2007-10-27

Intersection Algorithms Based On Geometric Intervals

Nicholas Stewart North
Brigham Young University - Provo

INTERSECTION ALGORITHMS BASED ON GEOMETRIC INTERVALS

by

Nicholas S. North

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Nicholas S. North

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Thomas W. Sederberg, Chair
Date	Dan Ventura
Date	Kevin D. Seppi

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Nicholas S. North in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Thomas W. Sederberg Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

INTERSECTION ALGORITHMS BASED ON GEOMETRIC INTERVALS

Nicholas S. North
Department of Computer Science
Master of Science

This thesis introduces new algorithms for solving curve/curve and ray/surface intersections. These algorithms introduce the concept of a *geometric interval* to extend the technique of Bézier clipping. A geometric interval is used to tightly bound a curve or surface or to contain a point on a curve or surface. Our algorithms retain the desirable characteristics of the Bézier clipping technique such as ease of implementation and the guarantee that all intersections over a given interval will be found. However, these new algorithms generally exhibit cubic convergence, improving on the observed quadratic convergence rate of Bézier clipping. This is achieved without significantly increasing computational complexity at each iteration. Timing tests show that the geometric interval algorithm is generally about 40-60% faster than Bézier clipping for curve/curve intersections. Ray tracing tests suggest that the geometric interval method is faster than the Bézier clipping technique by at least 25% when finding ray/surface intersections.

ACKNOWLEDGMENTS

This work would not have been possible without the advice and support of my thesis advisor, Dr. Thomas W. Sederberg. His constant encouragement, insight and advice were instrumental in directing this research to its final goal. I would like to thank Dr. Sederberg and the other members of my thesis committee, Dr. Dan Ventura and Dr. Kevin D. Seppi, for their advice and assistance as I completed this document.

I would also like to thank my wife Tamsin for all of the patience, encouragement, love and support that she has shown me at all times. She has never expressed any doubt in my ability to complete any aspect of this project. Her understanding and companionship has contributed as much to this work as any insight that I may have had on my own.

Finally, I would like to acknowledge the divine inspiration and guidance of our Heavenly Father and express gratitude for the influence that it has been on everyone involved in this work.

Contents

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 Free-form Curves and Surfaces 1

1.2 Intersections 2

1.3 Interval Arithmetic 3

1.4 Geometric Intervals 4

1.5 Fat Lines 5

1.6 Solving for Intersections 6

1.7 Overview 9

2 Background 11

2.1 Interval Arithmetic 11

2.1.1 Vectors 13

2.1.2 Polynomials 14

2.1.3 Overestimation 15

2.2 Bézier Curves 15

2.2.1 The de Casteljau Algorithm 18

2.2.2 Rational Curves 20

2.2.3 Derivatives 22

2.3 Bézier Surface Patches 23

2.3.1 The de Casteljau Algorithm 24
2.3.2 Rational Surfaces 26
2.3.3 Derivatives 27
2.4 Interval Bézier Curves 27
2.5 Hybrid Curves and Surfaces 28

3 Curve/curve Intersection **33**
3.1 Previous Work 33
3.2 The GeoClip Curve/curve Intersection Algorithm 36
3.2.1 Geometric Intervals 37
3.2.2 Fat Lines 39
3.2.3 Geometric Interval Clipping 40
3.2.4 Iterating 44
3.2.5 Multiple Intersections 47
3.3 Timing Comparisons 48
3.4 Observations and Conclusions 51

4 Ray/surface Intersection **53**
4.1 Previous Work 53
4.2 The GeoClip Ray/surface Intersection Algorithm 55
4.2.1 Projection to $\mathbb{R}^2$ 55
4.2.2 Geometric Intervals 57
4.2.3 Geometric Interval Clipping 59
4.2.4 Iterating 63
4.2.5 Multiple Intersections 66
4.3 Timing Comparisons 66
4.4 Observations and Conclusions 72

5 Conclusions and Future Directions **75**

5.1 Summary of Results 75

5.2 Future Directions 76

Bibliography **77**

List of Figures

1.1 Objects created using free-form curves. 1

1.2 Single and multiple intersection of free-form curves. 2

1.3 A Bézier curve and an interval vector, $[\mathbf{P}']$, bounding the derivative. . 3

1.4 Comparison of $[\mathbf{P}]$ and $[\mathbf{P}(t)]$ with the geometric intervals $[\mathbf{P}]_g$ and $[\mathbf{P}(t)]_g$. 5

1.5 A fat line is a pair of parallel lines enclosing a curve. 6

1.6 Clipping nonintersecting regions using geometric intervals and fat lines. 7

2.1 Interval vectors in $\mathbb{IR}^2$ are axis-aligned rectangular areas. 14

2.2 A cubic Bézier curve and its control polygon. 16

2.3 Linear, quadratic and cubic Bernstein polynomials $B_i^n(t)$ for $t \in [0, 1]$. 17

2.4 A Bézier curve is completely contained within its *convex hull*. 17

2.5 The de Casteljau algorithm repeatedly interpolates control points. . . 19

2.6 Curve subdivision using intermediate points from Figure 2.5. 19

2.7 A rational Bézier curve with weights $w_i = 1$ except w_1. 20

2.8 Rational Bézier curves make it possible to represent conic sections. . . 21

2.9 Hodographs define the derivative at each point along the curve. . . . 22

2.10 A degree 3×2 polynomial Bézier patch. 23

2.11 Evaluation of $\mathbf{P}(.5, .5)$ using the de Casteljau algorithm. 25

2.12 Subdivision of a Bézier patch using the de Casteljau algorithm. . . . 25

2.13 Rational Bézier patches arranged to form a cylinder. 26

2.14 A quadratic interval Bernstein polynomial. 28

2.15 A cubic Bézier curve expressed as a hybrid quadratic curve. 30

3.1 A fat line $\mathbf{L_P}$ which bounds a quartic curve $\mathbf{P}(t)$ by enclosing all $\mathbf{P}_i$. . 35
3.2 $[\mathbf{P}]_g$ represented as the surface $\mathbf{Q}(s,t)$ and hybrid curve $\hat{\mathbf{P}}(t)$. 39
3.3 Intersection of the Geometric interval $[\mathbf{P}]_g$ and the fat line $\mathbf{L_Q}$. 40
3.4 $\mathbf{P}(t) \neq \mathbf{Q}(u)$ when $[\hat{\delta}](t) < \delta_{\min}$ or $[\hat{\delta}](t) > \delta_{\max}$. 42
3.5 Intersection of the Geometric interval $[\mathbf{Q}]_g$ and the fat line $\mathbf{L}_{\mathbf{P}^{(1)}}$. . . . 45
3.6 $\mathbf{Q}(u) \neq \mathbf{P}(t)$ when $[\hat{\delta}](u) < \delta_{\min}$ or $[\hat{\delta}](u) > \delta_{\max}$. 45
3.7 Two intersections between curves $\mathbf{P}$ and $\mathbf{Q}$. 47
3.8 Each intersection is isolated after splitting curve $\mathbf{P}$ in half. 47

4.1 Bézier patch $\mathbf{P}$, the projection of $\tilde{\mathbf{P}}$. 57
4.2 Application of Theorems 4.1 and 4.2 to a projected surface patch $\mathbf{P}(s,t)$. 58
4.3 In this example, $[\hat{\delta}_0^s] = [\hat{\delta}_{0,0}^s, \hat{\delta}_{0,2}^s]$, $[\hat{\delta}_1^s] = [\hat{\delta}_{1,1,0}^s, \hat{\delta}_{1,3,0}^s]$ and $[\hat{\delta}_2^s] = [\hat{\delta}_{2,3}^s, \hat{\delta}_{2,1}^s]$. 61
4.4 $[\hat{\delta}^s](s)$ crosses the s axis at $s \in [0.26647, 0.40186]$. 62
4.5 $\mathbf{P}(s,t)$ is clipped to $s \in [0.26647, 0.40186]$ during the first clipping step. 63
4.6 The t clipping values are found by solving for the roots of $[\hat{\delta}^t](t)$. . . . 64
4.7 Convergence to $\mathbf{0}$. 65
4.8 Utah teapot rendering. 68
4.9 Time taken to calculate primary ray intersections with visible patches. 69
4.10 A patch from the teapot's knob, oriented to have multiple intersections. 70
4.11 Time taken to calculate primary ray intersections with the knob patch. 70
4.12 Time taken to calculate primary ray intersections with a body patch. 71

List of Tables

1.1 Convergence of intersection intervals for curves in Figure 1.6. 8

3.1 Convergence of intersection intervals for curves in Figure 3.3. 46

3.2 Relative computation times for polynomial Bézier curve intersection. 49

3.3 Relative computation times for rational Bézier curve intersection. . . 49

3.4 Relative computation time for cubic Bézier curves, by data set. 50

3.5 Relative computation time for cubic Bézier curves, by intersection count. 51

4.1 Relative computation times for randomly generated bicubic patches. . 67

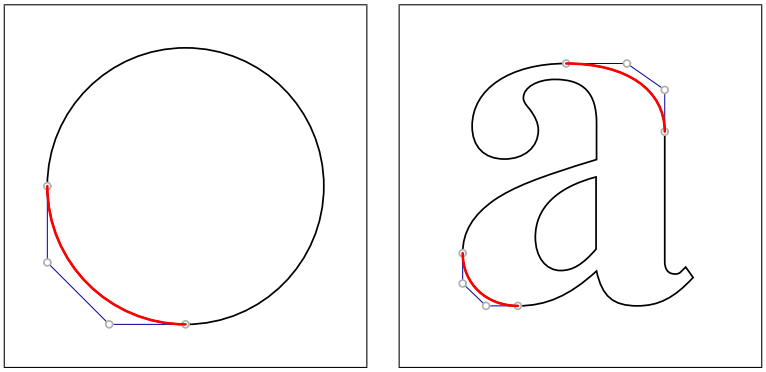
4.2 Relative ray/surface intersection computation times. 72

Chapter 1

Introduction

1.1 Free-form Curves and Surfaces

When an artist creates an illustration or, perhaps, designs a scalable font using a computer, it is highly likely that the software the artist uses will employ *free-form curves*. Similarly, *free-form surfaces* enable artists and engineers to produce three-dimensional objects accurately and efficiently using the computer. Free-form curves and surfaces have precise mathematical representations and yet they are generally easy for an artist or engineer to manipulate. The objects in Figure 1.1 were created using a popular type of free-form curve known as a Bézier curve.



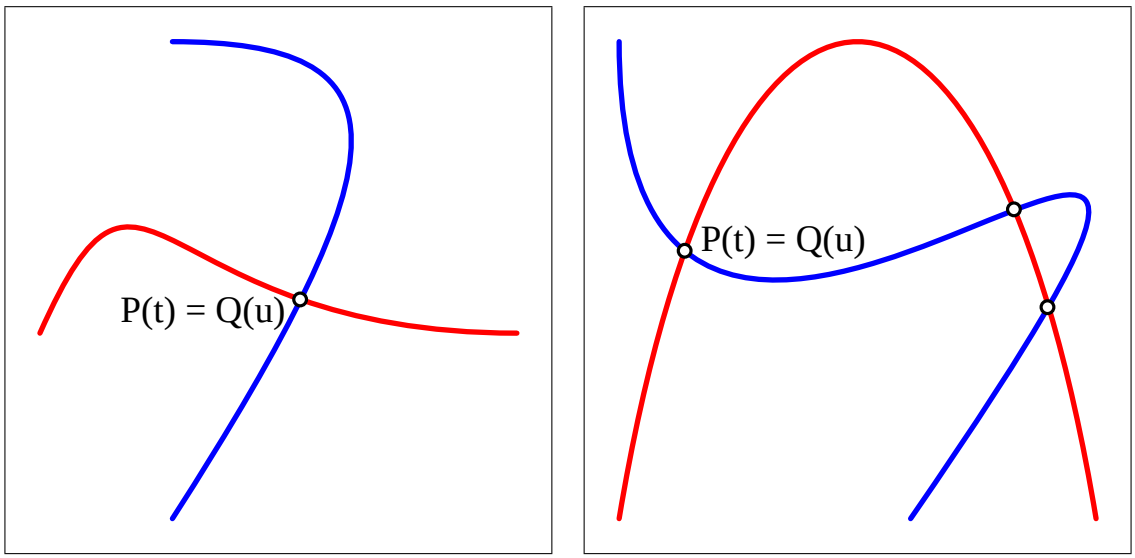
(a) A simple circle. (b) Font glyph for the letter *a*.

Figure 1.1: Objects created using free-form curves.

