20	0.18792
409600	30	15.64	0.18854
409600	40	27.05	0.18608
409600	50	42.13	0.18608
409600	60	60.06	0.18731
409600	70	80.08	0.18854
409600	80	104.20	0.18854
409600	90	135.25	0.18792
409600	100	175.33	0.18854

Table D.20: Trilinear projection scene triangle tessellation results on a 3,3,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.06	0.55771
409600	2	0.11	0.53959
409600	3	0.17	0.53462
409600	4	0.25	0.45222
409600	5	0.33	0.45470
409600	6	0.52	0.46538
409600	7	0.68	0.24423
409600	8	0.89	0.25292
409600	9	1.13	0.21718
409600	10	1.39	0.16977
409600	20	5.13	0.15463
409600	30	11.00	0.11889
409600	40	19.82	0.10623
409600	50	30.33	0.08761
409600	60	41.71	0.09010
409600	70	63.19	0.08563
409600	80	83.23	0.08439
409600	90	110.20	0.08464
409600	100	132.75	0.08017

Table D.21: Trilinear projection scene triangle tessellation results on a 4,2,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.07	0.59154
409600	2	0.11	0.36698
409600	3	0.17	0.26357
409600	4	0.24	0.20495
409600	5	0.33	0.17172
409600	6	0.51	0.15108
409600	7	0.66	0.13581
409600	8	0.83	0.12714
409600	9	1.03	0.12054
409600	10	1.26	0.11455
409600	20	4.70	0.09659
409600	30	9.91	0.08256
409600	40	17.26	0.07926
409600	50	26.61	0.07657
409600	60	40.04	0.07554
409600	70	58.39	0.07534
409600	80	77.77	0.07492
409600	90	101.10	0.07410

409600	100	122.44	0.07410

Table D.22: Trilinear projection scene triangle tessellation results on a 4,5 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.07	0.76289
409600	2	0.19	0.32960
409600	3	0.27	0.26822
409600	4	0.34	0.28924
409600	5	0.47	0.23459
409600	6	0.70	0.22141
409600	7	0.98	0.19731
409600	8	1.19	0.20544
409600	9	1.47	0.19647
409600	10	1.82	0.18330
409600	20	6.86	0.17741
409600	30	14.72	0.17209
409600	40	25.27	0.17237
409600	50	40.44	0.17096
409600	60	55.32	0.17265
409600	70	75.79	0.17152
409600	80	96.55	0.17180
409600	90	131.50	0.17068
409600	100	168.67	0.17096

Table D.23: Trilinear projection scene triangle tessellation results on a 6,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.34	0.38838
409600	2	0.39	0.20656
409600	3	0.43	0.13136
409600	4	0.45	0.11492
409600	5	0.60	0.09637
409600	6	0.79	0.08704
409600	7	0.97	0.08055
409600	8	1.23	0.03644
409600	9	1.49	0.03745
409600	10	1.85	0.03475
409600	20	6.67	0.02439
409600	30	15.32	0.01795
409600	40	27.32	0.01718
409600	50	44.83	0.01584
409600	60	63.81	0.01489
409600	70	83.42	0.01431
409600	80	103.20	0.01449
409600	90	130.25	0.01444
409600	100	158.86	0.01411

Table D.24: Trilinear projection scene triangle tessellation results on a 9 configuration example

D.7 Trilinear Projection with Parametric Triangle Slicing Results on Different Configurations

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.07	0.58885
409600	2	0.11	0.37333
409600	3	0.15	0.29312
409600	4	0.20	0.24747
409600	5	0.24	0.21976
409600	6	0.28	0.20085
409600	7	0.33	0.19107
409600	8	0.37	0.18389
409600	9	0.42	0.17672
409600	10	0.46	0.17020
409600	20	0.89	0.17216
409600	30	1.31	0.17998
409600	40	1.74	0.18194
409600	50	2.17	0.18128
409600	60	2.60	0.18291
409600	70	3.02	0.18226
409600	80	3.44	0.18259
409600	90	3.89	0.18324
409600	100	4.30	0.18455

Table D.25: Trilinear projection with parametric triangle slicing results on a 2,2,2,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.13	0.36672
409600	2	0.16	0.39993
409600	3	0.18	0.19488
409600	4	0.20	0.18527
409600	5	0.26	0.18436
409600	6	0.32	0.12974
409600	7	0.34	0.14172
409600	8	0.39	0.13845
409600	9	0.41	0.12067
409600	10	0.47	0.12920
409600	20	0.85	0.11740
409600	30	1.27	0.11504
409600	40	1.69	0.11450
409600	50	2.10	0.11323
409600	60	2.52	0.11450
409600	70	2.94	0.11287
409600	80	3.30	0.11359
409600	90	3.74	0.11305
409600	100	4.15	0.11359

Table D.26: Trilinear projection with parametric triangle slicing results on a 2,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.06	0.21134
409600	2	0.10	0.20333
409600	3	0.13	0.19409
409600	4	0.16	0.19409
409600	5	0.20	0.19532
409600	6	0.23	0.19285

409600	7	0.26	0.19100
409600	8	0.30	0.19347
409600	9	0.33	0.19285
409600	10	0.36	0.19162
409600	20	0.69	0.18608
409600	30	1.02	0.18608
409600	40	1.34	0.18669
409600	50	1.67	0.18731
409600	60	1.99	0.18731
409600	70	2.31	0.18731
409600	80	2.65	0.18484
409600	90	2.97	0.18546
409600	100	3.27	0.18731

Table D.27: Trilinear projection with parametric triangle slicing results on a 3,3,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.06	0.42988
409600	2	0.09	0.29586
409600	3	0.13	0.20179
409600	4	0.16	0.21023
409600	5	0.19	0.15413
409600	6	0.23	0.14321
409600	7	0.26	0.14172
409600	8	0.29	0.12187
409600	9	0.33	0.12261
409600	10	0.35	0.11591
409600	20	0.68	0.09605
409600	30	1.00	0.09109
409600	40	1.33	0.08861
409600	50	1.66	0.08737
409600	60	1.98	0.08563
409600	70	2.29	0.08563
409600	80	2.61	0.08662
409600	90	3.01	0.08513
409600	100	3.30	0.08563

Table D.28: Trilinear projection with parametric triangle slicing results on a 4,2,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.06	1.09412
409600	2	0.08	0.43426
409600	3	0.10	0.43220
409600	4	0.12	0.40578
409600	5	0.15	0.20722
409600	6	0.17	0.24871
409600	7	0.19	0.25077
409600	8	0.21	0.20372
409600	9	0.23	0.19051
409600	10	0.25	0.14551
409600	20	0.48	0.12549
409600	30	0.69	0.09928
409600	40	0.92	0.09040
409600	50	1.13	0.08937
409600	60	1.35	0.08731
409600	70	1.56	0.08669
409600	80	1.79	0.08400
409600	90	2.01	0.08318

409600	100	2.22	0.08421

Table D.29: Trilinear projection with parametric triangle slicing results on a 4,5 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.05	0.85706
409600	2	0.08	0.76962
409600	3	0.14	0.81362
409600	4	0.17	0.26654
409600	5	0.14	0.29540
409600	6	0.20	0.35622
409600	7	0.22	0.43105
409600	8	0.24	0.23487
409600	9	0.30	0.19254
409600	10	0.32	0.24019
409600	20	0.56	0.18021
409600	30	0.84	0.15807
409600	40	1.07	0.16200
409600	50	1.31	0.16312
409600	60	1.57	0.16003
409600	70	1.82	0.16368
409600	80	2.06	0.16704
409600	90	2.31	0.16312
409600	100	2.61	0.16592

Table D.30: Trilinear projection with parametric triangle slicing results on a 6,3 configuration example

Resolution (square pixels)	Tessellation Factor	Execution Time (ms)	Relative Error
409600	1	0.30	0.75621
409600	2	0.34	0.67250
409600	3	0.33	0.40378
409600	4	0.32	0.18357
409600	5	0.33	0.15479
409600	6	0.35	0.15025
409600	7	0.36	0.10211
409600	8	0.37	0.07449
409600	9	0.38	0.07324
409600	10	0.39	0.07906
409600	20	0.46	0.02755
409600	30	0.55	0.01907
409600	40	0.72	0.01712
409600	50	0.87	0.01548
409600	60	1.04	0.01515
409600	70	1.22	0.01488
409600	80	1.35	0.01502
409600	90	1.53	0.01499
409600	100	1.67	0.01522

Table D.31: Trilinear projection with parametric triangle slicing results on a 9 configuration example

Appendix E

Context in Planar 3D Navigation

Context in 3D Planar Navigation

Scott Vallance
School of Informatics & Engineering
The Flinders University of South Australia
(vallance@infoeng.flinders.edu.au)

Paul Calder
School of Informatics & Engineering
The Flinders University of South Australia
(calder@infoeng.flinders.edu.au)

Keywords: 3D graphics, navigation techniques, distortion viewing.

Abstract

One of the most frustrating barriers to the widespread use of 3D visualisation is the additional complexity in navigating 3D data. This paper details a new approach to improving navigation in 3D environments where the navigation is mainly planar. Data at a distance from the viewpoint is distorted as if projected onto a partial cylinder to approximate a plan view, thereby exposing information that may have been obscured. Previous approaches are compared with this new technique and screenshots presented. Implementation details of the technique are discussed as well as possible performance and useability issues.

1. Introduction

Navigating a virtual 3D space armed with nothing more than a 2D mouse, a keyboard and 2D screen is a difficult task. Systems which simply allow a user 6-degrees-of-freedom movement do so at the risk of making a difficult to use, unintuitive interface. Metaphors are a powerful way of simplifying interactive tasks and making it intuitive. In 3D navigation the most obvious metaphor is that of a virtual person.

In a virtual world, constraining the view to that of a human raises a few undesirable features. Firstly, down at ground level it can be very difficult to see where to go. Secondly, it can be hard to make sense of the lie of the land as objects close to the viewpoint can obscure important detail. There are three different types of views traditionally used with a virtual person: first, second and third person. A first person view looks out from the eyes of the virtual person. A second person view follows or tracks the virtual person, but from outside. A third person view is independent of the virtual person. First person views are the most direct.

Successful navigation relies on a clear understanding of the current context. The context is the surrounding information that allows a greater understanding of the environment. In a word processor the amount of information that can be seen either side of the line currently being typed is the context. In a 3D navigation problem, the context is the amount of data seen around the current location or focus. Trivially, some types of context can be increased by a larger field of view, but this context comes at the expense of the detail and resolution of the objects surrounding the viewpoint. Attempts have been made in 2D and 3D visualisation to increase this context without losing detail, examples of this work can be seen in [WINCH2000], [KEAHEY1998].

The goal of the work reported in this paper is to increase the context for navigation without unduly decreasing the directness of the view. Section 2 details a task that exposes shortcomings of previous approaches and our solution to it. Section 3 presents a comparison of the new technique with previous ones. Section 4 explains the

implementation of the technique, followed by the conclusion in Section 5. Future work arising from the paper is briefly outlined in Section 6.

2. Navigation tasks

This paper investigates techniques for improving navigation in 3D scenarios where the navigation is substantially planar. In particular we have investigated maze navigation scenarios where the task involves both navigation and interaction with the local environment.

Tasks involving just navigation might best be solved by providing a third person plan view. Tasks involving just interaction with the local environment might best be solved by providing a first person view. Tasks involving both navigation and interaction with the local environment require more complex views.

2.1 Techniques

Common approaches to tasks such as the one detailed above can involve single views or multiple views. Single views consist of a first, second or third person view. Multiple views traditionally involve a main view and secondary view to provide context. Examples of each of these techniques are:

- Single view: a first person view, such as in a flight simulator.
- Multiple views: a first person view, with a map (plan view) on top.