Many algorithms have been developed to make manipulating free-form curves and surfaces more practical and efficient. This paper introduces the concept of a

geometric interval and algorithms which exploit the properties of geometric intervals for finding curve/curve and ray/surface intersections efficiently.

1.2 Intersections



(a) Simple intersection. (b) Multiple intersections.

Figure 1.2: Single and multiple intersection of free-form curves.

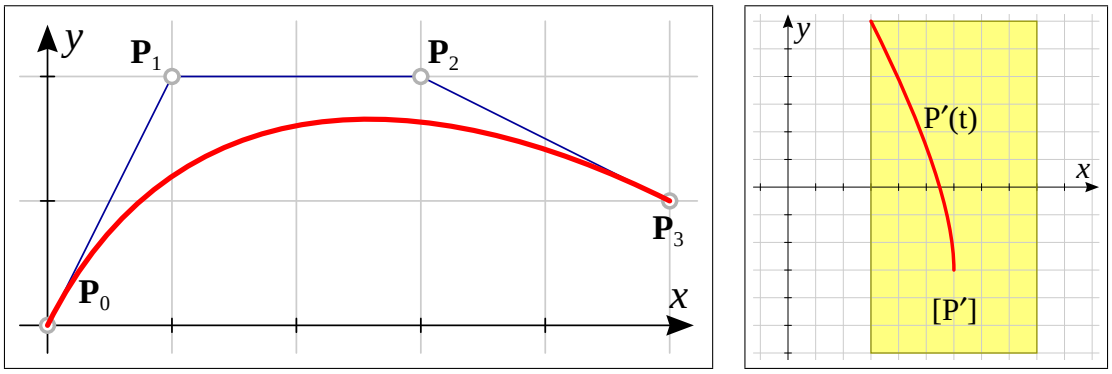
The intersection of two curves is the set of points where the curves meet, as illustrated in Figure 1.2. Two curves may have no points of intersection, a single common point (Figure 1.2a), or multiple common points (Figure 1.2b). Determining if two curves intersect and identifying all points of intersection is a fundamental requirement of many algorithms that utilize free-form curves.

Ray/surface intersection is a fundamental operation used in *ray tracing*. Ray tracing models the interaction of light with objects in an artificial scene. Modeling light allows the computer to create images of the scene for entertainment, product design or other purposes. Calculating ray/surface intersections quickly is key to making this process efficient.

1.3 Interval Arithmetic

The intersection algorithms presented in this paper are based on improvements to interval arithmetic techniques. Interval analysis is a mature field developed as a method of bounding uncertainty and computational error in calculations. Robust means of isolating roots in algebraic functions (for example [9, 17]) and of finding intersections (see [30]) have been developed using interval methods. The fundamentals of interval arithmetic are more fully explored in Section 2.1.

An interval is a simply a range of numbers defined by an upper and lower bound. For example, interval $[a, b]$ denotes the set $\{x \mid a \leq x \leq b\}$. We also use the notation $[x]$ to indicate that the quantity x is an interval. Intervals allow for calculations to be performed when the precise value of a variable is uncertain, but a bound on the possible values can be obtained. The rules governing such calculations define an interval arithmetic.



(a) **P**, a simple cubic Bézier curve. (b) Bound on the derivative.

Figure 1.3: A Bézier curve and an interval vector, $[\mathbf{P}']$, bounding the derivative.

The chief disadvantage of using standard intervals is the tendency towards overestimation. For example, consider the Bézier curve in Figure 1.3a, given by

$$\begin{aligned}\mathbf{P}(t) &= \big(x(t), y(t)\big) \\ &= (\mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0)t^3 + 3(\mathbf{P}_2 - 2\mathbf{P}_1 + \mathbf{P}_0)t^2 + 3(\mathbf{P}_1 - \mathbf{P}_0)t + \mathbf{P}_0,\end{aligned} \tag{1.1}$$

where $\mathbf{P}_0 = (0,0)$, $\mathbf{P}_1 = (1,2)$, $\mathbf{P}_2 = (3,2)$ and $\mathbf{P}_3 = (5,1)$. Differentiating, we have

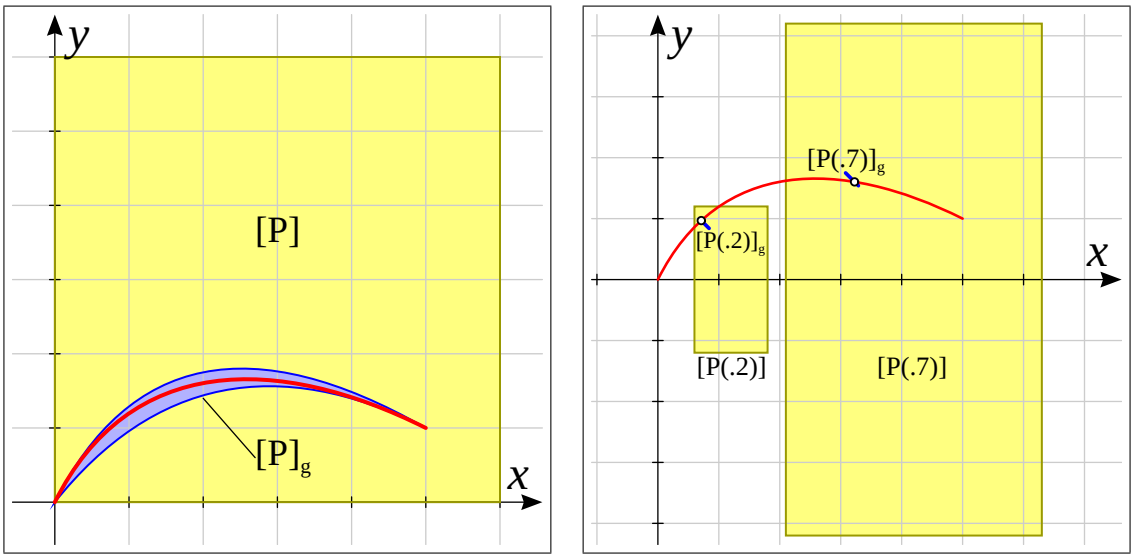
$$\begin{aligned}\mathbf{P}'(t) &= \big(x'(t), y'(t)\big) \\ &= 3(\mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0)t^2 + 6(\mathbf{P}_2 - 2\mathbf{P}_1 + \mathbf{P}_0)t + 3(\mathbf{P}_1 - \mathbf{P}_0).\end{aligned} \tag{1.2}$$

We can use interval arithmetic to determine bounds on the x and y components of this derivative function for all values of $[t] = [0,1]$. Doing this gives us the solution $x' \in [3,9]$ and $y' \in [-6,6]$. $[\mathbf{P}'] = ([x'],[y'])$ is the axis-aligned rectangle shown in Figure 1.3b. While it is clear that $\mathbf{P}'(t) \in [\mathbf{P}']$, as intended, $[\mathbf{P}']$ is a rather large bound of $\mathbf{P}'(t)$.

1.4 Geometric Intervals

Interval equations are easy to evaluate, but the resulting axis-aligned rectangular regions are generally loose bounds. Much tighter bounds can be found, but there is a tradeoff between evaluation cost and bounding efficiency. To provide a bound that very tight, but computationally much simpler than working directly with the curve or surface, we introduce the concept of a *geometric interval*. A geometric interval is a two- or three-dimensional region that is relatively easy to evaluate, but provides a tighter bound than axis-aligned rectangles.

For example, consider the Bézier curve in Figure 1.3a again. Using standard interval arithmetic, we can find a bound for this curve simply by evaluating $\mathbf{P}(t)$ over the interval $[t] = [0,1]$. For our example curve, this yields the interval vector $[\mathbf{P}] = ([0,0],[6,6])$, the rectangular region in Figure 1.4a. For comparison, the darker curved region in Figure 1.4a depicts one possible geometric interval bound for the same curve, designated $[\mathbf{P}]_g$. A complete discussion of this geometric interval is found in Section 3.2.



(a) The interval $[\mathbf{P}]$ (rectangle) bounds the curve $\mathbf{P}$ loosely. In contrast, the geometric interval $[\mathbf{P}]_g$ (curved area) is a much tighter bound for $\mathbf{P}$.

(b) Geometric intervals $[\mathbf{P}(.2)]_g$ and $[\mathbf{P}(.7)]_g$ (very small curves) are much tighter than the interval bounds $[\mathbf{P}(.2)]$ and $[\mathbf{P}(.7)]$ (rectangles).

Figure 1.4: Comparison of $[\mathbf{P}]$ and $[\mathbf{P}(t)]$ with the geometric intervals $[\mathbf{P}]_g$ and $[\mathbf{P}(t)]_g$.

One way that interval arithmetic can be used to compute the intersection of two curves is to invoke the first-order Taylor expansion $\mathbf{P}(t) \in \mathbf{P}_0 + [\mathbf{P}']t$. We may evaluate this expression more easily than evaluating the curve directly. The result is an interval vector which contains the point $\mathbf{P}(t)$, as illustrated by the rectangular regions in Figure 1.4b. Intersecting the Taylor expansions of two curves is a simple way to identify parameter intervals where the curves may overlap.

Geometric intervals may also be used to bound any point $\mathbf{P}(t)$ along the curve. A particularly tight geometric interval for this application is $[\mathbf{P}(t)]_g$, a curve that lies along $[\mathbf{P}]_g$ at t. Figure 1.4b illustrates the small curves $[\mathbf{P}(0.2)]_g$ and $[\mathbf{P}(0.7)]_g$, which contain the points $\mathbf{P}(0.2)$ and $\mathbf{P}(0.7)$, respectively.

1.5 Fat Lines

To solve for curve/curve intersections we also use another curve bounding technique: *fat lines*. A fat line, introduced in [27], is simply a pair of parallel lines that lie on

either side of the curve, enclosing the curve within. Geometric intervals are used to determine parameter ranges where a curve overlaps a fat line, thereby identifying regions that potentially contain intersections.

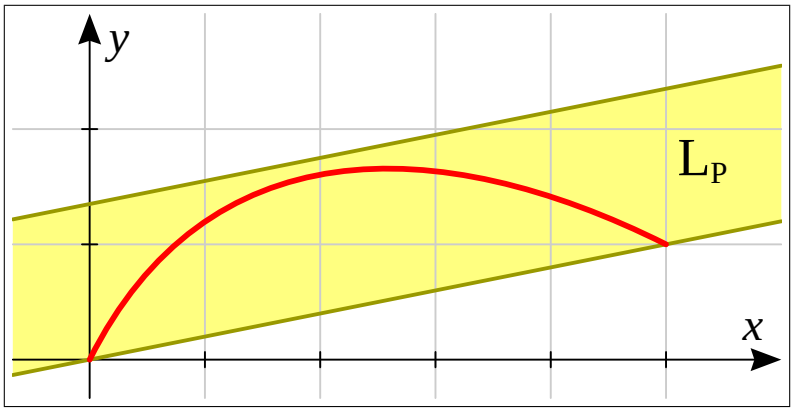


Figure 1.5: A fat line is a pair of parallel lines enclosing a curve.

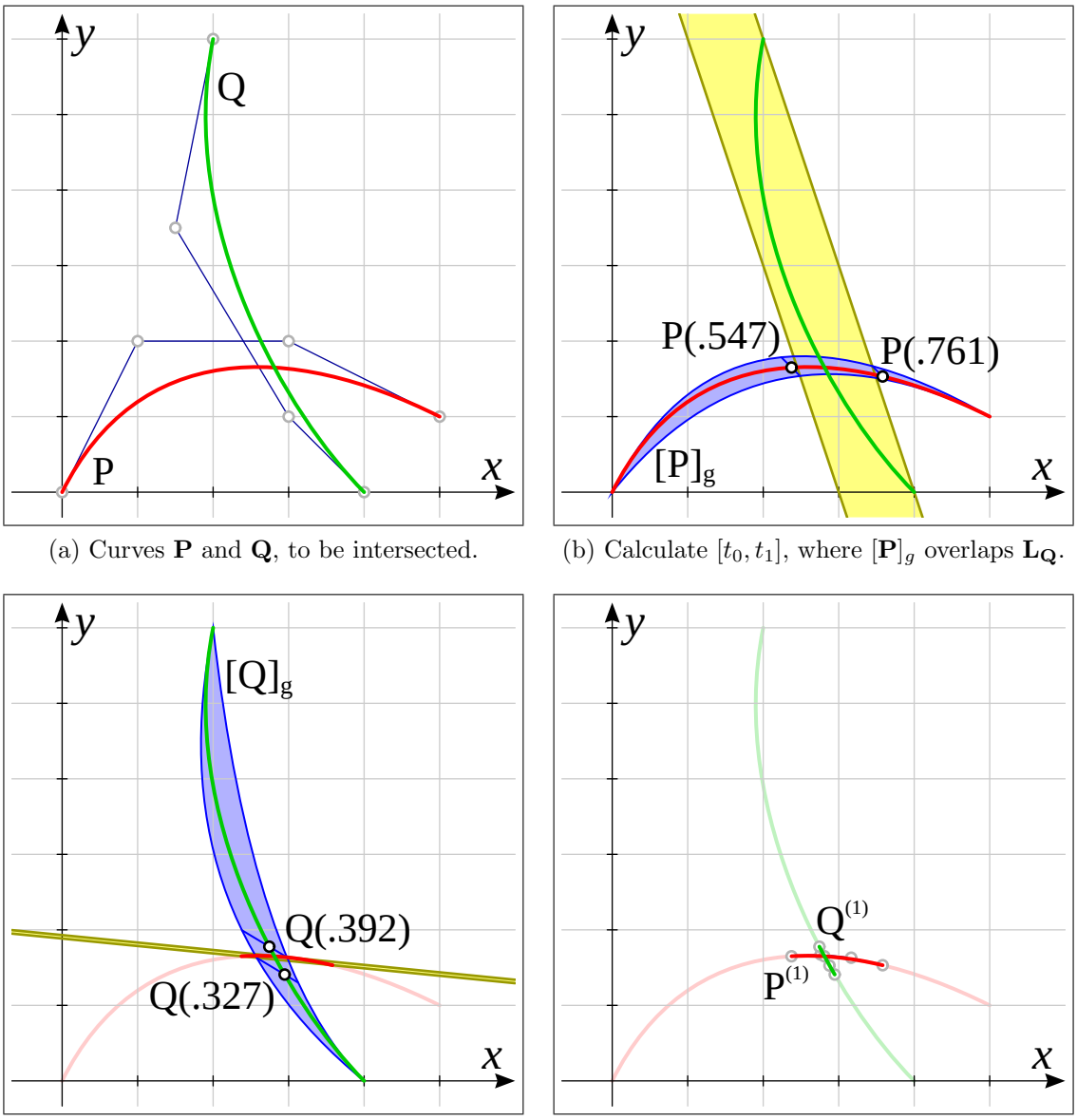
Figure 1.5 shows a fat line, designated $\mathbf{L_P}$, which bounds the curve $\mathbf{P}$. Although any pair of parallel lines enclosing a curve is a fat line, [27] outlines a simple scheme for finding tight fat lines for Bézier curves. This method requires nothing more than examining the control points of the curve.

1.6 Solving for Intersections

We now present a brief overview of the curve/curve intersection process to illustrate how geometric intervals and fat lines are used to isolate intersections. The intersection of the curves $\mathbf{P}(t)$ and $\mathbf{Q}(u)$, shown in Figure 1.6a, will be used as an example to drive the discussion of the algorithm. In this example, the two curves have a single intersection $\mathbf{P}(t^*) = \mathbf{Q}(u^*)$.

First, a fat line $\mathbf{L_Q}$ for the curve $\mathbf{Q}$ is found. The geometric interval for curve $\mathbf{P}$ is then used to determine the parameter interval $[t_0, t_1]$ where $[\mathbf{P}]_g$ overlaps $\mathbf{L_Q}$:

$$[t_0, t_1] \supseteq \{t \mid [\mathbf{P}(t)]_g \cap \mathbf{L_Q} \neq \emptyset\}. \tag{1.3}$$



(a) Curves $\mathbf{P}$ and $\mathbf{Q}$, to be intersected.

(b) Calculate $[t_0, t_1]$, where $[\mathbf{P}]_g$ overlaps $\mathbf{L}_\mathbf{Q}$.

(c) Calculate $[u_0, u_1]$, where $[\mathbf{Q}]_g$ overlaps $\mathbf{L}_{\mathbf{P}^{(1)}}$.

(d) New curves, $\mathbf{P}^{(1)}$ and $\mathbf{Q}^{(1)}$, after clipping.

Figure 1.6: Clipping nonintersecting regions using geometric intervals and fat lines.

This process is illustrated in Figure 1.6b. Since $\mathbf{P}(t) \neq \mathbf{Q}(u)$ for $t \notin [t_0, t_1]$ and $u \in [0, 1]$, we may clip away portions of curve $\mathbf{P}$ for which $t \notin [t_0, t_1]$, retaining the segment $\mathbf{P}^{(1)}$ where an intersection may still occur. Calculating $[t_0, t_1]$ is the heart of the algorithm and is explained in detail in Chapter 3.

Next, the process is reversed. We find the fat line $\mathbf{L}_{\mathbf{P}^{(1)}}$ for the curve segment $\mathbf{P}^{(1)}$. Then we calculate the parameter range $[u_0, u_1]$ corresponding to the overlap

of $[\mathbf{Q}]_g$ and $\mathbf{L}_{\mathbf{P}^{(1)}}$, as shown in Figure 1.6c. We then extract the subcurve $\mathbf{Q}^{(1)}$, the segment of $\mathbf{Q}$ spanning $[u_0, u_1]$, where an intersection may occur. The subcurves $\mathbf{P}^{(1)}$ and $\mathbf{Q}^{(1)}$ are shown in Figure 1.6d.

These steps assure that any intersection $\mathbf{P}(t) = \mathbf{Q}(u)$ must take place along the curve segments covered by the subcurves $\mathbf{P}^{(1)}$ and $\mathbf{Q}^{(1)}$. Note that, in subsequent iterations, $\mathbf{P}^{(i+1)}$ and $\mathbf{Q}^{(i+1)}$ can be approximated more accurately with a geometric interval or a fat line than $\mathbf{P}^{(i)}$ and $\mathbf{Q}^{(i)}$. Due to a rapid improvement in approximation accuracy, our intersection algorithm generally exhibits cubic convergence.

i	$\mathbf{P}$ *Intersection Interval* $[t_0^i, t_1^i]$	*Width of* $[t_0^i, t_1^i]$
0	$[0, 1]$	1.0000000×10^{0}
1	$[0.547013139706567, 0.761355820929153]$	2.1434268×10^{-1}
2	$[0.626365384338657, 0.627118468877733]$	7.5308454×10^{-4}
3	$[0.626551181798084, 0.626551181835295]$	$3.7211234 \times 10^{-11}$

(a) Refinement of the interval $[t_0, t_1]$ which bounds potential intersections on $\mathbf{P}$.

i	$\mathbf{Q}$ *Intersection Interval* $[u_0^i, u_1^i]$	*Width of* $[u_0^i, u_1^i]$
0	$[0, 1]$	1.0000000×10^{0}
1	$[0.326756310053875, 0.392066137230932]$	6.5309827×10^{-2}
2	$[0.369959987612443, 0.369975295646327]$	1.5308034×10^{-5}
3	$[0.369965211165735, 0.369965211165736]$	$2.2204460 \times 10^{-16}$

(b) Refinement of the interval $[u_0, u_1]$ which bounds potential intersections on $\mathbf{Q}$.

Table 1.1: Convergence of intersection intervals for curves in Figure 1.6.

The interval $[t_0^i, t_1^i]$ converges to the parameter value of the intersection on curve $\mathbf{P}$ as $i \to \infty$. Likewise, $[u_0^i, u_1^i]$ always contains the intersection on $\mathbf{Q}$ and narrows until the intersection parameter is found. Table 1.1 shows the values of these intervals for each iteration of the intersection process. Table 1.1a illustrates the progress of $[t_0^i, t_1^i]$ on curve $\mathbf{P}$ while Table 1.1b shows $[u_0^i, u_1^i]$ on curve $\mathbf{Q}$. Note that the third column of these tables contains the width of the parameter interval. The width of $[t_0^i, t_1^i]$ is simply the difference $t_1^i - t_0^i$. The width is useful for gauging how tightly an interval has bounded a quantity by showing which digit first accounts for

the difference between the upper and lower bound. The interval widths show that the number of matching digits approximately triples with each iteration. By the fourth iteration, t^* and u^* have been approximated with over 16 digits of accuracy.

1.7 Overview

Geometric intervals provide an elegant basis for the curve/curve intersection algorithm outlined here. The concept of a geometric interval also leads to a unique method for finding intersections between a ray and a surface. In the ray/surface algorithm, a geometric interval is constructed for the surface, enabling us to simplify the search for intersections. This thesis details both of these geometric interval-based intersection algorithms.

Chapter 2 contains an introduction to the background material that serves as the theoretical foundations for this research. This chapter includes a discussion of interval arithmetic, Bézier curves, Bézier surfaces, interval Bézier curves and hybrid curves. Chapter 3 outlines the developement of the curve/curve intersection algorithm, beginning with an overview of previous work on finding intersections between planar curves. Chapter 4, on ray/surface intersections, follows a similar progression, starting with a discussion of current ray/surface intersection algorithms and then documenting the development of the geometric interval method. Chapter 5 concludes with remarks on the effectiveness of geometric intervals for finding intersections as well as a discussion of future directions to be explored.

Chapter 2

Background

This chapter introduces the mathematical foundations on which geometric intervals are based, including interval arithmetic, Bézier curves, Bézier surfaces, interval Bézier curves and hybrid curves. Each of these concepts plays an important role in understanding the curve/curve and ray/surface intersection algorithms.

2.1 Interval Arithmetic

Standard interval arithmetic techniques form the framework for the geometric interval methods developed in this thesis. For a more thorough discussion of interval arithmetic see Ramon E. Moore's *Interval Analysis* [17].

A scalar interval is a closed set of real values. Intervals are written [a,b], where

$$[a, b] = \{x \mid a \leq x \leq b\}. \tag{2.1}$$

We define $\mathbb{IR}$ to be the set of all real scalar intervals.

Intervals can be thought of as numbers for which the exact value is uncertain. Viewed in this way, we can define the standard arithmetic operations for intervals. These operations form an *interval arithmetic*. If $*$ is one of the binary arithmetic operations in $\{+, -, \cdot, /\}$ and we apply $*$ to the intervals $[a, b]$ and $[c, d]$, the result is

$$[a, b] * [c, d] = \{x * y \mid x \in [a, b],\ y \in [c, d]\}. \tag{2.2}$$

Equivalently, we define the following in terms of the endpoints a, b, c and d:

$$[a,b] + [c,d] = [a+c, b+d] \tag{2.3}$$

$$[a,b] - [c,d] = [a-d, b-c] \tag{2.4}$$

$$[a,b] \cdot [c,d] = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \tag{2.5}$$

$$[a,b] \,/\, [c,d] = \begin{cases} [a,b] \cdot [1/d, 1/c] & \text{if } 0 \notin [c,d], \\ [-\infty, a/c] & \text{if } 0 \le a < b \text{ and } c < d = 0, \\ [a/d, +\infty] & \text{if } 0 \le a < b \text{ and } c = 0 < d, \\ \{[-\infty, a/c] \cup [a/d, +\infty]\} & \text{if } 0 \le a < b \text{ and } c < 0 < d, \\ [-\infty, b/d] & \text{if } a < b \le 0 \text{ and } c = 0 < d, \\ [b/c, +\infty] & \text{if } a < b \le 0 \text{ and } c < d = 0, \\ \{[-\infty, b/d] \cup [b/c, +\infty]\} & \text{if } a < b \le 0 \text{ and } c < 0 < d, \\ [-\infty, +\infty] & \text{if } a \le 0 \le b \text{ and } c \le 0 \le d, \\ \emptyset & \text{if } 0 \notin [a,b] \text{ and } c = d = 0. \end{cases} \tag{2.6}$$

The cases of Equation 2.6 are required to handle division by zero (see [10, 21]). Note that when $c < 0 < d$ the result may be the union of two intervals. Representing this set as $[-\infty, +\infty]$, the tightest interval containing the set, ensures that division always yields a single interval. This is the convention which we have chosen to follow.

Any real number a is equivalent to the interval $[a, a]$. This is useful as it enables us to mix intervals and real valued operands in Equation 2.2. For example, we can write $a + [c,d] = [a+c, a+d]$. To distinguish a real variable from its interval counterpart, we use the notation $[x]$ to denote that the variable x is an interval.