Another approach to the problem is World-in-Miniature [STOAKLEY1995]. Here a small third person view shows all of the data in a miniaturised form, and an representation of the user can be moved through the data. A plan view on a first person view is a particular type of World-in-Miniature visualisation.

While a combination of a first and third person view (for example a superimposed plan view) satisfies the navigation task by providing both context and first person directness, it contains an undesirable separation between the information. To navigate, the user need only focus upon the plan view. When interacting directly with the local environment the user must switch to focusing on the first person view. Although this takes only a small amount of time, it reduces the consistency of the interaction, and as a result it may negatively impact on the user's spatial understanding.

2.2 Removing the separation between viewpoints

To remove this separation, we rely on the fact that the information provided by each viewpoint is not needed equally for data at different distances to the user. Put simply, close data needs to be viewed in first person and far data needs to be viewed in either second or third person. To implement this, data is distorted according to its distance from the user, so that close data is the same as a normal first person view, while distorted data approximates looking at that data from a second person viewpoint above the maze. This can be seen in the following figure, which shows a distorted maze:

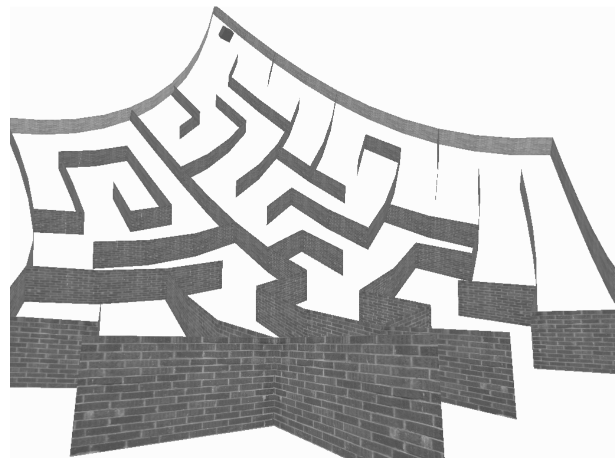


Figure 1: A maze distorted to show context

In this approach both context and directness of interaction are provided. The disadvantage of this approach is the loss of context immediately around the viewpoint. Detail cannot be seen over the walls very close to the viewpoint in the above figure.

3. Comparison

In this section screenshots of various techniques that might be used to combine navigation and local interaction are presented and discussed. The amount of context each technique provides can clearly be seen. Examples of single view solutions to the task will be presented first, followed by multiple views and finally distorted views.

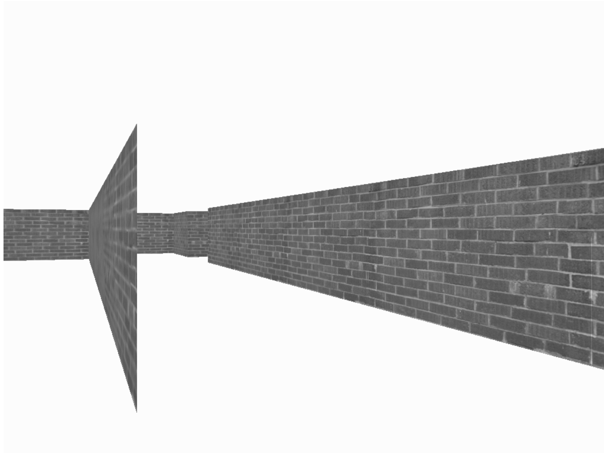


Figure 2: A first person view of the maze

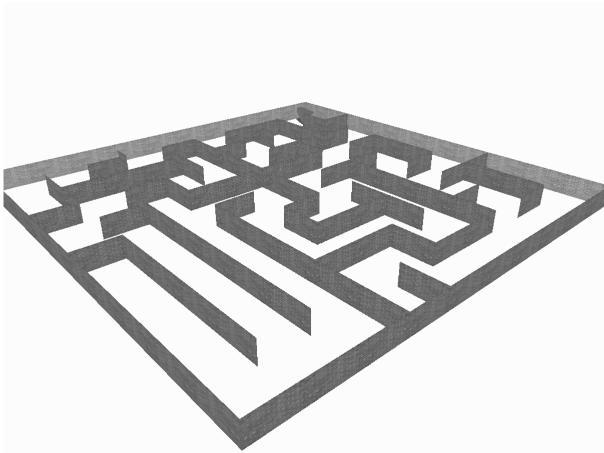


Figure 3: A third person view of the maze

Displaying only a single view limits the interaction. The view in Figure 2 allows easy local interaction but difficult navigation and the view in Figure 3 has easy navigation at the expense of directness. Navigation in Figure 2 is difficult because the choice of route is unclear without exploration. Route exploration can be time consuming, and is undesirable in most real world (non-entertainment) applications.

To alleviate the need to explore without removing the directness of a first person view, multiple viewpoints can be used. Figures 4 and 5 provide a discrete combination of viewpoints. Navigation is possible with a view that provides context (the smaller view in both figures), and local interaction is easy through the larger first person view.

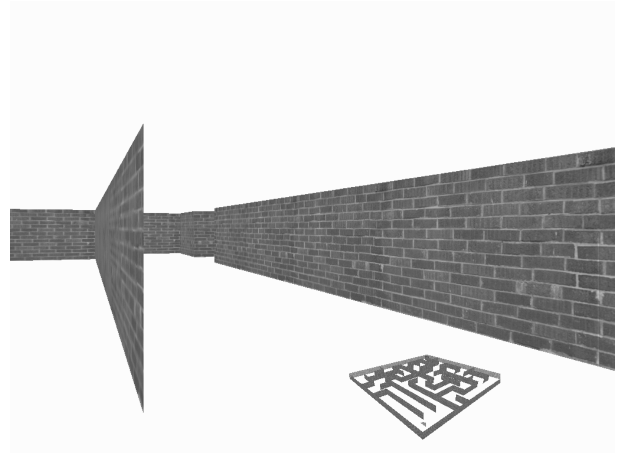


Figure 4: A World-in-Miniature view

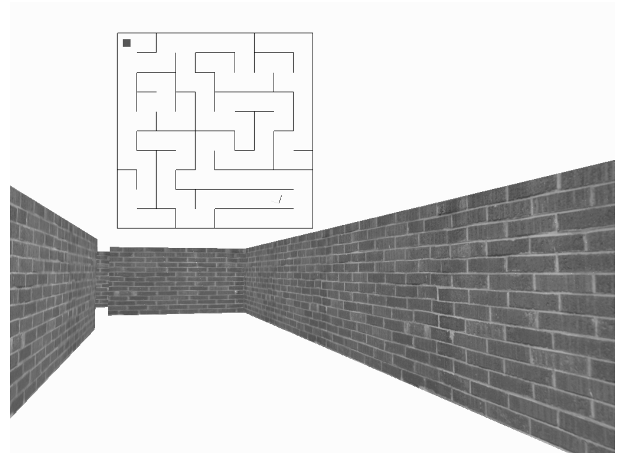


Figure 5: Plan view and first person view combined

Figures 4 and 5 suffer from a need to switch from one view to another, this can be quite disconcerting as is discussed in [PAUSCH1995]. In their paper they present a method of specifying navigation with World-in-Miniature views to alleviate this problem. Once the user specifies a path, the context view rotates around and zooms in to become the first person view. There still exists a fundamental separation in views with this technique, and rapid swapping between the context and first person view, may become confusing.

Separation of focus tends to lead to confusion when the user changes focus. The user is unsure of how the viewpoints relate to one another. In tasks where the user's spatial awareness must not be compromised, such as planning and architecture, this can be a serious concern.

The next series of figures show the bending technique, with varying bending *thresholds* (see Section 4).

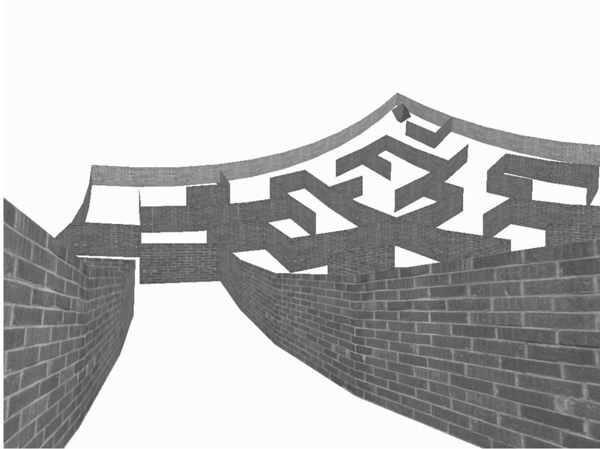


Figure 6: Bending navigation with a 'close' threshold.

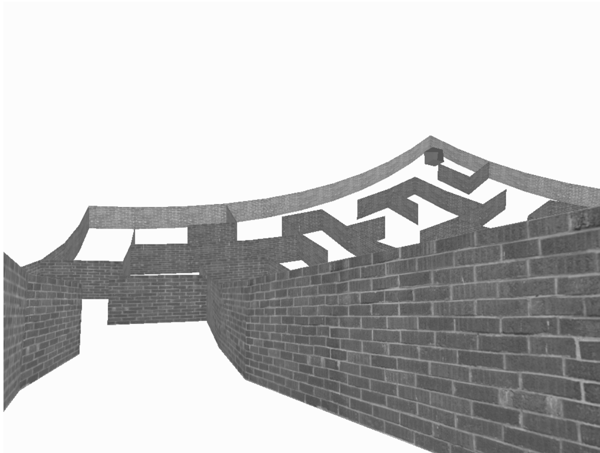


Figure 7: Bending with a 'medium' threshold.

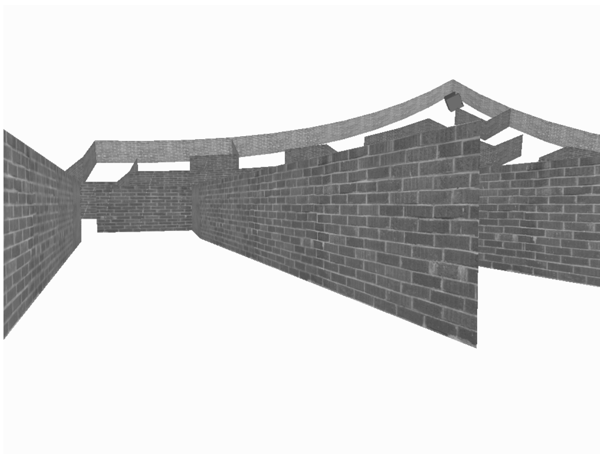


Figure 8: Bending with a 'far' threshold.

Unlike the plan view, the distorted view does not show all the maze. This means it is not a perfect tool for navigation, and may require, at some stages, the user to move around to gain the appropriate context. However continuity between navigation information and direct viewing is not compromised with this technique.

Scenarios in which navigation data and local environment data are intertwined present a problem for this technique. For instance, when seeing if a close object aligns with a couple of more distant objects, bending will make this impossible. A solution would be to have some areas distorted and others left alone, but specifying this would be difficult.

4. Implementing the technique

To make a smooth transition from the first person view, to the distorted view, each vertex is rotated separately according to its distance from the user. This distance is measured only in terms of the plane the data lies on, so the effect is similar to bending the plane itself. The plane of the maze is simply the *xz*-plane. This means that the vertical displacement of the vertex from the user has no effect on its rotation. Figure 9 shows the idea.