It follows directly from Equation 2.3 and Equation 2.5 that interval addition and multiplication are commutative and associative. That is,

$$[u] + ([v] + [w]) = ([u] + [v]) + [w] \qquad [u] + [v] = [v] + [u] \tag{2.7}$$

$$[u] \cdot ([v] \cdot [w]) = ([u] \cdot [v]) \cdot [w] \qquad [u] \cdot [v] = [v] \cdot [u] \tag{2.8}$$

However, interval multiplication is only *subdistributive* over addition:

$$[u] \cdot ([v] + [w]) \subseteq [u][v] + [u][w]. \tag{2.9}$$

A notable exception to this occurs when $[u]$ degenerates to a real scalar value u. In this case, for any $u, [v]$ and $[w]$, the following holds

$$u \cdot ([v] + [w]) = u[v] + u[w]. \tag{2.10}$$

For convenience, we will usually omit the dot indicating multiplication, preferring the simpler notation $[u][v] \equiv [u] \cdot [v]$.

It is often useful to speak of the *width* of an interval when discussing how tightly an interval bounds some unknown quantity. Likewise, it is common to refer to a real value that is midway between the endpoints of an interval, called the *midpoint*. The width $w([x])$ and midpoint $m([x])$ of an interval $[x] = [a, b]$ are defined as

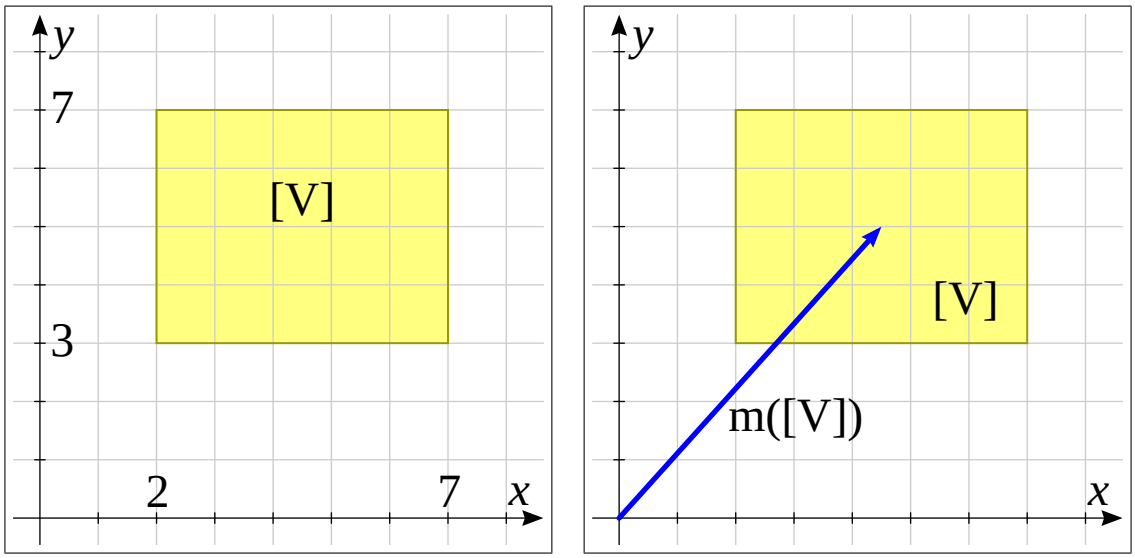
$$w([a, b]) = b - a, \qquad m([a, b]) = \frac{a + b}{2}. \tag{2.11}$$

2.1.1 Vectors

Vectors can also be made up of interval components. An n-dimensional interval vector $[\mathbf{V}] \in \mathbb{IR}^n$ has elements $[v_1], \ldots, [v_n]$, whose component $[v_i]$ is an interval bound in dimension i. This can be conceptualized as an axis-aligned hyperrectangle of dimension n. The width and midpoint of an interval vector are defined as

$$w([\mathbf{V}]) = (w([v_1]), \ldots, w([v_n])), \tag{2.12}$$

$$m([\mathbf{V}]) = (m([v_1]), \ldots, m([v_n])). \tag{2.13}$$



(a) The interval vector $[\mathbf{V}] = ([2,7],[3,7])$. (b) $[\mathbf{V}]$ contains its midpoint vector $m([V])$.

Figure 2.1: Interval vectors in $\mathbb{IR}^2$ are axis-aligned rectangular areas.

Figure 2.1a illustrates the vector $[\mathbf{V}] = ([2,7],[3,7])$. This vector contains all vectors (x,y) where $x \in [2,7]$ and $y \in [3,7]$. The midpoint of this interval vector is $m([\mathbf{V}]) = (4.5,5)$, as shown in Figure 2.1b.

2.1.2 Polynomials

The coefficients and/or variables of a polynomial can be intervals. For example, Equation 1.2 in Section 1.3 was evaluated using the interval $[t] = [0,1]$ to find an interval vector bounding the derivative of a Bézier curve. Evaluating this polynomial required nothing more than following the rules of addition and multiplication as defined above. However, we first factored this polynomial using the well known Horner's scheme [11] to obtain

$$\mathbf{P}'([t]) = (3(\mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0)[t] + 6(\mathbf{P}_2 - 2\mathbf{P}_1 + \mathbf{P}_0))[t] + 3(\mathbf{P}_1 - \mathbf{P}_0). \quad (2.14)$$

The use of Horner's rule to factor interval polynomials before evaluation is common as it tends to reduce overestimation (see [6, 18]).

2.1.3 Overestimation

Whenever interval arithmetic is used with polynomials the result generally will be overly conservative. The two main sources of this overestimation are the the *dependency problem* and the fact that interval vectors bound each dimension independently. The dependency problem arises whenever interval arithmetic *decorrelates* intervals that are actually identical. That is, a seemingly simple operation such as $[u]^2$ is calculated as

$$[u]^2 = [u] \cdot [u] = \{x \cdot y \mid x \in [u],\ y \in [u]\} \tag{2.15}$$

and, therefore, the two instances of $[u]$ are treated as if they were independent. Moore [17] shows that overestimation may occur when evaluating any expression for which interval variables occur more than once.

Another type of overestimation occurs due to the fact that interval vectors are only capable of bounding each dimension independently, resulting in an axis-aligned rectangular bound. As a result, the true solution set of an expression which is oblique and thin would be contained in an area much larger than necessary.

A third type of overestimation arises in computer software when floating point numbers are used for interval bounds. Floating point numbers are limited in precision and require rounding when a real number cannot be exactly represented. However, rounded interval arithmetic can be properly implemented with the aid of standard floating point hardware [10]. Tracking floating point rounding errors results in overestimation that is small compared to the other sources of overestimation.

2.2 Bézier Curves

Our curve/curve intersection algorithm is based on Bézier curves. See [5] for details of Bézier curves, including material outside the scope of this paper.

Bézier curves are expressed in terms of the Bernstein polynomial basis functions

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i, \quad 0 \leq i \leq n. \tag{2.16}$$

A degree n Bézier curve $\mathbf{P}$ is defined as

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t), \quad t \in [0,1]. \tag{2.17}$$

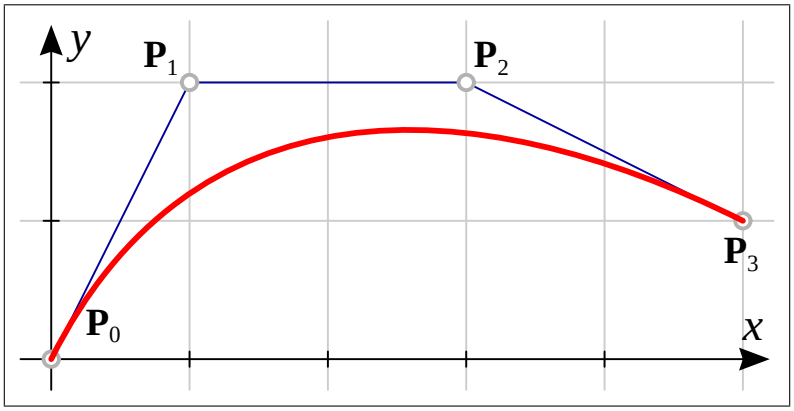
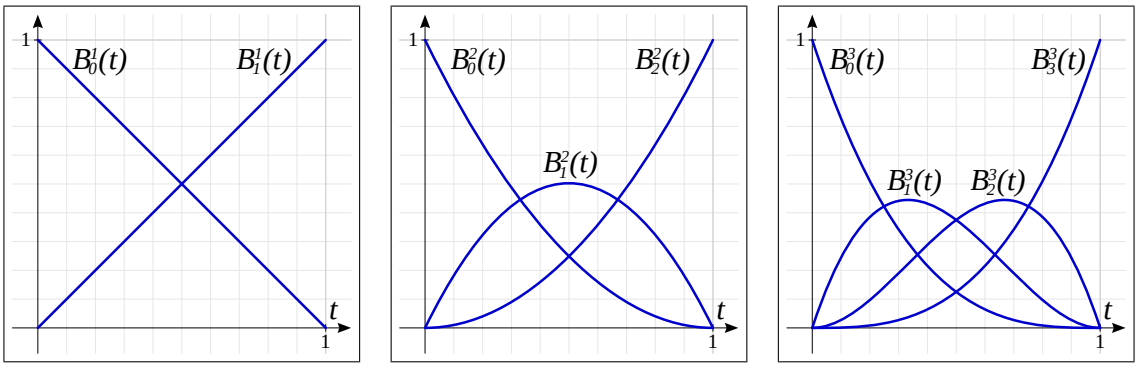


Figure 2.2: A cubic Bézier curve and its control polygon.

The vector coefficients $\mathbf{P}_i = (x_i, y_i)$ are the Bézier *control points*. For example, the cubic Bézier curve in Figure 2.2 has control points $\mathbf{P}_0 = (0,0)$, $\mathbf{P}_1 = (1,2)$, $\mathbf{P}_2 = (3,2)$ and $\mathbf{P}_3 = (5,1)$. The control points, connected in order, form what is referred to as the *Bézier polygon* or *control polygon* of the curve segment. This control polygon approximates the shape of the curve, a property that makes Bézier curves intuitive to use in an interactive environment.

A point $\mathbf{P}(t)$ on a Bézier curve is a weighted average of its control points, with the influence of each control point varying with t. $B_i^n(t)$ defines the influence of control point P_i on the curve for a given parameter value.



(a) Bernstein polynomials B_i^1. (b) Bernstein polynomials B_i^2. (c) Bernstein polynomials B_i^3.

Figure 2.3: Linear, quadratic and cubic Bernstein polynomials $B_i^n(t)$ for $t \in [0, 1]$.

Bernstein basis functions for degree $n \in \{1, 2, 3\}$ are illustrated in Figure 2.3. Some important properties of these basis functions include:

- Partition of unity: $$\sum_{i=0}^{n} B_i^n(t) \equiv 1, \tag{2.18}$$
- Positivity: $$B_i^n(t) \geq 0, \ t \in [0, 1], \tag{2.19}$$
- Recursion: $$B_i^n(t) = (1 - t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t). \tag{2.20}$$

Partition of unity ensures that the relationship between the Bézier control points P_i and the curve remain constant under affine transformations. This means that operations such as translation, rotation or scaling on the control points will result in the curve being translated, rotated or scaled in exactly the same manner.

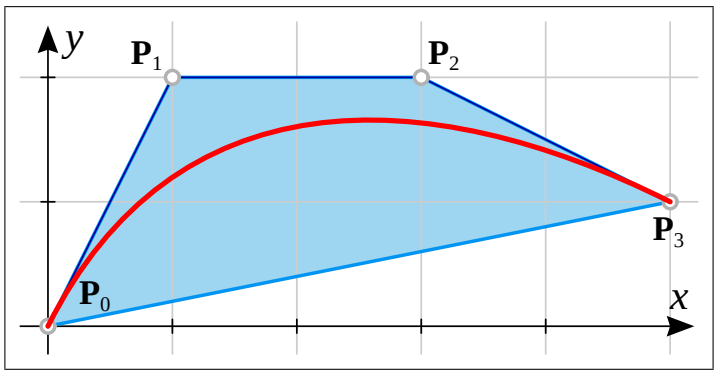


Figure 2.4: A Bézier curve is completely contained within its *convex hull*.

Partition of unity and positivity guarantee that point $\mathbf{P}(t),\ t \in [0,1]$ can be expressed as a convex combination of the Bézier control points. Therefore, the entire curve lies completely within the convex hull formed by the points $\mathbf{P}_i$. The convex hull of the set of points $\{\mathbf{P}_i\}$ is the smallest convex set containing $\{\mathbf{P}_i\}$. The shaded region of Figure 2.4 is an example of a convex hull. The convex hull property of Bézier curves is often exploited in algorithms to bound the location of the curve.