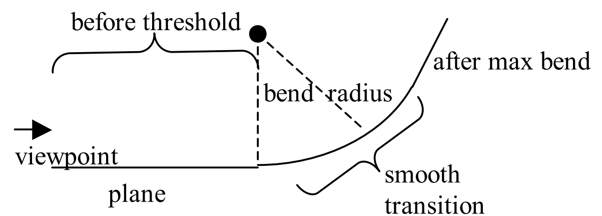


Figure 9: Schematic of the plane after bending

In general, to find the distance from the user in terms of some plane, first project both the user's position and the vertex onto that plane. This is easy in the specific case where the plane is the *xz* plane. All that needs to be done is to discard the *y* axis value. So the distance measure is:

$$d = \sqrt{(x_v - x_p)^2 + (z_v - z_p)^2}$$

Where:
d = distance
x_v = x coordinate of the viewpoint
z_v = z coordinate of the viewpoint
x_p = x coordinate of the vertex
z_p = z coordinate of the vertex

Once the distance is obtained it is checked against a threshold value. The threshold specifies where the bending effect should begin. Additionally there is a radius value for the bend, which effects how large the smooth transition area is. Once the point is further than the distance threshold plus the radius, it is simply rotated by the maximum amount. So the amount of rotation is:

$$\begin{cases} 0 & d < t \\ \dfrac{d - t}{r} & d < t + r \\ m & otherwise \end{cases}$$

Where:
d = distance
t = threshold
r = radius
m = maximum bend

The amount of rotation increases linearly over the *bend radius*. Non-linear relationships between rotation and distance have subtle effects on the technique. A non-linear function can be used to instigate a slow onset of rotation followed by a fast climax. This would look like:

$$\begin{cases} 0 & d < t \\ \left(\dfrac{d - t}{r}\right)^{f} & d < t + r \\ m & otherwise \end{cases}$$

Where:
f = some exponent, larger values mean a quicker transition
d = distance
t = threshold
r = radius
m = maximum bend

The point and axis of rotation depend on the exact kind of bending effect desired. In the prototype system, vertexes are distorted in a fashion similar to if they were projected onto a portion of a cylinder. It differs slightly from a cylinder in that vertices are distorted more according to their distance, even if they are the same distance from the start of the cylinder. The cylinder runs perpendicular to the direction of the view. So for each vertex, the axis of rotation is also in the direction of the view vector. The point around which the rotation occurs is above the closest point from the vertex, on the line perpendicular to the user in front of them and *threshold* distance from them. This can be seen in figure 10.

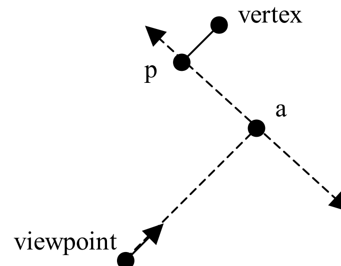


Figure 10: Determining the point of rotation for a cylindrical bend

In this diagram, *a* is the point *threshold* distance from the user, and *p* is the point closest to the vertex on the line perpendicular from the viewpoint. So the point of rotation is above *p* by the amount of the bend radius. In the case of the plane being the *xz* plane, above simply means setting the *y* value to being that of the bend radius. If the plane is arbitrary, then the point of rotation is above the plane (with respect to the viewer) in the direction perpendicular to the plane. For a spherical bend, the point of rotation would simply lie on the line between the viewpoint and the vertex, projected onto the plane.

Once the axis of rotation, the point of rotation and the amount of rotation are established it is simply a matter of using any maths library to implement the transformation. Unfortunately, the OpenGL API does not allow rotation transformations to be applied to vertices separately in a triangle [SEGAL1998]. If the rotation were to be applied at triangle level then splits would occur between the triangles of the scene that were previously flush against one another.

This technique is applied per frame on a per vertex basis. Consequently performance is a concern. The technique should be applied in conjunction with view frustum culling, so only a visible triangle's vertices will be modified.

4.1 Using the Technique

It is clear from this example that certain conditions need to be met for the technique to be helpful. Firstly, the data needs to be able to be seen from a plan view. If the data

has a roof or covering, context is still going to be obscured. The technique is not applicable to arbitrary 3D data sets, and as with any technique should only be used when the benefits are clear. Secondly, data to be presented in a first person view must be close to the viewpoint. Some data may be distant from the user but not obscured. This data will be distorted, even though it would have been seen without distortion.

In the maze scenario the data is dense and regularly distributed. In a sparsely populated data set, the effect of the technique may be unclear. In these circumstances it may be advantageous to render a 2D grid on the plane of bending. In general, the effect of any technique on data should be directly visualized so that it is clear what is inherent in the data and what is a result of the technique.

The amount of context can be varied by the *threshold* before the bend and the angle of the bend. A steeper and closer bending section decreases the similarity to a normal first person view, but increases the context. This raises the question of user specification of such parameters. Varying the bend threshold interactively may well be an effective way of controlling the amount of context. However other parameters may be less intuitive to manipulate. The type of bend used, whether it is cylindrical or spherical, and the acceleration of rotation (whether the bend is smooth or not) may well change the efficacy of the technique. Allowing changing of these parameters may unnecessarily complicate the interface.

4.2 Applications

Many interactions that are constrained to traversal on a surface are likely to have the context obscured by detail that projects out from the surface. This is particularly prevalent where there is a human presence metaphor being used in the interaction. Furthermore constrained navigation in 3D can be much more effective than free flying as generally only 2D input devices are used. For example Hanson and Wernert [HANSON1997] present a system in which all 3D navigation is constrained to an arbitrary 2D guide manifold, which is more intuitive than free-flying.

Some examples of possible real applications for this technique are the navigation and display of Geographic Information Systems (GIS) data for mining, planning and resource management, and real world navigation aided by augmented reality.

GIS data tends to be very large and planar, and as such would lend itself well to navigation by bending. Augmented reality allows virtual data to be displayed over real vision. A map could be overlayed over the top of real scenery and then distorted in the distance to provide easy navigation. Essentially anywhere the data is mainly on a single plane, and the data in the distance provides information necessary for navigation, is a potential application area.

5. Conclusion

Every task has different requirements, and no one technique can provide all the solutions, but for certain problems the technique presented in this paper can provide context without separate viewpoints. This is particularly useful where distant data is to be used for navigation, and close data needs to be viewed to facilitate local interaction. In applications whose data mimics the real world this technique is most likely to be valuable.

This paper provides an exploration of concept and a guide to the implementation of a novel navigation technique. As with all new techniques, its value depends on the applications found for it. Considering the wealth of first person, virtual human, interactions, there is the potential for quite a number of applications that can benefit from the concepts explored herein.

6. Future Work

The task presented in this paper will be extended and presented to a series of users to gauge the distorting technique's effectiveness. It is expected that timing results for the task, as well as user feedback, should give a good indication of the relative merit of the techniques. Following this, the extension of the technique to a more realistic task will be considered.

Further areas of work raised by this paper stem mainly from the implementation and specification of different kinds of distortions. Arbitrary distortion of certain regions and not others can enable easier navigation in a wider range of applications. However, how these arbitrary distortions are specified, whether by the user or designer, is an untackled problem. In such complicated environments it would most likely be necessary to directly visualize the distortion itself. This may be done in the form of a grid, but other solutions would have to be investigated.

7. References

[HANSON1997] Hanson, A. J. and Wernert., E. Constrained 3D navigation with 2D controllers. In *Proceedings of Visualization '97*, pp. 175-182. IEEE Computer Society Press, 1997.

[KEAHEY1998] Keahey, T.A. The Generalized Detail-In-Context Problem. *Proceedings of the IEEE Symposium on Information Visualization.* IEEE. 1998.

[PAUSCH1995] Pausch, R. Burnette, R. Brockway, D. and Weiblen, M. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. *Proceedings of the 22nd annual ACM conference on Computer graphics*, pp 399 – 400. 1995.

[SEGAL1998] Segal, M and Akeley, K. The OpenGL Graphics System: A Specification (Version 1.2). ftp://sgigate.sgi.com/pub/opengl/doc/opengl1.2/opengl1.2.pdf). 1998

[STOAKLEY1995] Stoakley, R, Conway M. and Pausch, R. Virtual reality on a WIM: interactive worlds in miniature. *Conference proceedings on Human factors in computing systems.* pp.265 – 272. 1995.

[WINCH2000] Winch, D., Calder, P. and Smith, R. (Focus + Context)3: Distortion-orientated displays in three dimensions. In *Proceedings of the 1st Australasian User Interface Conference AUIC 2000,* Canberra, Australia, Jan-Feb 2000, pp. 126-133.

Appendix F

Multi-Perspective Images for Visualisation

Multi-Perspective Images for Visualisation

Scott Vallance **Paul Calder**

School of Informatics and Engineering
Flinders University of South Australia
PO Box 2100, Adelaide 5001, South Australia

{vallance,calder}@infoeng.flinders.edu.au

Abstract

This paper describes the concept, and previous realisations, of multi-perspective images in nature, art and visualisation. By showing how distortions have been used for visualisation, it motivates the use of multi-perspective images, which are similar in effect to object based distortions. A new API being developed to facilitate multi-perspective rendering is presented, with particular reference to its suitability for interactive applications. This API is demonstrated in a simple example of a multi-perspective image, where five faces of a cube are shown at once. Further work necessary to make multi-perspective images for visualisation a reality is discussed.

1 Introduction

We define a multi-perspective image as multiple views of a single scene from different perspectives. These views are joined seamlessly to form an image that is a coherent whole, without discrete subsections. The concept of continuously joined views is not new. The idea has been realised in many different forms; for example, reflections on curved objects and lens effects can constitute natural multi-perspective views. Apart from the inherent aesthetics of the concept, we seek to explore its use as a visualisation tool.

The motivation for exploring multi-perspective rendering comes from the limitations of human sight in a 3D world. The view we can see from our two eyes can be in many situations very limited. We cannot see over tall objects, nor through opaque objects, and we cannot see forwards and backwards at the same time. In a virtual world, the first two limitations can easily be dismissed – we can fly over any object, and turn what we wish transparent. The third limitation is what this paper specifically addresses. We seek not just to be able to see forwards and backwards at the same time, but to see from different places – without having multiple separate views.

Real world multi-perspective images are used as visualisation aids already. An obvious example is curved mirrors on roads, which provide views of both directions at T-junctions. We wish to allow for even more flexibility in virtual worlds, so that visualisers can effectively look out from arbitrary curved mirrors.

2 Previous Work

Previous work falls into three sections: work relating to viewing from multiple viewpoints continuously, techniques for distorting objects, and methods for rendering reflections on curved objects. While the previous work on multi-perspective images gives a clearer understanding on the nature of the images, it is the work on distortions that prompts our visualisation focus. Previous work on reflections on curved objects relates to the practical implementation of multi-perspective rendering.

2.1 Multi-Perspective Images

In art, Chinese landscape painting, Cubism, and M. C. Escher have explored the idea of multi-perspective images. Chinese landscape paintings contain different focuses, or sub-images, which are seamlessly joined. These paintings are similar to the panoramas used for cartoon drawing and image resynthesis, as is discussed in Chu and Tai (2001). For example, in Figure 1 the perspective shifts from left to right, following the path of the stream.

M. C. Escher depicted a view with multiple vanishing points, or perspectives, in his work "High and Low" Escher (1992). This work, see Figure 2, has five different vanishing points: top left and right, centre, and bottom left and right. The top section of the image presents a view from above, and the bottom section a view of the same image but from ground level. While the automatic generation of an image like this from 3D geometry may not be practical, it illustrates the concept and the aesthetic potential.

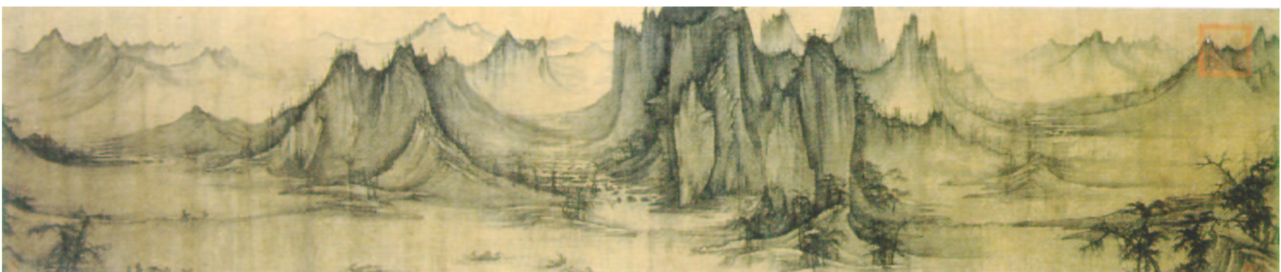


Figure 1: "Fisherman's Evening Song" by Xu Daoning, Circa 11th Century.

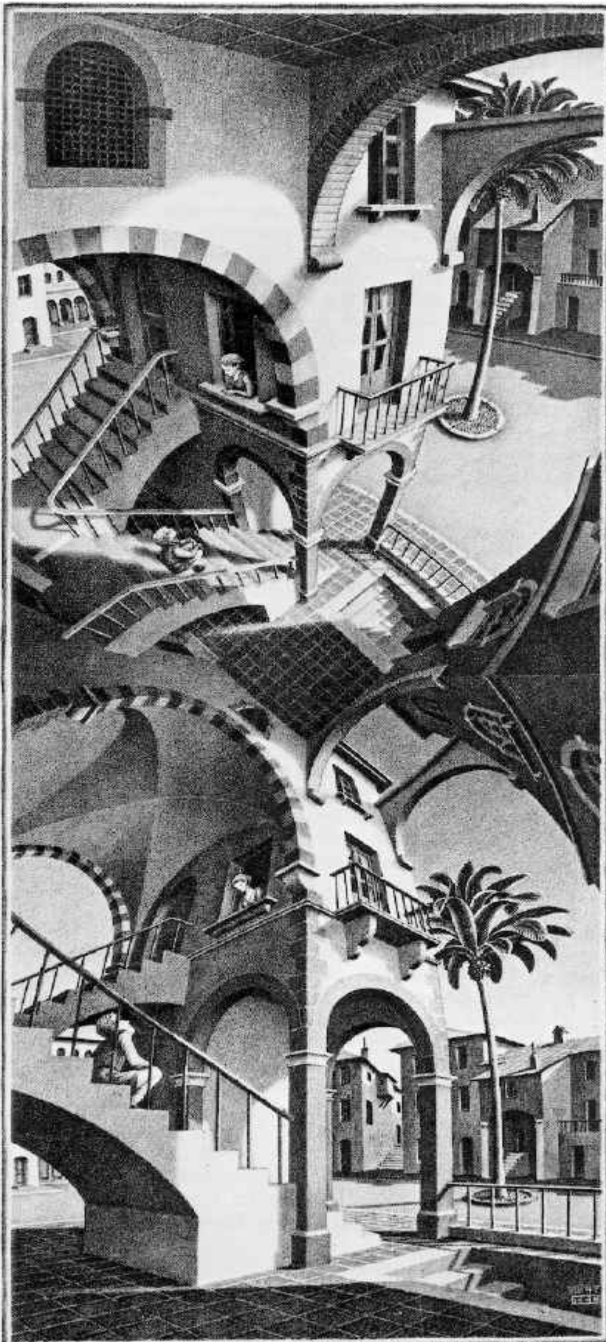


Figure 2: "High and Low" by M. C. Escher

Hand-drawn and computer-generated panoramas with multiple points of view have been used as the basis of image resynthesis. Cartoon animation from panoramas is an early example of resynthesis, which is explained and adapted to computer-generated images in Wood, Finkelstein, Hughes, Thayer and Salesin (1997). When a small subsection of the panorama is viewed it approximates a standard single viewpoint. If these subsections are taken over a path on the panorama, they give the appearance of motion when sequenced into an animation. This is because the viewpoint shifts continuously in a multi-perspective panorama.

Rademacher and Bishop (1998) present more generalised multi-perspective images as the basis for resynthesis, the advantage being a variable level of sampling without multiple separate images. The paper calls these Multiple-Centre-Of-Projection images. One section of the panorama can be from close to a portion of the image, giving a high sampling for that area, while others are further away and capture more of the object. Moving a virtual camera through the scene generates the multi-perspective panorama. At regular intervals the camera captures a single line of pixels for the final panorama. These lines, either rows or columns, are placed next to each other, so that the viewpoint smoothly changes from one to the next. This is effectively a virtual strip camera, which is explained below. Figure 3 shows a Multiple-Centre-Of-Projection image of an elephant. The virtual camera path goes from one side of the elephant to the other, and we can simultaneously see both sides and the front.

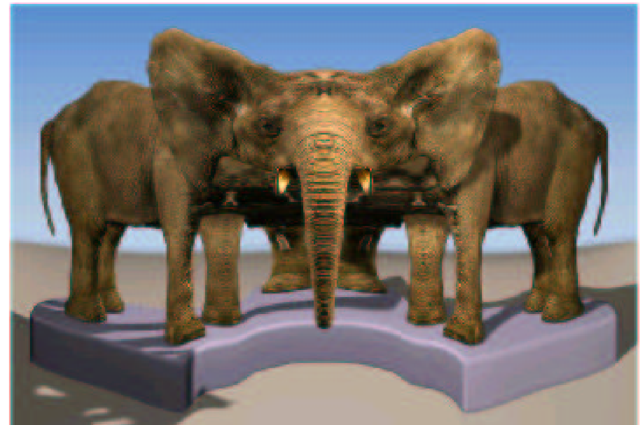


Figure 3: Multi-Centre-Of-Projection image of an elephant taken from Rademacher and Bishop

Strip cameras are used in surveillance and mapping. These cameras have a continuous roll of film that slides past a slit as a picture is being taken. The camera may be moved whilst the shooting, providing a change in point of view from one section of the film to another. If used from a moving plane these cameras can capture a long section of curved earth as if it were flat. The cameras have also been used for artistic purposes, capturing strange and unusual images, such as in Robert Davidhazy's work (Figure 4).

Figure 4: A strip camera image showing a head from all sides taken from Davidhazy (2001)

In Loffelmann and Groller (1996) the idea of rendering from multiple viewpoints with ray tracing is examined. By developing an extended camera for ray tracing, the authors present multi-perspective images with visualisation as an application. Essentially an extended camera is a set of 3D rays. Each ray has a starting position and a direction, and these rays are used by a conventional ray-tracer to draw the scene. Unlike a

conventional ray-tracer, these rays do not necessarily originate from the same point.

Figure 5: A conventionally rendered set of columns

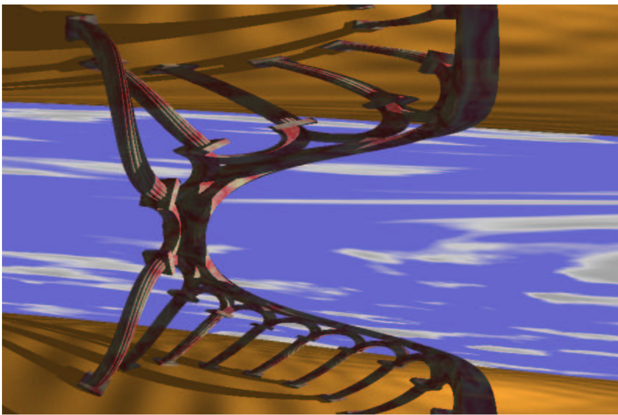


Figure 6: Columns rendered from a torus surface

Figures 5 and 6, both taken from Loffelmann (2001) show the extended camera ray tracing in practice. The first image is rendered normally, however the second is rendered out of a torus camera. The rays used all start on the surface of the torus and point in the direction of the surface normal.

The extended camera is comprised of three sections: an object space transformer, a picture space transformer and a parameter space transformer. The object space transformer is a function that returns the 3D start position of a ray given its corresponding index on the 2D screen. This function defines a surface in 3D, and the scene is rendered "out of" this surface. Defining a multi-perspective image from a rendering surface is a convenient representation, so we have adopted it for our work. A multi-perspective image could otherwise be defined as a camera path, such as in Rademacher and Bishop (1998) described previously. However, the camera path can be described as a surface, and vice versa.

2.2 Distortions

Distortions of the data, while being fundamentally different in implementation than rendering multi-perspective images, highlight the potential and applications for multi-perspective images. Both seek to present the data in a changed way so that previously unseen properties become apparent.

Distorting an object to view it better is most commonly illustrated with the Mercator projection. This takes a sphere, generally the earth, and transforms it to a 2D map. In this map directions are conserved, though sizes are not, to allow for easy sea navigation. Although the distortion of size is an artefact, it allows for a better understanding of some aspects of the globe. This is ultimately the point of distorting the data, either directly or by rendering through a curved surface – to better illustrate certain properties of the data.

In Hurdal, Bowers, Stephenson, Sumners, Rehm, Schaper and Rottenberg (1999) a scan of a brain was distorted into a nearest approximation flat projection. This allows a better appreciation of the layout of the brain from medical imaging, with the intention of improving surgical planning. This type of distortion is highly dependant on the nature of the data, and does not translate well to the visualisation of arbitrary scenes.

Distortion Orientated Displays are a general visualisation tool based around the distortion of data. These displays seek to show detail and context simultaneously. The general problem is that when detail is shown, much of the screen is filled with that detail. If the surrounding data is shown at the same level of detail, it would not fit on the screen. To accommodate this, in a distortion view there is a region of focus at a certain detail level, which smoothly transitions to a region of context at a lower detail level. This can be seen in Smith (1997). Using a distortion called a frustum display, the author was able to achieve levels of detail sufficient for a city level road map, whilst showing the context of the whole of Australia.

In 3D, both Keahey (1998) and Winch, Calder and Smith (2000) expanded the idea of distortion orientated displays to allow for regions of zoom – regions where the scene data was expanded to a larger size. These prove useful for highlighting sections of particular detail in a scene without zooming and therefore cutting out periphery data.

The idea of detail and context is to have at least two different perspectives on the data. In this case the perspectives are not literal changes in viewpoint, but in operating conditions. In Vallance and Calder (2001) the idea of distorting a mainly planar world onto the inside of cylinder was examined. This was proposed, in the application of virtual maze navigation, so that two different perspectives on the maze could be simultaneously realised: a local view of the undistorted maze walls, and a navigational view of the distant maze perpendicular to the users viewpoint. Figure 7 shows an example of the maze distortion.

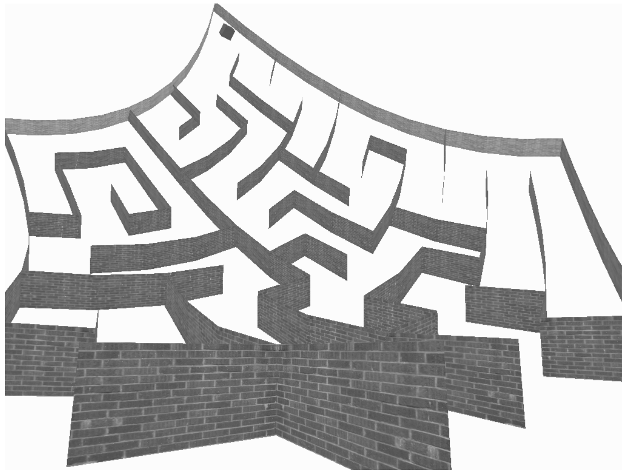


Figure 7: A maze distorted in a cylindrical fashion to show context

The cylindrical distortion works only because the data lies mainly on a single plane. For arbitrary 3D data it is unclear how the mapping onto a cylinder would be helpful.