The property of recursion enables us to express $B_i^n(t)$ using linear interpolation of the Bernstein polynomials $B_{i-1}^{n-1}(t)$ and $B_i^{n-1}(t)$, each of which are degree $n-1$. Each of these can then be expressed using Bernstein polynomials of degree $n-2$. Since $B_i^0(t) \equiv 1$, any Bernstein polynomial can be constructed through repeated linear interpolation.

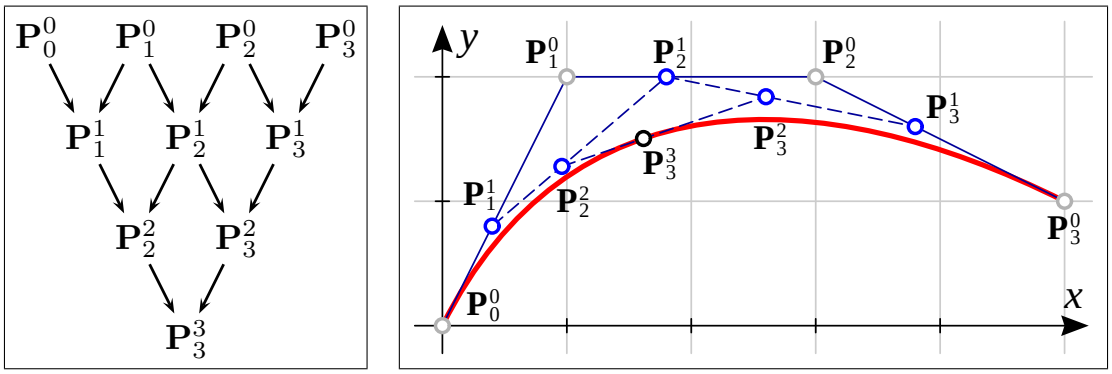
2.2.1 The de Casteljau Algorithm

The decomposition of Bernstein basis functions through repeated linear interpolation leads directly to the de Casteljau algorithm [8] for the evaluation of Bézier curves. The de Casteljau algorithm defines a set of intermediate points

$$\mathbf{P}_i^n(t) = (1-t)\mathbf{P}_{i-1}^{n-1} + t\mathbf{P}_i^{n-1}, \tag{2.21}$$

where $\mathbf{P}_i^0 = \mathbf{P}_i$. These intermediate points are shown in Figure 2.5a for $n = 3$.

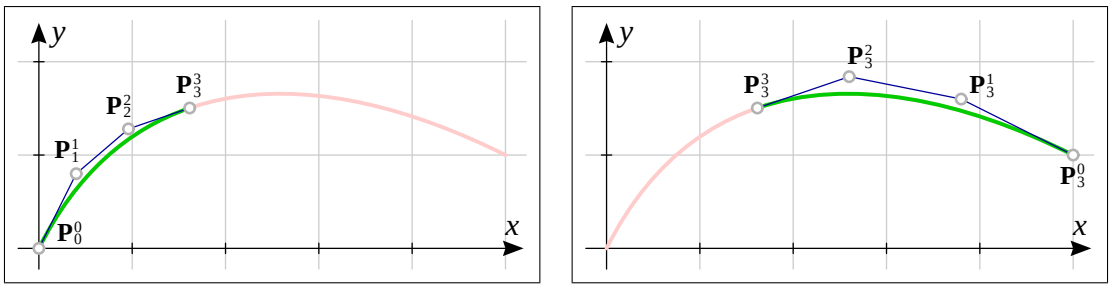
Through recursive application of Equation 2.21, we obtain a single point $\mathbf{P}_n^n$ which is equivalent to $\mathbf{P}(t)$. This process of repeated linear interpolation is illustrated for a cubic curve in Figure 2.5b. This is not the most computationally efficient method to evaluate $\mathbf{P}(t)$ for higher values of n. However, the de Casteljau algorithm is a numerically stable method to evaluate points on a curve as well as an important theoretical basis for the study of Bézier curves.



(a) Intermediate points. (b) Evaluation of $\mathbf{P}(0.4) = \mathbf{P}_3^3$ using the de Casteljau algorithm.

Figure 2.5: The de Casteljau algorithm repeatedly interpolates control points.

Besides evaluation of points on the curve, the de Casteljau algorithm can also be used to find the Bézier control points for a portion of a curve. A point $\mathbf{P}(t)$ subdivides a curve into two segments, corresponding to the parameter intervals $[0, t]$ and $[t, 1]$. The control polygon for the first segment is made up of auxiliary points $\mathbf{P}_0^0, \mathbf{P}_1^1, \ldots, \mathbf{P}_n^n$ from the de Casteljau scheme. The first curve segment from Figure 2.5b is shown in Figure 2.6a. Similarly, points $\mathbf{P}_n^n, \mathbf{P}_n^{n-1}, \ldots, \mathbf{P}_n^0$ define the control polygon for the second subcurve, as shown in Figure 2.6b.



(a) Segment one, formed from points $\mathbf{P}_i^i, i \leq n$. (b) Segment two, formed from points $\mathbf{P}_n^i, i \leq n$.

Figure 2.6: Curve subdivision using intermediate points from Figure 2.5.

It is possible to chain Bézier curves together into a so-called composite Bézier curve. For example, the two curve segments in Figure 2.6 meet smoothly at point $\mathbf{P}_3^3$, forming a composite curve that could be continued by appending a new curve at point $\mathbf{P}_0^0$ or $\mathbf{P}_3^0$. There are simple rules for determining the continuity of the composite

curve where segments meet, a fact that has helped to drive the popularity of Bézier curves. See [5] for more information on composite Bézier curves.

2.2.2 Rational Curves

The Bézier curves discussed so far are called *polynomial* Bézier curves. This is to distinguish them from *rational* Bézier curves, which include a nonnegative weight w_i associated with each control point $\mathbf{P}_i$. Weights are used to adjust the shape of the curve, as shown in Figure 2.7.

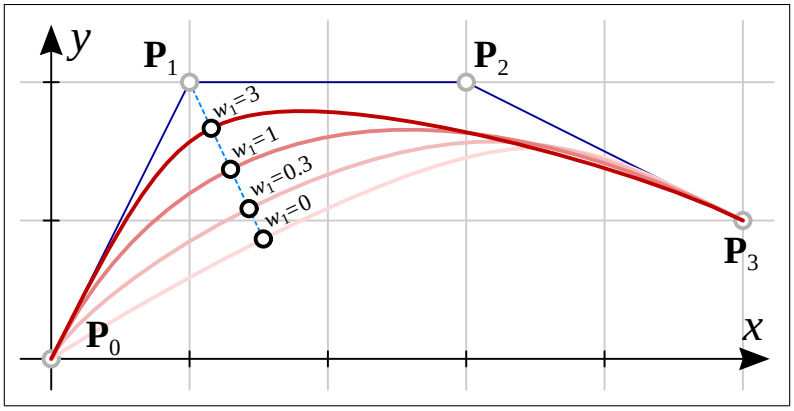


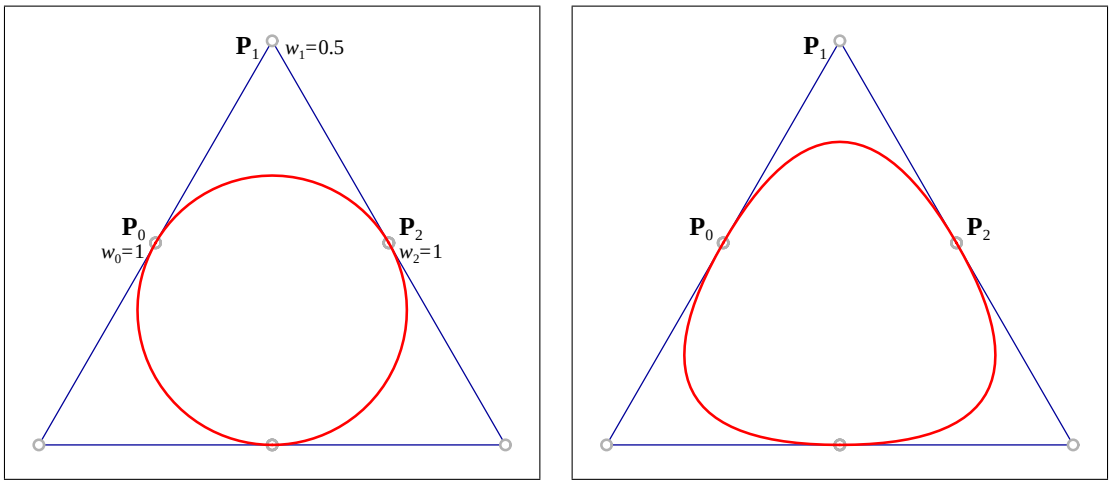
Figure 2.7: A rational Bézier curve with weights $w_i = 1$ except w_1.

Each weight w_i scales the basis function $B_i^n(t)$ that controls the influence of control point $\mathbf{P}_i$. A rational Bézier curve is expressed as

$$\mathbf{P}(t) = \frac{\sum_{i=0}^{n} \mathbf{P}_i w_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)}. \tag{2.22}$$

Rational curves were first introduced into the field of computer-aided geometric design by Steven A. Coons [7]. Rational Bézier curves are a more flexible representation since they encompass polynomial curves (by setting $w_i = 1$ for $i = 0, \ldots, n$) and make it possible to represent conic sections. For example, Figure 2.8a is a composite Bézier curve made up of three rational quadratic Bézier segments. For comparison, Figure 2.8b shows the corresponding composite curve made up of polynomial Bézier

segments. With the exception of parabolas, conic sections can only be approximated with polynomial Bézier curves.



(a) Circle composed of rational Bézier curves. (b) Composite polynomial Bézier curve.

Figure 2.8: Rational Bézier curves make it possible to represent conic sections.

The added generality of rational curves does, however, carry a computational cost. Many algorithms, such as the de Casteljau algorithm, can be implemented by treating the numerator and denominator of Equation 2.22 as if they were independent curves [5]. Equivalently, the control points $\mathbf{P}_i$ in $\mathbb{R}^d$ can be mapped to homogeneous coordinates $(w_i\mathbf{P}_i, w_i)$ in $\mathbb{R}^{d+1}$. The algorithm is then applied to these coordinates and the resulting control points $(w_j\mathbf{P}_j, w_j)$ are projected back to $\mathbb{R}^d$ by dividing out the weight: $\mathbf{P}_j = \frac{1}{w_j}(w_j\mathbf{P}_i, w_j)$. Homogeneous coordinates simplify the process by packing the weight with position, allowing both to be computed with a single application of the algorithm.

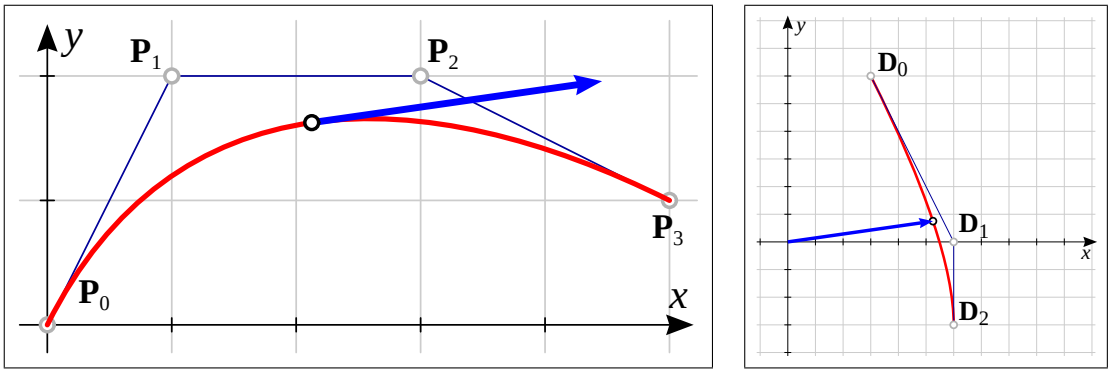
Some algorithms cannot be adapted as readily to the rational Bézier case. For example, finding the derivative of a polynomial Bézier curve is a fairly simple process, as described in the next section. The derivative of a rational Bézier curve cannot be computed directly using the same procedure.

2.2.3 Derivatives

The first derivative of a degree n polynomial Bézier curve can be expressed as a degree $n-1$ polynomial Bézier curve with control points

$$\mathbf{D}_i = n(\mathbf{P}_{i+1} - \mathbf{P}_i). \tag{2.23}$$

The resulting derivative curve is called a *hodograph*. The vector $\mathbf{P}'(0.5)$ is shown as the tangent vector at $\mathbf{P}(0.5)$ of the Bézier curve in Figure 2.9a. The same vector is shown in Figure 2.9b as a vector from the origin to the hodograph point $\mathbf{D}(0.5) = \mathbf{P}'(0.5)$. The control points for this curve are $D_0 = 3(P_1 - P_0)$, $D_1 = 3(P_2 - P_1)$ and $D_2 = 3(P_3 - P_2)$.