2.3 Reflections on Curved Objects

Reflections on curved surfaces are a natural form of multi-perspective image. Computer graphics has long been interested in realistically rendering virtual scenes including reflections. Research into generating reflections on curved objects, especially for real-time graphics, shows how multi-perspective rendering can be implemented. However reflections on curved objects are a subset of multi-perspective images. Moreover they need not be entirely accurate (they need only look appropriate) and form only a small portion of the screen. Multi-perspective images for visualisation have different accuracy requirements and take up more of the screen.

Ray tracing is the most direct way to render reflections from curved surfaces. When a ray intersects a reflective surface it bounces off in a direction determined by the angle of intersection with surface. Reflections drawn in this manner are accurate, however ray tracing tends to be very slow. Various methods exist for accelerating the ray tracing. These methods range from reducing the number of ray-surface intersection tests with hierarchal space subdivision, to parallelising the calculations. Without massively parallel hardware it seems unlikely real-time ray tracing will be achieved soon, the problem is discussed in Jansen (1993).

Environment mapping was initially suggested by Blinn Blinn and Newell (1976), and is often used to approximate curved reflections for real-time graphics. When a scene is rendered from a particular viewpoint, the light coming into that point is sampled. With enough samples for a particular viewpoint, the technique can approximate the colour of any ray shot from that point. A popular implementation of environment mapping involves rendering to the six faces of a cube centred on a point. When used for generating reflections, this centre point is the centre of the reflective object. For each ray that bounces off the reflective object, a ray is shot from the centre point in the direction of the reflected ray into the cube. This approximation moves each reflected ray to the centre of the cube. For scenes where the reflected objects are far away from the reflective object the technique works well; otherwise the approximation is obvious. For the purposes of multi-perspective images environment maps are not sufficiently accurate. The light is sampled at only one point, making it unsuited for multi-perspective, or multi-viewpoint, rendering.

An extension to the concept of environment mapping is proposed by Cho (2000) to provide more accurate images. Instead of simply sampling the light coming into a point, a depth-mapped image is calculated for each of the six cube faces of the environment map. Reflection rays can then be traced into the 3D depth map, without needed to approximate the start point of the ray. This technique amounts to ray tracing the scene, though through a modified representation of the geometry (the depth map) that provides a significant performance enhancement in some cases. Static scenes are required for this technique, as otherwise the depth maps need to be recomputed at each frame, which is very expensive. In a multi-perspective image for visualisation, the surface will be moving and not static in relation to the scene, so extended environment maps would be unsuited.

Another type of technique based on environment mapping is presented in Hakura, Snyder and Lengyel (2001). In this technique, layers of environment maps are used. Different environment maps may be used according to the viewer's location and direction of view, to avoid the failings of standard environment maps. Once again, this technique requires a static scene relative to the reflective object to be effective. Rendering these reflections from a series of stored images is actually image based rendering. Other image based rendering techniques, such as the Lumigraph (Gortler, Grzesczuk, Szeliski and Cohen 1996) can be useful for rendering reflections. These techniques represent the light in the space of a scene, and are sampled with 2D slices to generate a particular view. The drawback of such a system is that it is currently not used for many visualisation purposes, has a large memory overhead, and needs an unchanging scene.

The most intriguing approximation of rendering reflections on curved surfaces is described in Ofek and Rappoport (1998). In this method, objects are transformed by the reflective surface so that they may be rendering from a single viewpoint. In essence the data is distorted to an approximation of how it will look after being viewed from a reflective surface, and then rendered. It requires an appropriate tessellation of both reflective surface and scene object so that lines that should now appear curved do. The performance of this technique is sufficient for real-time rendering of moderate scenes. The technique works on standard polygon scenes making it suited for visualisation tasks, and easy integration into current applications.

3 An API for Multi-Perspective Rendering

In developing an API to facilitate rendering from multiple perspectives the key concerns were:

- To separate, as much as possible, technique from calling interface.
- To allow easy integration into existing visualisation applications.
- To allow for expansion of functionality and techniques.

To achieve interactive rendering of multi-perspective images a variety of different techniques will need to be evaluated. The API is designed to flexible enough so that different techniques can be used without major changes to the visualisation code that uses the API.
One of the most popular API's for 3D graphics is Silicon Graphics Inc.'s OpenGL. It provides a clear and powerful set of instructions for building graphical programs. The specification is described in Segal and Akeley (1998). This API forms the inspiration for our design, and many of the commands resemble OpenGL syntax. By basing the API on OpenGL it should allow for easy integration into existing visualisation programs that use OpenGL.

The multi-perspective rendering API covers two main tasks: describing a surface to render from and a scene to be rendered. Two abstract classes define the base level interfaces provided by the API for these two tasks.

Geometry: Describes the scene to be rendered.

Functions	Description
Begin()	Start of a geometry description block. Vertex, normal and colour may only occur between a Begin() and an End()
End()	End of a geometry block
Vertex3f(float x, y, z)	Place a vertex of a triangle in the scene with the current normal and colour
Normal3f(float x, y, z)	Specify the current normal
Color3f(float a, r, g, b)	Specify the current color
Other	Various other commands to specify textures and other graphical properties

Surface: Describes the surface to render from.

Functions	Description
BeginSurface()	Start specifying a surface, vertex and normal commands may only appear between a Beginsurface and Endsurface
EndSurface()	End a surface specification block
Vertex(float x, y, z)	A vertex in the surface, the exact meaning is dependant on the type of rendering technique
Normal(float x, y, z)	Current normal for each vertex
Viewport(float l, r, t, p)	The viewing dimensions of the screen. Determines how the surface is mapped to the screen

These classes are accessed through the Renderer class, which is a conglomeration of all the interface functions so that they may be accessed without reference to particular Geometry or Surface objects. The rendering technique implemented in the Renderer class, which uses specific Geometry and Surface objects to generate an image. This is done so that the interface looks and feels more like OpenGL.

These abstract classes are inherited by specific implementations. For instance, the rendering of multi-perspective images can be done with a ray tracer, the surface can be specified as a set of rays and the geometry as a set of triangles. This defines three classes that inherit from the base classes: RayTraceRenderer, RaySet and TriStore. Figure 8 shows the relationships of the classes. The solid arrows indicate inheritance and the dashed arrows show usage.

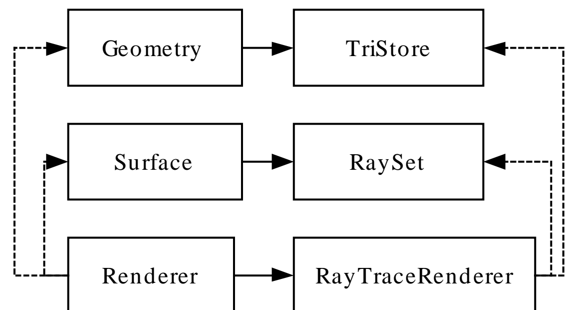


Figure 8: Diagram of base classes and a raytracing renderer

In the surface class, like the geometry class, the main mechanism for passing information to the lower level rendering technique is the vertex and normal commands. This symmetry is deliberate, so that, with the right technique, a surface can be used to render from or to draw with little modification. Also this OpenGL-style immediate-mode specification of surfaces suits an interactive application where the surface may be changing from frame to frame.

The types of rendering surfaces that can be represented easily with only vertex and normal commands is similar

to the types of conventional surfaces easily specified with those primitives. Simple point-based geometry is analogous to a simple rayset surface, where vertex and normal commands denote a ray's origin and direction.

Triangular patches are probably the most common form of 3D geometry and rendering surfaces can also be described in this manner. While polygon patches are not curved, it is common practice to make them appear so by interpolating properties between the vertices of the constituent triangles. By taking the normal values at a particular point as denoting the direction of a ray origin at that point, and interpolating these normals between the vertices, a graduated or curved surface is approximated.

Parametric surfaces are also easily specified with vertex and normal commands, with vertices interpreted as control points of a patch. It is hoped that these different ways of specifying surfaces will be sufficiently rich for most purposes.

To illustrate how the API works here is an example based on using the RayTraceRenderer described in Figure 8:

```
RayTraceRenderer rtr;

void SpecifySurface() {
  rtr.Viewport(0,0,1,1);
  rtr.BeginSurface(); {
    rtr.Normal3f(0.0,0.0,-1.0);
    rtr.Vertex3f(0.0,0.0,0.0);
    rtr.Normal3f(0.0,0.0,1.0);
    rtr.Vertex3f(0.0,0.0,-1.0);
  } rtr.EndSurface();
}

void SpecifyScene() {
  rtr.Begin(); {
    rtr.Normal3f(0.0,0.0,1.0);
    rtr.Vertex3f(1.0,0.0,-1.0);
    rtr.Vertex3f(-1.0,0.0,-1.0);
    rtr.Vertex3f(0.0,1.0,-1.0);
  } rtr.End();
}
```

These functions, appropriate called, trace two rays into a scene comprising of a single triangle. The RaySet and TriStore objects are hidden by the RayTraceRenderer and are accessed through that class.

4 Raytracing Implementation

As a first case implementation a ray tracing algorithm was used, with rendering surfaces defined as a set of rays. This implementation was used to render a simple scene with a cube a tiled floor, shown here in Figure 9. While basic, the scene has some properties that make it an appropriate demonstration of multi-perspective rendering. First the cube has six numbered sides, in a normal perspective view at most three sides can be seen at once, due to self occlusion. Second, the tiled floor provides a reference to the effect of the rendering surface that is easily perceived.

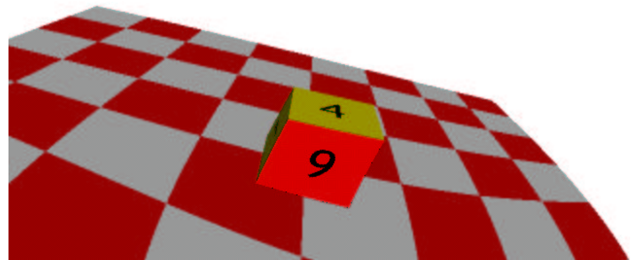


Figure 9: The cube scene

To simultaneously show five sides of the cube, a curved surface is placed over the scene. This surface is constructed initially from 16 control point Bezier patch, which is decomposed in a set of rays. The surface can be seen here in Figure 10, and is roughly hemispherical in nature. Figure 11 shows the cube scene rendered through the surface, with five of the cube's faces visible.

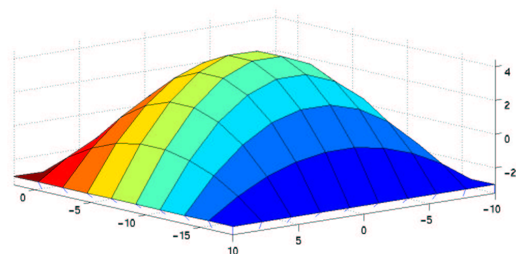


Figure 10: A surface for visualising all sides of a cube

The surface spans the cube scene, with the rays pointing inwards in a direction normal to the surface. The generation of the rays from the Bezier equations and control points is a significant performance cost in itself.

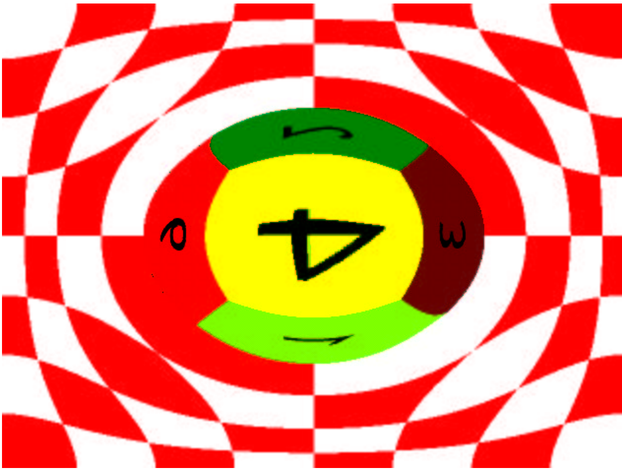


Figure 11: The cube scene rendered from the surface in Figure 10

The surface was constructed by manually entering appropriate values for the Bezier control points. This is undesirable from an interaction point of view, one of the key aims of our research. As a first experiment in the interactive specification of rendering surfaces, a 1-dimensional control is proposed. A slider dictates the amount of perspective on a normal rendering surface. A zero value corresponds to an orthographic projection, and the positive values correspond to an increasingly severe perspective projection. Negative values correspond to a 'reverse' perspective projection where distant objects seem larger, and closer objects smaller. Figures 12 through 14 show a perspective view, an orthographic view and finally a 'reverse' perspective, respectively.

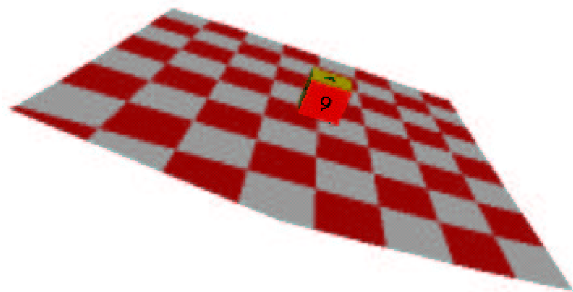


Figure 12: A perspective projection

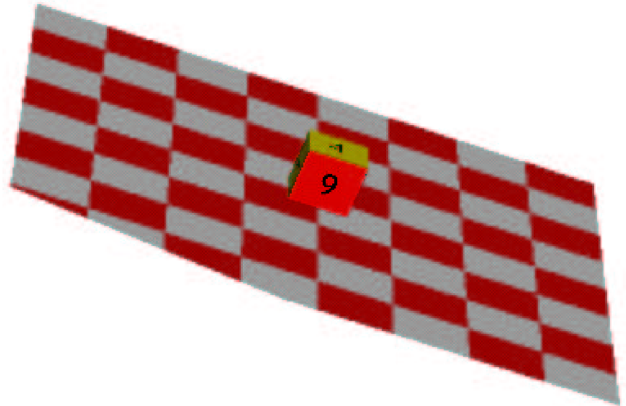


Figure 13: An orthographic projection

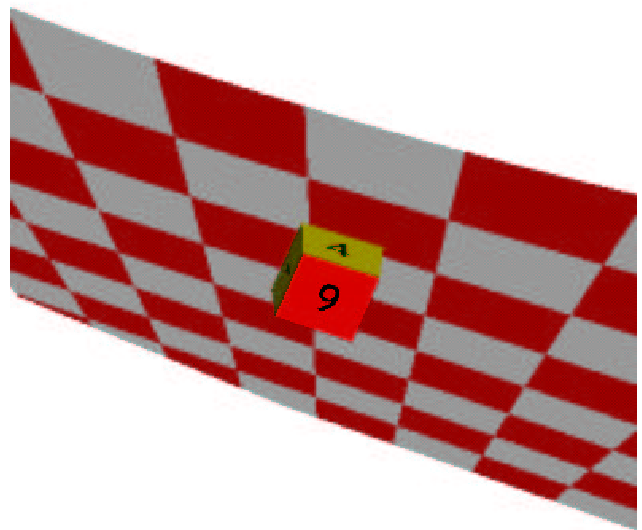


Figure 14: A 'reverse' perspective projection

The floor of the cube scene appears to be upside down in Figure 14, as the distant top edge is now larger than the closer bottom edge. A simple 1-dimensional slider barely captures the possibilities of interactively controlling the rendering surface, but does allow for an evaluation of such things as API design and system performance. The OpenGL like multi-perspective rendering API is well suited to interactively specified rendering surfaces because the surface is specified in a manner similar to that of dynamic geometry.

The rendering performance of the ray tracing implementation of the API is too slow for interactive performance. Frames take seconds to render each, even over such a simple scene. The ray tracing implementation is admittedly naïve, and does not include such common speedups as BSP culling of scene geometry. The API is designed to easily allow for the development and integration of new techniques, and these are already being worked on.

5 Conclusions and Further Work

While the performance of the API in terms of frames per second is far from interactive, the design and conceptual groundwork has been laid for more detailed investigation into multi-perspective images from visualisation. By extending previous work on interactive reflections on curved objects to multi-perspective images for visualisation performance issues will be addressed. The raytracing implementation described in this paper forms an important base line for measuring the accuracy of faster implementations.

The interactive manipulation of viewing surfaces is an unexplored field, the simple 1-dimensional interaction described here is a start, though things will be much more complicated with more dimensions of freedom. Another aspect this paper does not touch on is in the useful application of multi-perspective images. The cube world in Figure 11 demonstrates clearly one of our reasons for pursuing multi-perspective images, the ability to see more than is otherwise possible. With appropriate tools and interfaces we believe multi-perspective images will be a valuable tool for visualising complex data.

6 References

BLINN, J.F. and M.E. NEWELL (1976): Texture and Reflection in Computer Generated Images. *Communications of the ACM*, v19, n10, p542-547.

CHO, F. (2000): *Towards Interactive Ray Tracing in Two- and Three-Dimensions.* PhD Thesis. University of California at Berkeley.

CHU, S. H. and C. L. TAI (2001): Animating Chinese Landscape Paintings and Panorama using Multi-Perspective Modeling, *Proceedings of Computer Graphics International 2001*, Hong Kong, IEEE Press.

DAVIDHAZY, A. (2001):, *Peripheral Portraits and Other Strip Camera Photographs*, Retrieved October, 2001: http://www.rit.edu/~andpph/exhibit-6.html .

ESCHER, M. C. (1992): *The Graphic Work.* Evergreen, Germany.

GORTLER, S., R. GRZESCZUK, R. SZELISKI and M. COHEN (1996): The Lumigraph. In *Proceedings of SIGGRAPH 96,* pp. 43-54.

HAKURA, Z., SNYDER, J., and LENGYEL, J. (2001): Parameterized Environment Maps. *In Proceedings of the 2001 Symposium on Interactive 3D Graphics, I3D01.*

HURDAL, M. K., P. L. BOWERS, K. STEPHENSON, D. W. L. SUMNERS, K. REHM, K., SCHAPER, D. and A. ROTTENBERG (1999): Quasi-conformally flat mapping the human cerebellum, in C. Taylor and A. Colchester (eds), *Medical Image Computing and Computer-Assisted Intervention*, *Vol. 1679 of Lecture Notes in Computer Science*, Springer, Berlin, pp. 279-286.

JANSEN, F. W. (1993): Realism in real-time? In *Proceedings Fourth Eurographics Workshop on Rendering,* Cohen, Puech and Sillion (eds).

KEAHEY, T. A. (1998): The Generalized Detail-In-Context Problem. *Proceedings of the IEEE Symposium on Information Visualization.* IEEE.

LOFFELMANN, H. and E. GROLLER (1996): Ray tracing with extended cameras. The *Journal of Visualization and Computer Animation*, 7(4): pp. 211-227.

LOFFELMANN, H. (2001): *Diploma Thesis, "Extended Cameras for Ray Tracing"*. Retrieved October 1, 2001, http://www.cg.tuwien.ac.at/~helwig/projects/dipl/

OFEK, E. and A. RAPPOPORT (1998): Interactive reflections on curved objects. In *Proceedings SIGGRAPH 98*, pp 333-342.

RADEMACHER, P., and G. BISHOP (1998): Multiple-center-of-projection images. In *Proceedings of SIGGRAPH 98*, pp 199-206. ACM.

SEGAL, M., and K. AKELEY (1998): The OpenGL Graphics System: A Specification (Version 1.2). ftp://sgigate.sgi.com/pub/opengl/doc/opengl1.2/opengl1.2.pdf

SMITH, R. (1997): *Distortion Oriented Displays for Demanding Applications.* PhD Thesis, Gippsland School of Computing and Information Technology, Monash University.

VALLANCE, S. and P. CALDER (2001): Context in 3D Planar Navigation. In *Proceedings of the Australasian User Interface Conference AUIC 2001,* Queensland, pp. 93-99.

WINCH, D., P. CALDER, and R. SMITH (2000): Focus + Context^3: Distortion-orientated displays in three dimensions. In *Proceedings of the 1st Australasian User Interface Conference AUIC 2000,* Canberra, Australia, pp. 126-133.

WOOD, D., A. FINKELSTEIN, J. HUGHES, C. THAYER, and D. SALESIN (1997): Multiperspective Panoramas for Cel Animation. In *Proceedings of SIGGRAPH 97 Conference*, pp 243-250, ACM

Appendix G

Inward Looking Projections

Inward Looking Projections

Scott Vallance Paul Calder
School of Informatics and Engineering
Flinders University of South Australia
{scott.vallance|calder}@infoeng.flinders.edu.au

Abstract

This paper presents a technique for rendering inward looking projections. We call these projections anti-perpsective because they run counter to normal perspective projection and converge to a point in front of the image plane. A real world analog for this type of projection is the reflection seen on a spherically concave mirror. In fact the technique can be used to render reflections on perfectly spherically curved mirrors. Our implementation builds on environment mapping work that captures a sample of the light coming into a point; we propose an inward looking spherical camera that captures a view from all around an object. The aim of this research is to provide a visualisation technique capable of viewing multiple sides of an object in a single image.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation – Viewing Algorithms, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Texture.

Keywords: projections, reflections, environment mapping.

1 Introduction

Unlike many graphics techniques we are not concerned with matching real world phenomena. Instead this work comes about from purely visualisation motivated approach. Our aim was to provide information otherwise unavailable. Specifically we conceived the anti-perspective projection as a way to see both sides of an object at once.

This projection works similarly to looking into a spherically concave mirror, that surrounds an object close to the mirror. The view shows more of the object than can be seen than in a perspective projection. Taking this idea to its conclusion, we propose a fully spherically projection that totally surrounds objects within it. When using these projections to view an object, properties that span the entirety of the object, such as surface details, can be seen in a single view. The result is somewhat similar to a Plate Carree projection of our globe, though for arbitrary objects.

2 Map Projections

A Plate Carree projection of the globe maps the sphere onto a cylinder that is then unraveled. Essentially the Plate Carree projection is a longitude/latitude grid, as discussed in Birch [1964]. Although somewhat lacking as a cartographical tool, it does adequately project a 3D sphere onto a plane so that all angles may be viewed. Such mapping techniques effectively take a 3D point and map it to a position on a 2D plane. With a polygon model which is defined by its vertices, applying the projection on each vertex leaves the intervening detail, such as any textures, incorrectly transformed. In computer graphics, projections like this are commonly used to make maps that can guide the production of textures. In this context it is called UV mapping. This paper provides a technique to achieve similar results that are correctly transformed all across an object and not just at the vertices.