(a) Derivative of a Bézier curve at $t = 0.5$.

(b) Hodograph of (a).

Figure 2.9: Hodographs define the derivative at each point along the curve.

Higher derivatives can be obtained by repeating this process on the hodograph curve. Each differentiation step produces a new curve that is one degree lower than the prior curve.

Constructing a hodograph using this process is valid *only* for polynomial Bézier curves. The hodograph of a rational Bézier curve must instead be expressed as a degree $2n$ rational Bézier curve.

2.3 Bézier Surface Patches

A degree $n \times m$ polynomial tensor product Bézier surface patch is defined by

$$\mathbf{P}(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{P}_{i,j} B_i^n(s) B_j^m(t), \tag{2.24}$$

where s and t range over the unit parameter square: $s \in [0,1],\ t \in [0,1]$. Note that the blending function for each control point $\mathbf{P}_{i,j}$ is the product of two Bernstein basis functions $B_i^n(s)B_j^m(t)$, one for each of the parametric parameters s and t.

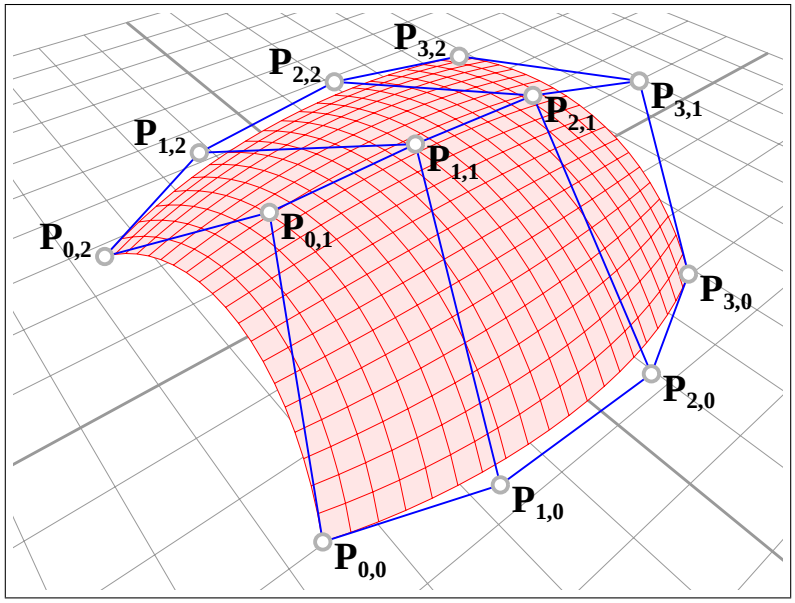


Figure 2.10: A degree 3×2 polynomial Bézier patch.

Figure 2.10 illustrates a degree 3×2 Bézier patch defined by an array of control points $\mathbf{P}_{i,j}$ arranged in an $(n+1) \times (m+1)$ grid. This grid is called the *Bézier net* or *control grid*. The four corner control points, $\mathbf{P}_{0,0}$, $\mathbf{P}_{n,0}$, $\mathbf{P}_{0,m}$ and $\mathbf{P}_{n,m}$, are always on the surface, just as the first and last Bézier curve control points lie on the curve. In fact, the edges of the patch are Bézier curves. For example, if we set $t = 0$ in

Equation 2.24, we obtain

$$\mathbf{P}(s,0) = \sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{P}_{i,j}B_i^n(s)B_j^m(0) = \sum_{i=0}^{n}\mathbf{P}_{i,0}B_i^n(s) \tag{2.25}$$

since $B_0^m(0) = 1$ and $B_j^m(0) = 0$ for $j > 0$. This is simply a Bézier curve defined by the n control points $\mathbf{P}_{i,0}$. The same process can be repeated for $s = 0,\ s = 1$ and $t = 1$ to show that each edge of a Bézier patch corresponds to a Bézier curve.

Since Bézier patches are defined using the Bernstein basis functions $B_i^n(s)$ and $B_j^m(t)$ they share many properties with Bézier curves, including:

- Partition of unity:
$$\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(s)B_j^m(t) \equiv 1, \tag{2.26}$$

- Positivity:
$$B_i^n(s)B_j^m(t) \geq 0,\ s \in [0,1],\ t \in [0,1], \tag{2.27}$$

- Recursion:
$$\begin{aligned} &B_i^n(s)B_j^m(t) \\ &\quad = (1-s)B_i^{n-1}(s)B_j^m(t) + sB_{i-1}^{n-1}(s)B_j^m(t) \\ &\quad = (1-t)B_i^n(s)B_j^{m-1}(t) + tB_i^n(s)B_{j-1}^{m-1}(t) \end{aligned} \tag{2.28}$$

As before, partition of unity ensures that the relationship between the surface and its control points is affinely invariant. With the positivity property, partition of unity also guarantees that each point on the surface can be expressed as a convex combination of control points and, therefore, a Bézier patch lies within the convex hull of its control polygon.

2.3.1 The de Casteljau Algorithm

The property of recursion enables us to apply the de Casteljau algorithm to the control grid to evaluate $\mathbf{P}(s,t)$ or to subdivide the patch (see Section 2.2.1).

To evaluate a point on the surface, the de Casteljau algorithm is first applied successively to the $n+1$ control points of each row, resulting in $m+1$ evaluated

points. These points are the control vertices of an s *iso-parameter* curve, a Bézier curve on the surface corresponding to a constant s parameter value. This curve is then evaluated at t using the de Casteljau algorithm to obtain $\mathbf{P}(s,t)$. Alternatively, we could first evaluate each column to obtain a t iso-parameter curve which we would then evaluate at s. This process is illustrated in Figure 2.11.

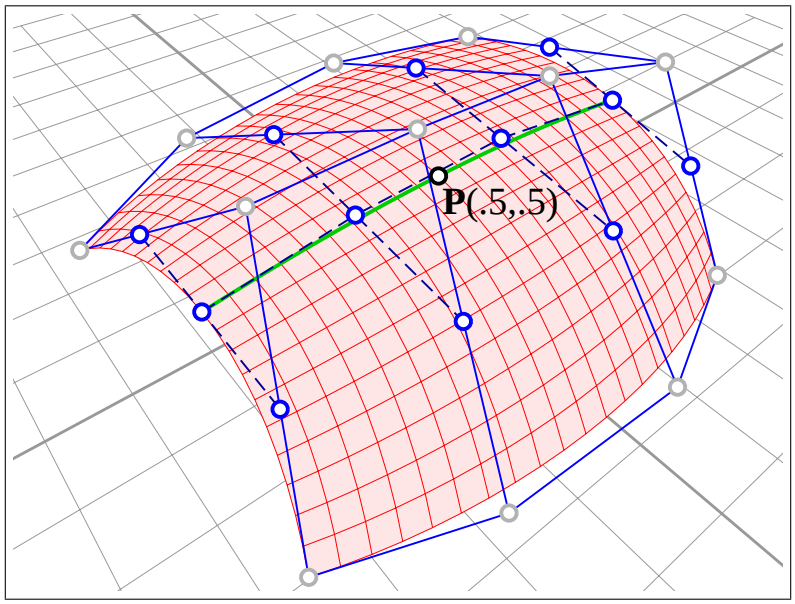
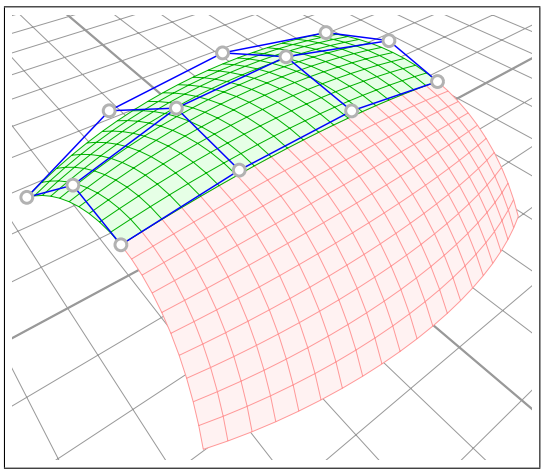
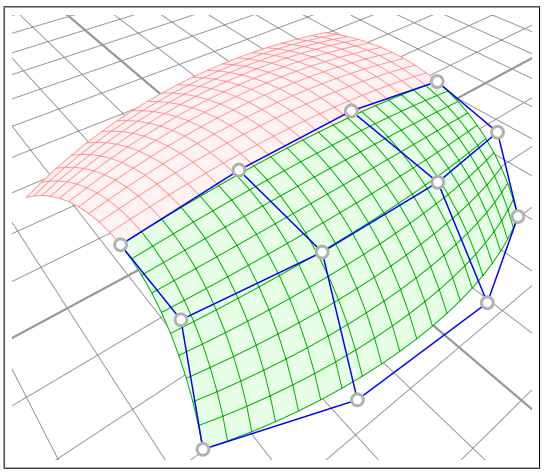


Figure 2.11: Evaluation of $\mathbf{P}(.5,.5)$ using the de Casteljau algorithm.



(a) Segment one, spanning $t \in [0.5, 1]$.



(b) Segment two, spanning $t \in [0, 0.5]$.

Figure 2.12: Subdivision of a Bézier patch using the de Casteljau algorithm.

A patch can be subdivided at some parameter value s by using the de Casteljau algorithm to subdivide each row of the control polygon as if it were a degree n curve. The result is two new surfaces, dividing the s parameter into the intervals $[0, s]$ and $[s, 1]$. Subdivision in the t parameter direction is accomplished by applying the de Casteljau algorithm to each column. Figure 2.12 shows the surface from Figure 2.10 split at $t = 0.5$. Repeated subdivision makes it possible to extract a Bézier subpatch from the original surface for an arbitrary parameter region $([s_0, s_1], [t_0, t_1])$.

2.3.2 Rational Surfaces

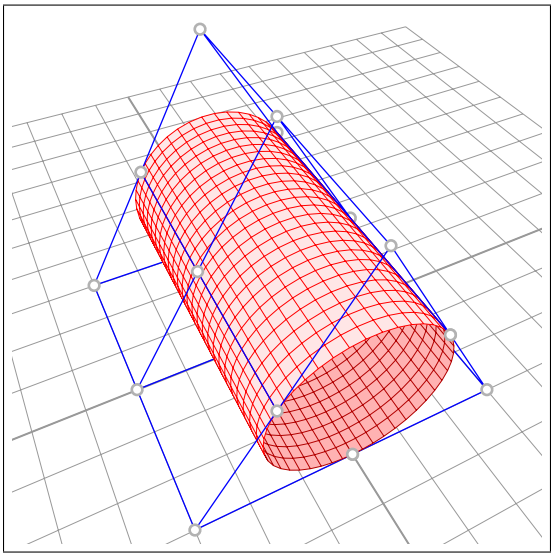


Figure 2.13: Rational Bézier patches arranged to form a cylinder.

As with Bézier curves, it is also possible to associate weights with the control points to attain greater flexibility in defining shape. Doing so forms a rational Bézier patch, which is expressed as

$$\mathbf{P}(s,t) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{P}_{i,j}w_{i,j}B_i^n(s)B_j^m(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{i,j}B_i^n(s)B_j^m(t)}. \tag{2.29}$$

The benefits of rational Bézier patches are the same as those of rational curves (see Section 2.2.2). Without rational surfaces it would only be possible to approximate

shapes like cones, spheres or cylinders (such as the one in Figure 2.13) using Bézier surface patches.

2.3.3 Derivatives

Partial derivatives of Bézier surface patches are found by applying the techniques presented in Section 2.2.3 to each row or column of the surface. For example, to find the s hodograph surface $\mathbf{P_s}(s,t)$, corresponding to the first partial in s, we find the hodograph of each row in the Bézier net. $\mathbf{P_t}(s,t)$ consists of the hodograph of each column of the control polygon.

Mixed partials are found by constructing the hodograph for one parameter direction followed by the other. For example, the mixed partial $\mathbf{P_{st}}(s,t)$ can be found as the t partial of $\mathbf{P_s}(s,t)$ or as the s partial of $\mathbf{P_t}(s,t)$. Of course, higher derivatives can be found by repeated application of the hodograph rules: $\mathbf{P_{ss}}(s,t)$ is the s hodograph of $\mathbf{P_s}(s,t)$.