3 Perspective Projection

A traditional perspective projection mimics the way our eye sees the world so that distant objects appear smaller. In ray tracing terms, rays emanate from the eye-point and diverge from one another. This view can be implemented with a simple transformation that projects 3D points onto a plane representing the screen, along the line to the viewpoint. Details of the transformation can be found in graphics references such as Foley, van Dam, Feiner and Hughes [1990].

Conventional perspective projections use a z-buffer to ensure that only visible surfaces are drawn. A z-buffer stores a depth value for each pixel and typically only overwrites a pixel if the new value has a smaller depth value. This ensures the final value of the pixel has come from the 3D surface closest to the viewpoint. This depth is stored in a finite number of bits so a maximum and minimum depth need be established. Along with the constraints of the screen size these measurements define a viewing volume (see Figure 1). Any 3D data within that volume can be projected; anything outside is culled.

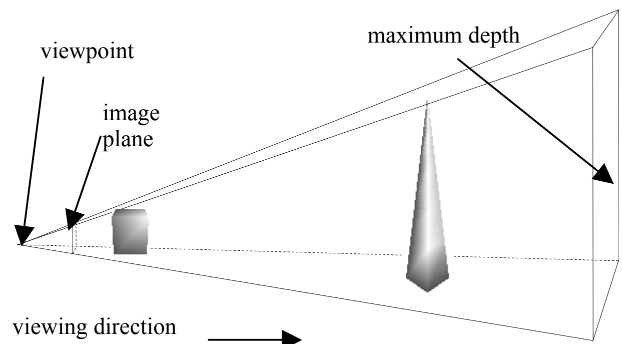


Figure 1: A Perspective Projection Viewing Volume

Note that in Figure 1, both the pyramid and cube would be roughly the same height in the rendered image. The section from

the image plane to the viewpoint is not actually part of the viewing volume, but is drawn to show how the volume converges to the viewpoint.

4 Anti-Perspective Projection

In an anti-perspective projection there is no viewpoint; instead there is a point of convergence. In ray tracing terms, rays emanate from the image plane and converge to a point in front of it. The viewing volume this creates is shown in Figure 2.

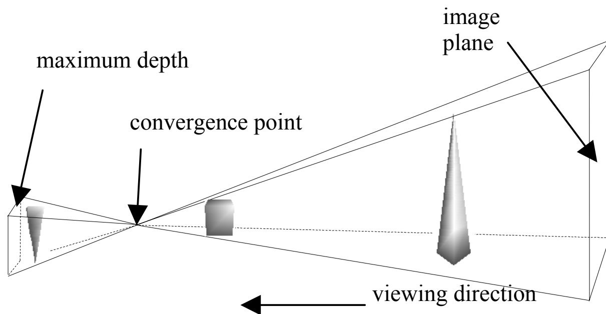


Figure 2: A Anti-Perspective Viewing Volume

The anti-perspective projection divides the scene into two parts, the near field (before the point of convergence) and the far field. For the far field volume everything we see will be upside down but otherwise like a standard perspective projection. For the near field volume more distant objects will appear larger than closer ones.

4.1 Near Field Viewing Volume

Our algorithm for rendering the near field viewing volume takes advantage of the similarities between this view volume and a perspective view volume. The only difference is that we wish to project the data onto the large end of the viewing volume instead of the smaller. In terms of the projection transformation itself the direction is irrelevant. This means we can render the volume as a perspective transformation starting at the point of convergence and ending in the image plane.

Direction is only important when determining the pixel depth. If we use a modified z-buffer algorithm that only overwrites a pixel if is further than the previous value, we can effectively draw only those faces which are closest to the far end of the volume. OpenGL provides *glDepthFunc()* to chose the type of depth test to be done. Passing a value of *GL_GREATER* will set the z-buffer to work in the desired manner.

Certain properties in rendering are direction sensitive, including lighting calculations and front-face determination through winding. For correct rendering these need to be adjusted to compensate for the backwards nature of the rendering. Winding direction needs be reversed and normal vectors pointed in the opposite direction. Our implementation uses the OpenGL mechanism for changing the winding used to determine front faces with *glFrontFace()*. Normal vectors can be multiplied by –1 to flip them.

The series of operations to render the near field view volume are as follows:

- Enable reverse depth buffering.
- Flip front face winding if applicable.
- Rotate the scene so the z-axis lies perpendicular to, and facing, the image plane
- Translate the scene so the viewpoint is the point of convergence
- Render the scene normally with normal vectors flipped.

4.2 Far Field Viewing Volume

Any objects after the point of convergence are upside down. This section of the viewing volume can be rendered as a perspective projection from the point of convergence. The maximum depth of the viewing volume is the end of the overall viewing volume and the minimum depth is set close to the viewpoint. If this minimum depth is too large, objects near the point of convergence may be culled effectively leaving a hole in the projection. The near field volume also has a hole between the minimum depth and point of convergence. As long as neither of these holes contains any vertices the projection will render correctly.

The far field view volume can be rendered as follows:

- Rotate the scene so that the z-axis is perpendicular to, and facing away from, the image plane.
- Translate the scene is so the viewpoint is the point of convergence.
- Rotate the scene 180 degrees around the z-axis.
- Render the scene.

4.3 Combining the Viewing Volumes

After rendering the near field view volume, the z-buffer depth values span a range of possible depth values with the maximum depth at the point of convergence. With appropriate use of *glDepthRange()* before the section is rendered we can restrict the maximum depth value to a portion of the entire range. To render the far field view volume the depth range to lie between the absolute maximum and the maximum of the portion we just rendered. As the depth values for the first section were reversed, we set the second depth range to be reversed (with the far value smaller than the near value) and continue using our backwards z-buffer. To combine the two volumes:

- Set the depth range to be [0.0, portion], where portion is the fraction of the entire view volume before the point of convergence.
- Render the near field volume.
- Set the depth range to [1.0, portion].
- Render the far field volume, still using the backwards z-buffer.

The depth range needs to be switched because we are combining two views from opposite directions. It is equally valid to render the near field volume with a reversed depth range and normal z-buffer and then the far field volume with a normal depth range and z-buffer. Computationally the methods are the same, though we initially implemented the near field volume with a reversed z-buffer.

A perspective projection of a cube is shown in Figure 3, which is then rendered in anti-perspective view in Figure 4. The bottom of the cube, which is not visible in Figure 3, can clearly be seen in Figure 4. In Figure 4 it almost appears that we can see inside the cube, but it is the outer faces that are showing. The two back faces are still obscured, to display them a projection must totally surround the cube.

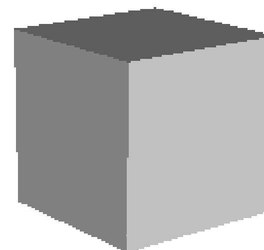


Figure 3: A perspective Projection of a Cube

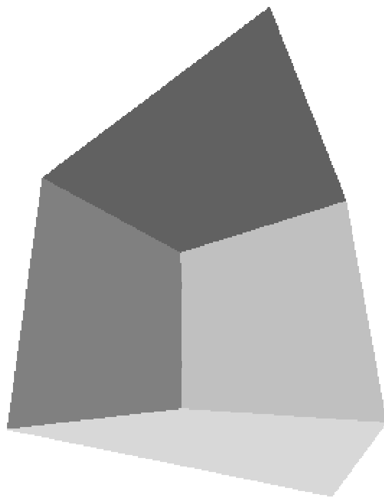


Figure 4: An Anti-Perspective Projection of a Cube

4.4 Spherical Concave Mirror Reflections

A spherical concave mirror behaves exactly as an anti-perspective projection. Any set of rays emanating from a single point that reflect off the mirror will pass through a point of convergence. An anti-perspective view positioned at the position of the concave mirror that has the same point of convergence will capture the same information if there are no inter-reflections. Therefore the anti-perspective view can be used to texture the concave mirror object accurately portraying the reflection from that viewpoint on the mirror. This is shown in Figure 5.

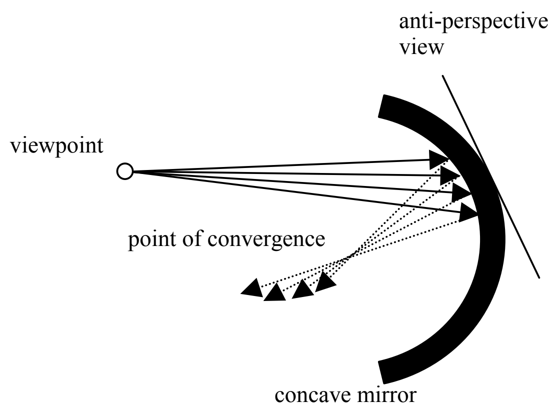


Figure 5: Reflection Rays on a Spherically Concave Mirror

If the point of convergence is surrounded by the concave mirror a single anti-perspective view is not sufficient. Multiple anti-perspective views can be combined in this case. Combining these views can be done similar to the way in which spherical environment maps can be generated from cube maps and is detailed in the following section.

5 Virtual Spherical Camera

Conceptually a spherical camera captures all the light coming into a point. This generates something like an extended fish-eye lens view where the front, back, top, bottom and sides can all be seen. Typically such images are used as the basis of environment mapping. Environment mapping allows for an approximation of complex reflections to be made, by simplifying reflections such that they all effectively bounce off the centre of the sphere represented by the environment map. The idea was first published by Blinn and Newell [1976].

The efficient generation of these environment maps is a significant concern because in a dynamic scene the environment map must be updated every frame. The environment map is typically one of two different types: a sphere map or a cube map. A sphere map has the middle as the front of the view, surrounded by the sides, top and bottom. The front forms a ring around the edge. A cube map presents the front, back, sides, top and bottom as separate segment of the final image like a dissected cube. Figures 6 and 7 show a sphere map and cube map respectively.

James Blinn's original environment mapping work used a different type of map, sometimes called a longitude/latitude map. This is because the x and y axes of the map correspond to longitude and latitude, similar to the way a Plate Carree map projection is similarly laid out. Figure 8 shows an environment map from Blinn's original paper and is available from Debevec [2002]. Each different shaded section represents a wall, ceiling or floor of a simple cubic room.

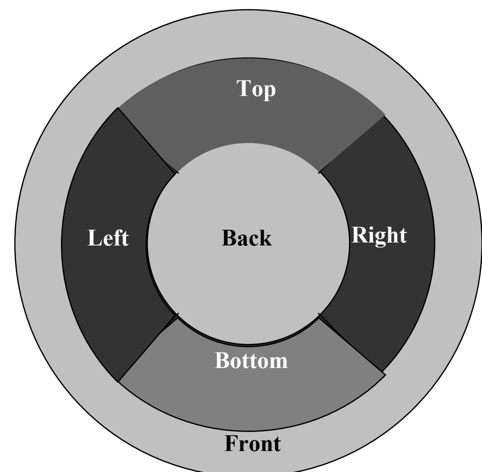


Figure 6: A Sphere Map

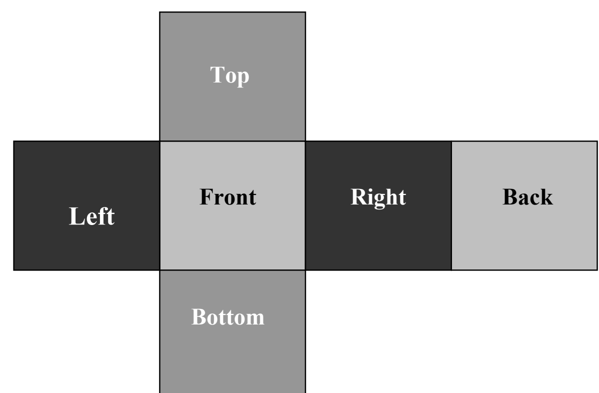


Figure 7: A Cube Map

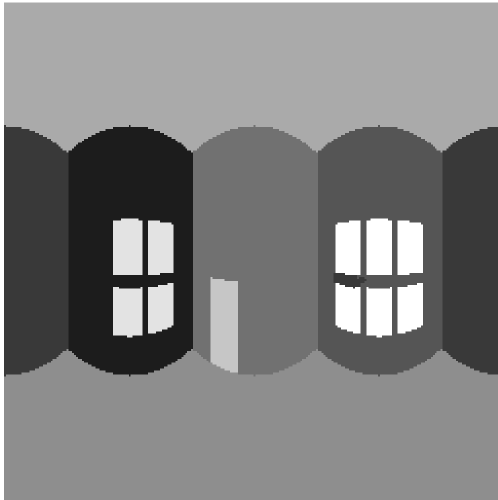


Figure 8: A Longitude/Latitude Map From Blinn [1976]

A cube map can be created by rendering a view for each of the faces. The sphere map is more difficult to render but can be created by warping a cube map to the appropriate shape as detailed by Ned Greene [Greene 1986]. A 2D mesh warp can transform the cube map’s faces, once rendered, to that of the sphere map.