Derivatives of rational surfaces are more problematic, as with rational Bézier curves. See [22] for a discussion on finding derivatives of rational Bézier patches.

2.4 Interval Bézier Curves

The concept of the interval Bézier curve was introduced by Sederberg and Farouki [25], where an interval Bézier curve was used to retain error bounds from an approximation process. An interval Bézier curve simply uses interval vectors for control points and is evaluated using standard interval arithmetic.

Interval Bézier curves will be used in this thesis in a simplified form. A Bézier curve may be used to represent a polynomial in the Bernstein basis by restricting each control point to a single scalar. When this is done, the curve is known as an *explicit Bézier curve* or a *Bernstein polynomial*. An interval Bernstein polynomial

is, therefore, a polynomial in the Bernstein basis with interval coefficients. Interval Bernstein polynomials are also discussed in [25].

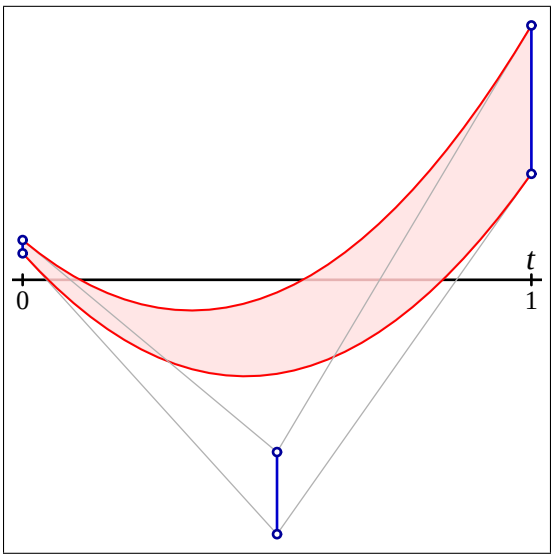


Figure 2.14: A quadratic interval Bernstein polynomial.

For example, the Bernstein polynomial in Figure 2.14 has interval coefficients, as indicated by the vertical bars at each control point. As with all Bernstein polynomials the control points are evenly distributed in s between 0 and 1. The polynomial is bounded by an upper and lower curve, found using the upper and lower extremes of each coefficient, respectively. The roots of this interval polynomial are themselves intervals, with each root being defined by the roots of the bounding curves.

2.5 Hybrid Curves and Surfaces

We refer to any curve or surface that is allowed to have a moving control point as a *hybrid* curve or surface. A *moving control point* is a control vertex with a variable location defined by some other curve or surface. The equation which defines the moving control point shares one or more parameters with the hybrid. For example, a hybrid Bézier curve $\hat{\mathbf{P}}(t)$ could have a moving control point $\hat{\mathbf{P}}_1(t)$ which is the point at t on the Bézier curve defining the path of the moving control point.

Ball [1] originated the idea of using moving control points to approximate curves. He formulated a rational cubic curve as a quadratic Bézier curve by setting the middle control point in motion. Later, Sederberg and Kakimoto [26] expanded on this idea, using hybrid polynomial Bézier curves to approximate rational Bézier curves.

As an illustration, we will now construct a simple hybrid curve. We will derive a quadratic hybrid curve with a single moving control point that is equivalent to a cubic polynomial Bézier curve. To find the path that the moving control point follows, we start with the equation of a cubic polynomial Bézier curve:

$$\mathbf{P}(t) = \mathbf{P}_0 B_0^3(t) + \mathbf{P}_1 B_1^3(t) + \mathbf{P}_2 B_2^3(t) + \mathbf{P}_3 B_2^3(t). \tag{2.30}$$

Expanding the Bernstein polynomials $B_i^3(t)$ we obtain

$$\mathbf{P}(t) = \mathbf{P}_0(1-t)^3 + 3\mathbf{P}_1(1-t)^2 t + 3\mathbf{P}_2(1-t)t^2 + \mathbf{P}_3 t^3. \tag{2.31}$$

We then rewrite the first and last terms by applying

$$\mathbf{P}_0(1-t)^3 = \mathbf{P}_0(1-t)^2 - \mathbf{P}_0(1-t)^2 t, \tag{2.32}$$

$$\mathbf{P}_3 t^3 = \mathbf{P}_3 t^2 - \mathbf{P}_3(1-t)t^2, \tag{2.33}$$

which allows us to rewrite the cubic curve equation using the quadratic Bernstein basis functions $B_0^2(t) = (1-t)^2$ and $B_2^2(t) = t^2$ for the first and last control point. We then group everything else to find the moving control point:

$$\begin{aligned}
\mathbf{P}(t) &= \mathbf{P}_0(1-t)^2 + (3\mathbf{P}_1 - \mathbf{P}_0)(1-t)^2 t + (3\mathbf{P}_2 - \mathbf{P}_3)(1-t)t^2 + \mathbf{P}_3 t^2 \\
&= \mathbf{P}_0(1-t)^2 + [(3\mathbf{P}_1 - \mathbf{P}_0)(1-t) + (3\mathbf{P}_2 - \mathbf{P}_3)t](1-t)t + \mathbf{P}_3 t^2 \\
&= \mathbf{P}_0 B_0^2(t) + \left[\frac{(3\mathbf{P}_1 - \mathbf{P}_0)}{2}(1-t) + \frac{(3\mathbf{P}_2 - \mathbf{P}_3)}{2}t\right] B_1^2(t) + \mathbf{P}_3 B_2^2(t).
\end{aligned} \tag{2.34}$$

We may therefore express the cubic Bézier curve $\mathbf{P}$ as a hybrid quadratic curve $\hat{\mathbf{P}}$ with control points

$$\hat{\mathbf{P}}_0 = \mathbf{P}_0, \quad \hat{\mathbf{P}}_1(t) = \frac{(3\mathbf{P}_1 - \mathbf{P}_0)}{2}(1-t) + \frac{(3\mathbf{P}_2 - \mathbf{P}_3)}{2}t \quad \text{and } \hat{\mathbf{P}}_2 = \mathbf{P}_3. \tag{2.35}$$

This hybrid curve is shown in Figure 2.15.

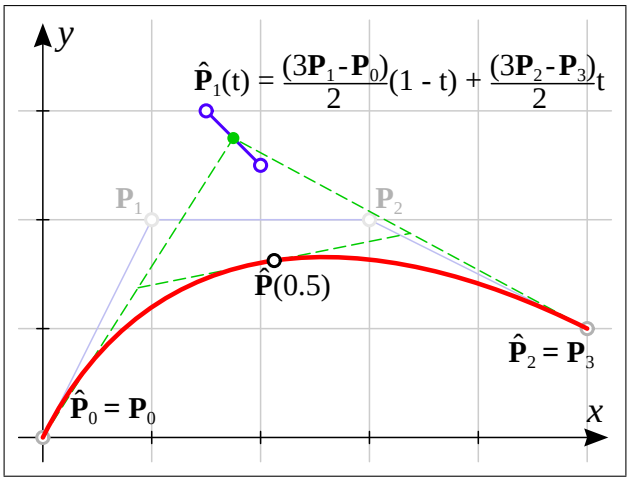


Figure 2.15: A cubic Bézier curve expressed as a hybrid quadratic curve.

To evaluate a point on a hybrid curve, we first determine the location of all moving control points given the parameter value t. This is done by evaluating the expressions which control their position using t. Once the moving control points have been fixed, the hybrid curve can be evaluated as if it were a standard curve. For example, to evaluate $\hat{\mathbf{P}}(t)$ for $t = 0.5$ we must first evaluate $\hat{\mathbf{P}}_1(0.5)$ and then use the resulting point to evaluate $\hat{\mathbf{P}}(0.5)$, as shown in Figure 2.15.

The simplest way to create a hybrid curve is to use algebraic manipulation, as shown. Using the same principles we can express any Bézier curve using many different configurations of moving control points. It is also possible to reformulate a Bézier surface patch as a hybrid surface with one or more moving control points.

The geometry of hybrid curves and surfaces lends valuable insight into the properties of the curve or surface being represented. For example, the curve $\hat{\mathbf{P}}$ in

Figure 2.15 would be a true quadratic Bézier curve if the moving control point $\hat{\mathbf{P}}_1(t)$ were to degenerate to a single point. The size of a moving control point can therefore be viewed as a measure of how well $\mathbf{P}$, the original cubic curve, can be represented by a quadratic curve. Hybrid curves and surfaces, and the properties of the curves and surfaces they represent, play an important role in the development of geometric interval methods.

Chapter 3

Curve/curve Intersection

This chapter presents an algorithm, referred to as GeoClip, for calculating points of intersection between two planar Bézier curves. A core feature of GeoClip is the application of geometric intervals to streamline the task of processing curve geometry. GeoClip builds upon an algorithm which we will call BezClip [27], a popular and efficient algorithm that exhibits quadratic convergence.

BezClip takes advantage of the convex hull property of Bézier curves to identify curve regions that may contain intersections and therefore merit further investigation. The chief advantage of the geometric interval approach over BezClip is the ability to more accurately identify regions of interest. The result is an algorithm that is as robust as BezClip, but which exhibits cubic convergence.

Section 3.1 reviews several predominant approaches to the curve/curve intersection problem, including BezClip. Some of these algorithms form the basis for evaluating the performance of our technique. In Section 3.2, our algorithm, GeoClip, is presented in detail. Timing comparisons are provided in Section 3.3. Finally, Section 3.4 is devoted to observations and conclusions.

3.1 Previous Work

What follows is a brief overview of prominent curve/curve intersection algorithms, labeled Subdiv [14], KopInt [13], Impl [28] and BezClip [27].

SUBDIV [14] takes advantage of the convex hull property of Bézier curves. If the convex hulls of the two curves overlap, the curves may intersect. In this case the curves are subdivided using the de Casteljau algorithm. The process then continues recursively, subdividing curve segments with overlapping convex hulls until the remaining curve segments are sufficiently small to be declared intersection points. The SUBDIV algorithm is simple, but not problem free. For example, SUBDIV commonly reports the same intersection multiple times, convergence is only linear, and computing the convex hull can be expensive so approximations are usually preferred, increasing the number of iterations required.

KOPINT [13] employs a binary subdivision scheme, much like SUBDIV. However, in the KOPINT scheme the curves are first preprocessed by locating the parameter values corresponding to all horizontal and vertical tangents. The curve is then split into intervals which may only have these horizontal and vertical tangent points as end points, if at all. Any two points within such an interval are sufficient to define an axis-aligned bounding box which contains the curve segment between the points. Bounding boxes are defined for all intervals and tested against the bounding boxes for another curve. Intervals for overlapping bounding boxes are subdivided by evaluating the midpoints of the intervals, creating two new bounding boxes for each interval. The process repeats until points of intersection have been satisfactorily approximated. The power of this method lies in the fact that only a single point on the curve must be evaluated to subdivide an interval, a $O(n)$ operation that is much less expensive than the $O(n^2)$ de Casteljau algorithm. While this makes the KOPINT algorithm faster than SUBDIV, the rate of convergence is still only linear.

IMPL [24, 28] is an algebraic approach to finding curve/curve intersections. One curve's parametric equation is *implicitized*: converted from the parametric form $x = x(t)$, $y = y(t)$ to an implicit equation of the form $f(x, y) = 0$. The other curve's parametric equations $x(u)$ and $y(u)$ are substituted into the implicit equation to form

the polynomial $f(x(u), y(u)) = g(u)$, which has roots at all values of u for which an intersection point exists. Intersection locations are found by applying the u values to the parametric curve equation. If desired, parameter values for the implicitized curve can be found with an *inversion equation* which maps (x, y) locations on the implicit curve back to the corresponding parameter value t of the original parametric curve. The speed of IMPL depends heavily on the quality of the root finder used to solve $g(t) = 0$. Although IMPL is very fast for low degree curves, it is generally too inefficient and numerically unstable to be practical for curves of above degree five.

BEZCLIP [27] builds on the convex hull property and ideas from interval analysis to solve the curve/curve intersection problem. However, instead of using the convex hull directly, as is done in SUBDIV, BEZCLIP uses a simple geometric construction called a *fat line*. A fat line is the region between a pair of parallel lines which contains a Bézier curve. The method presented in [27] for constructing a fat line involves first forming a line $\overline{L}$ passing through $\mathbf{P}_0$ and $\mathbf{P}_n$, the first and last control points of the curve. The boundaries of the fat line are then chosen such that they are parallel to $\overline{L}$ and contain the curve between them. This can be done by ensuring that all control points are included, as shown in Figure 3.1.