A cube has six faces so six views must be rendered to form one. A simpler solid shape, the tetrahedron, has four faces and it can also be used to generate environment maps. Fortes [2000] presented a tetrahedral environment map that can be warped to a sphere or other map. The advantage of the tetrahedral map is that only four views need be rendered.

6 Inwards Spherical Projection

An inwards spherical camera is like a normal spherical camera except that the rays are directed from the surface of the sphere to the centre, rather than the other way, thus generating a view of every side of an object. In the same way that warping several normal perspective views can generate a spherical view, warping anti-perspective views can generate an inverse spherical view.

Our implementation uses a tetrahedron as a base for the spherical projection to reduce the number of projections needed. For each face of the tetrahedron we render an anti-perspective view. The anti-perspective views converge on a point in the centre of the tetrahedron. The extent of the anti-perspective viewing volumes is the point of convergence, so only the near field volume need be rendered for each one. For visualisation purposes it was unnecessary for the view to continue past the point of convergence, though this could easily be done.

Once the four views are rendered they can be warped to another environment map style. For this we chose implementation a longitude/latitude map. In environment mapping the map is an intermediate data structure, whereas in our visualisation work the map is displayed. To generate the map, the views are copied to textures and then used to texture a mesh that warps the views to the shape of the longitude/latitude map.

The series of operations is as follows:

- For each face of the tetrahedron render an anti-perspective view from the face converging on the centre of the tetrahedron, to a portion of the screen
- Copy the screen to the texture buffer
- Draw a 2D warped mesh (in the shape of the desired map) using the textures previously rendered

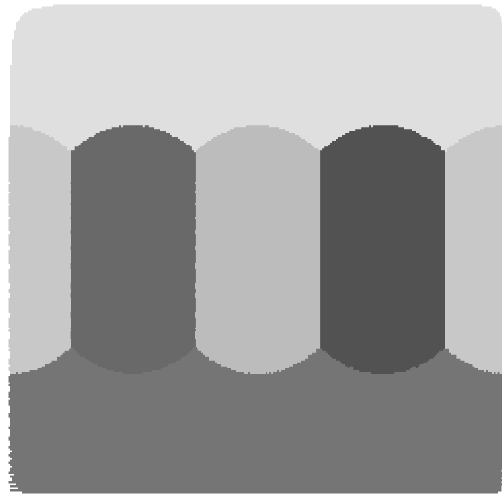


Figure 9: An Inward Looking Spherical View of a Cube

Figure 9 shows a cube rendered with an inward spherical projection. It very closely resembles the environment map in Figure 8, but instead of being inside a cubic room the virtual camera surrounds the cube. All sides of the cube are visible in a single continuous image.

7 Conclusions and Further Work

This paper presents a technique for rendering anti-perspective projections. It can be used both for implementing reflections on spherically concave mirrors and as a visualisation tool. By building on research into environment maps, the projection can be extended into an inward looking spherical camera. This can render a view surrounding an object, showing all sides of an object at once. The performance of the technique is acceptable for interactive use as it is comparable to work in interactive generation of standard environment maps.

We are currently investigating further aspects of the visualisation and other uses of the techniques. For example the inward spherical projection may provide an interesting basis for image resynthesis as it captures a lot of the information for an object in a single image. A cylindrical projection is being investigated to compliment the spherical projection.

References

T. W. Birch. *Maps topographical and statiscal.* Oxford Press, 1964.

J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542--547, Oct. 1976.

P. Debevec, http://www.debevec.org/ReflectionMapping/Blinn/, Accessed on September 2002.

J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer graphics: principles and practice*, 2nd edition. Addison-Wesley, 1990.

T. Fortes. *Tetrahedron Environment Maps*, Master's Thesis, Department of Computing Science, Chalmers University of Technology, Gothenburg, Sweden, 2000.

N. Greene, Environment Mapping and Other Applications of World Projections, *IEEE Computer Graphics and Applications,* November 1986, pp. 2130.

Bibliography

[BN76] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

[Bus03] Samuel Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, 2003.

[CA00] Min Chen and James Arvo. Perturbation methods for interactive specular reflections. *IEEE Transactions on Visualization and Computer Graphics*, 6(3):253–264, 2000.

[CCF97] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications: Special Issue on Information Visualization*, 17(4):42–51, / 1997.

[CHH02] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 37–46. Eurographics Association, 2002.

[Cho00] F. Cho. *Towards Interative Ray Tracing in Two- and Three-Dimensions*. PhD thesis, University of California at Berkeley, 2000.

[Chu01] Siu-Hang Chu. Animating chinese landscape paintings and panoramas. Master's thesis, Hong Kong University of Science and Technology, 2001.

[CT01] S. H. Chu and C. L. Tai. Animating Chinese Landscape Paintings and Panorama using Multi-Perspective Modeling. In *Proc. of Computer Graphics International*. IEEE Press, 2001.

[Dav01] A. Davidhazy. Peripheral portraits and other strip camera photographs. Retrieved October, 2001 from the World Wide Web `http://www.rit.edu/~andpph/exhibit-6.html`, 2001.

[Die96] Paul Diefenbach. *Pipeline Rendering: Interaction and Realism Through Hardware-Based Multi-Pass Rendering.* PhD thesis, University of Pennsylvania, 1996.

[Dur02] Fredo Durand. An invitation to discuss computer depiction. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 111–124. ACM Press, 2002.

[Esc92] M. C. Escher. *The Graphic Work.* Evergreen, 1992.

[FDFH90] J. D. Foley, A. Van Damme, S. K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1990.

[GGSC96] S. Gortler, R. Grzescuzk, R. Szeliski, and M. Cohen. The lumigraph. In *Proc. of SIGGRAPH*, 1996.

[Gla99] Georg Glaeser. Reflections on spheres and cylinders of revolution. *Journal for Geometry and Graphics*, 3(2):121–139, 1999.

[Gla00] Andrew Glassner. Cubism and cameras: Free-form optics for computer graphics. Technical report, Microsoft Research, 2000.

[Hai87] Eric Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11), 1987.

[HBS+99] M. K. Hurdal, P. L. Bowers, K. Stephenson, D. W. L. Sumners, K. Rehm, D. Schaper, and A. Rottenberg. Quasi-conformally flat mapping the human cerebellum. In C. Taylor and A. Colchester, editors, *Medical Image Computing and Computer-Assisted Intervention*, volume 1679 of *Lecture Notes in Computer Science*, pages 279–286. Springer, 1999.

[HH84] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *Proceedings of the 11th annual conference on computer graphics and interactive techniques*, pages 119–127, 1984.

[HLCS99] Wolfgang Heidrich, Hendrik Lensch, Michael F. Cohen, and Hans-Peter Seidel. Light field techniques for reflections and refractions. In *Rendering Techniques '99: Proceedings of the 10th Eurographics Workshop on Rendering (EGRW-99)*, 1999.

[HPP00] Vlastimil Havran, Jan Prikryl, and Werner Purgathofer. Statistical comparison of ray-shooting efficiency schemes. Technical Report TR-168-2-00-14, Vienna University of Technology, 2000.

[HSL01] Z. Hakura, J. Snyder, and J. Lengyel. Parameterized environment maps. In *Proc. of the 2001 Symposium of Interactive 3D Graphics*, 2001.

[Jan93] F. W. Jansen. Realism in real-time? In *Proc. of the Fourth Eurographics Workshop on Rendering*, 1993.

[Kea98] T. A. Keahey. The generalized detail-in-context problem. In *Proc. of the IEEE Symposium on Information Visualization*, 1998.

[LA94] Y. K. Leung and M. D. Aerley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, 1994.

[LG96] Helwig Löffelmann and E. Gröller. Ray tracing with extended cameras. *Journal of Visualization and Computer Animation*, 7:211–228, 1996.

[Löf95] Helwig Löffelmann. Extended cameras for ray tracing. Master's thesis, Vienna Technical Institute, 1995.

[OR99] Eyal Ofek and Ari Rappoport. Interactive reflections on curved objects. In *Proc of SIGGRAPH 99*, 1999.

[oVPL04] Persistence of Vision Pty. Ltd. (2004). *Persistence of Vision (TM) Raytracer.* Persistence of Vision Pty. Ltd., Williamstown, Victoria, Australia. http://www.povray.org/, 2004.

[PBMH02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002.

[RB98] P. Rademacher and G. Bishop. Multiple-center-of-projection images. In *Proc. of SIGGRAPH 98*, 1998.

[RGL04] Augusto Roman, Gaurav Garg, and Marc Levoy. Interactive design of multi-perspective images for visualizing urban landscapes. In *Proceedings of IEEE Visualization 2004*, 2004.

[SA98] M. Segal and K. Akeley. *The OpenGL Graphics System: a Specification (Version 1.2)*, 1998.

[Sin02] Karan Singh. A Fresh Perspective. In *Proc. Graphics Interface*, pages 17–24, May 2002.

[Smi97] R. Smith. *Distortion Oriented Displays for Demanding Applications.* PhD thesis, Gippsland School of Computing and Information Technology, Monash University, 1997.

[VC01a] Scott Vallance and Paul Calder. Context in 3d planar navigation. In *Proc. of the Australian User Interface Conference AUIC 2001*, 2001.

[VC01b] Scott Vallance and Paul Calder. Multi-perspective images for visualisation. In *Proceedings of the Visual Information Processing Workshop 2001*, 2001.

[VC03] Scott Vallance and Paul Calder. Inward looking projections. In *Proceedings of GRAPHITE 2003*, 2003.

[WCS00] Donovan Winch, Paul Calder, and Ray Smith. Focus + context^3: Distortion-orientated displays in three dimensions. In *Proc. of the Australasian User Interface Conference AUIC 200*, 2000.

[WCS01] Donovan Winch, Paul Calder, and Raymond Smith. Solving the occlusion problem for three-dimensional distortion-oriented displays. In *Proceedings of the 2nd Australasian conference on User interface*, pages 108–115. IEEE Computer Society, 2001.

[WFH$^+$97] D. Wood, A. FinkelStein, J. Hughes, C. Thayer, and D. Salesin. Multiperspective panoramas for cel animation. In *Proc of SIGGRAPH 97*, 1997.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[Wil97] J. Willats. *Art and Representation.* Princeton University Press, 1997.

[YM04] Jingyi Yu and Leonard McMillan. General linear cameras. In Tomás Pajdla and Jiri Matas, editors, *Computer Vision - ECCV 2004, 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part II*, volume 3022 of *Lecture Notes in Computer Science*. Springer, 2004.