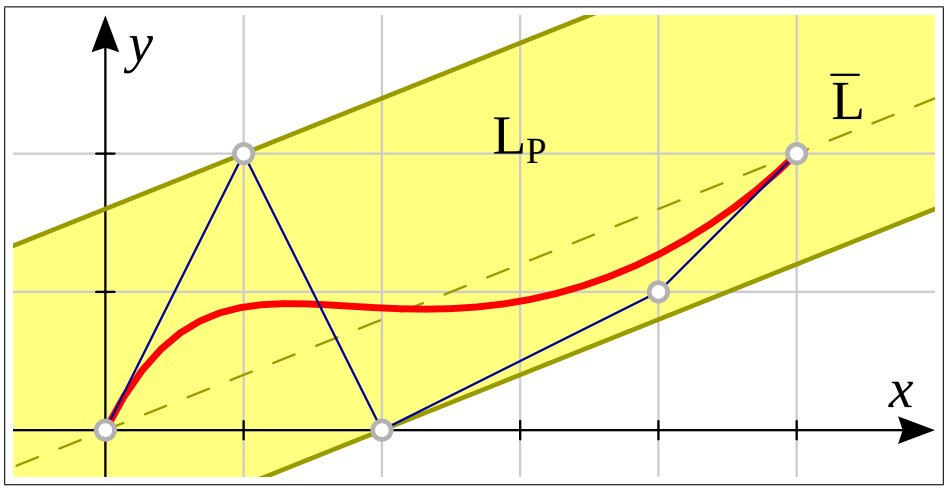


Figure 3.1: A fat line $\mathbf{L_P}$ which bounds a quartic curve $\mathbf{P}(t)$ by enclosing all $\mathbf{P}_i$.

Given a fat line $\mathbf{L_P}$ for curve $\mathbf{P}(t)$, a Bernstein polynomial curve representing the distance from $\mathbf{L_P}$ to another curve, $\mathbf{Q}(u)$, is calculated. The convex hull of this polynomial is then used to calculate an interval $[u_0, u_1]$ defining the parameter range where $\mathbf{Q}(u)$ overlaps $\mathbf{L_P}$. Outside of this range there can be no overlap between the curves and, therefore, $\mathbf{P}(t) \neq \mathbf{Q}(u)$. The curve $\mathbf{Q}$ is *clipped*, or subdivided, using the de Casteljau algorithm such that it spans only the region of interest, $[u_0, u_1]$. The process is then reversed by finding the fat line $\mathbf{L_Q}$ containing $\mathbf{Q}$ and calculating the interval $[t_0, t_1]$ where $\mathbf{P}(t)$ overlaps $\mathbf{L_Q}$. Each curve is iteratively refined until all intersection points have been identified to within tolerance.

BezClip uses the de Casteljau algorithm to perform clipping at each iteration, so the per-iteration complexity is similar to Subdiv. However, BezClip exhibits quadratic convergence since it discards segments of each curve more intelligently and is therefore significantly faster than Subdiv. Speed tests in [27] indicate that BezClip is always faster than Subdiv and generally faster than Impl for curves of degree higher than four.

Another notable algorithm is the algebraic pruning technique of [15]. This algorithm transforms curve/curve intersection into an eigenvalue problem. Algebraic pruning is competitive with BezClip for simple intersections but is significantly slower for more complicated cases.

3.2 The GeoClip Curve/curve Intersection Algorithm

GeoClip is essentially an extension to BezClip. The general process of clipping away nonintersecting curve segments remains intact. Fat lines are also retained as a mechanism for bounding Bézier curves. Our key contribution is the introduction of geometric intervals to improve calculation of the parameter interval $[t_0, t_1]$ where a curve $\mathbf{P}(t)$ overlaps a fat line $\mathbf{L}$.

3.2.1 Geometric Intervals

We employ *hybrid Bézier curves* (see Section 2.5) to formulate the geometric interval for an arbitrary Bézier curve $\mathbf{P}(t)$, which we designate $[\mathbf{P}]_g$. Our goal is to transform all curves $\mathbf{P}(t)$ of degree $d \geq 2$ into a quadratic hybrid curve with a single moving control point and fixed end points. Once our curve is in quadratic form, we will be able to efficiently calculate the overlap interval of $[\mathbf{P}]_g$ and a fat line.

Section 2.5 contains a derivation of this transformation for a cubic Bézier curve. What follows is a derivation of the hybrid curve transformation for all curves.

Theorem 3.1. *Given a Bézier curve $\mathbf{P}(t)$ of degree $d \geq 2$, with control points $\mathbf{P}_i$, there exists an equivalent quadratic hybrid curve $\hat{\mathbf{P}}$ with fixed control points $\hat{\mathbf{P}}_0 = \mathbf{P}_0$ and $\hat{\mathbf{P}}_2 = \mathbf{P}_d$ and a moving control point $\hat{\mathbf{P}}_1(t)$. The moving control point $\hat{\mathbf{P}}_1(t)$ is itself a Bézier curve of degree $d-2$ with control points:*

$$\hat{\mathbf{P}}_{1,i-1} = \frac{a_i\mathbf{P}_0 + b_i\mathbf{P}_i + c_i\mathbf{P}_d}{a_i + b_i + c_i}, \tag{3.1}$$

where $a_i = (d-i)(1-(d-i))$, $b_i = d(d-1)$, $c_i = i(1-i)$ and $i \in \{1, \ldots, d-1\}$.

Proof. The diagonal curve $\mathbf{P}(t) = \mathbf{Q}(t,t)$ of a degree $m \times n$ Bézier surface $\mathbf{Q}(s,t)$ is a degree $m+n$ Bézier curve with control points

$$\mathbf{P}_i = \frac{1}{\binom{m+n}{i}} \sum_{j+k=i} \binom{m}{j}\binom{n}{k}\mathbf{Q}_{j,k}. \tag{3.2}$$

If we set $m = 2$ then $j \in \{0,1,2\}$ and $(j,k) \in \{(0,i),(1,i-1),(2,i-2)\}$. We may therefore expand the summation and rearrange terms to obtain

$$\binom{n+2}{i}\mathbf{P}_i = \binom{2}{0}\binom{n}{i}\mathbf{Q}_{0,i} + \binom{2}{1}\binom{n}{i-1}\mathbf{Q}_{1,i-1} + \binom{2}{2}\binom{n}{i-2}\mathbf{Q}_{2,i-2}. \tag{3.3}$$

If the control points $\mathbf{P}_i$ of a degree $d = n+2$ diagonal curve $\mathbf{P}(t)$ are known, we may set all $\mathbf{Q}_{0,i} = \mathbf{P}_0$ and all $\mathbf{Q}_{2,i-2} = \mathbf{P}_d$. This allows us to solve for the control points $\mathbf{Q}_{1,i-1}$ of $\mathbf{Q}(s,t)$, for which $\mathbf{P}(t)$ is the diagonal curve:

$$\binom{d}{i}\mathbf{P}_i = \binom{d-2}{i}\mathbf{P}_0 + 2\binom{d-2}{i-1}\mathbf{Q}_{1,i-1} + \binom{d-2}{i-2}\mathbf{P}_d, \tag{3.4}$$

$$\begin{aligned}\mathbf{Q}_{1,i-1} &= \frac{1}{2\binom{d-2}{i-1}}\left[\binom{d}{i}\mathbf{P}_i - \binom{d-2}{i}\mathbf{P}_0 - \binom{d-2}{i-2}\mathbf{P}_d\right] \\ &= \frac{(d-i)(1-(d-i))}{2i(d-i)}\mathbf{P}_0 + \frac{d(d-1)}{2i(d-i)}\mathbf{P}_i + \frac{i(1-i)}{2i(d-i)}\mathbf{P}_d,\end{aligned} \tag{3.5}$$

for $i \in \{1,\ldots,d-1\}$. Setting $a_i = (d-i)(1-(d-i))$, $b_i = d(d-1)$ and $c_i = i(1-i)$, and observing that $a_i + b_i + c_i = 2i(d-i)$, we have

$$\mathbf{Q}_{1,i-1} = \frac{a_i\mathbf{P}_0 + b_i\mathbf{P}_i + c_i\mathbf{P}_d}{a_i + b_i + c_i}. \tag{3.6}$$

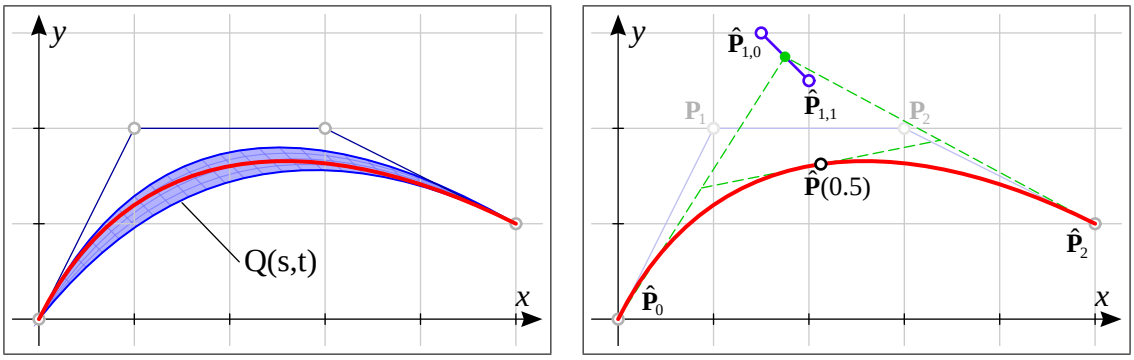
Since $\mathbf{Q}_{0,k}$ has been collapsed to $\mathbf{P}_0$, and since $\mathbf{Q}_{2,k}$ has been set to $\mathbf{P}_d$, we may therefore evaluate the $s = t$ diagonal curve of $\mathbf{Q}(s,t)$ using the following formula:

$$\mathbf{Q}(t,t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\hat{\mathbf{P}}_1(t) + t^2\mathbf{P}_d, \tag{3.7}$$

where $\hat{\mathbf{P}}_1(t)$ is a degree $d-2$ Bézier curve made up of the control points $\hat{\mathbf{P}}_{1,i-1} = \mathbf{Q}_{1,i-1}$ for $i \in \{1,\ldots,d-1\}$. This is simply a quadratic hybrid Bézier curve with a moving control point $\hat{\mathbf{P}}_1(t)$ and fixed control points $\mathbf{P}_0$ and $\mathbf{P}_d$. □

In the proof for Theorem 3.1, the Bézier curve $\mathbf{P}(t)$ is equated to the diagonal curve of a Bézier surface $\mathbf{Q}(s,t)$ as an intermediate step. We then derive a hybrid Bézier curve $\hat{\mathbf{P}}(t)$, which is equivalent to the diagonal curve $\mathbf{P}(t)$, from the control points of $\mathbf{Q}(s,t)$. While it is possible to avoid $\mathbf{Q}(s,t)$ and derive $\hat{\mathbf{P}}(t)$ algebraically, using only the control points of $\mathbf{P}(t)$, it is often convenient to think of the Bézier curve $\mathbf{P}(t)$ as the diagonal curve of the surface $\mathbf{Q}(s,t)$. For example, the the surface

patch forms a natural bounding area for the Bézier curve, as shown in Figure 3.2a. Figure 3.2b illustrates the equivalent hybrid curve $\hat{\mathbf{P}}(t)$, evaluated at $t = 0.5$.



(a) $\mathbf{Q}(s,t)$ encloses the curve since $\mathbf{P}(t) = \mathbf{Q}(t,t)$. (b) Hybrid curve $\hat{\mathbf{P}}(t)$, evaluated at $t = 0.5$.

Figure 3.2: $[\mathbf{P}]_g$ represented as the surface $\mathbf{Q}(s,t)$ and hybrid curve $\hat{\mathbf{P}}(t)$.

3.2.2 Fat Lines

Fat lines, which we borrow from the BezClip algorithm, are another means of bounding a Bézier curve. We briefly review their construction here.

The line $\overline{L}$ passes through the end points $\mathbf{P}_0$ and $\mathbf{P}_d$ of a degree d Bézier curve $\mathbf{P}(t)$. If we represent $\overline{L}$ with the implicit equation

$$\delta(x,y) = ax + by + c = 0, \tag{3.8}$$

then $\delta(x,y)$ is the signed distance from $\overline{L}$ to any point (x,y) scaled by $\sqrt{a^2+b^2}$. We define the fat line containing the curve $\mathbf{P}(t)$ and its control points as

$$\mathbf{L_P} = \{(x,y) \mid \delta(x,y) \in [\delta_{\min}, \delta_{\max}]\}, \tag{3.9}$$

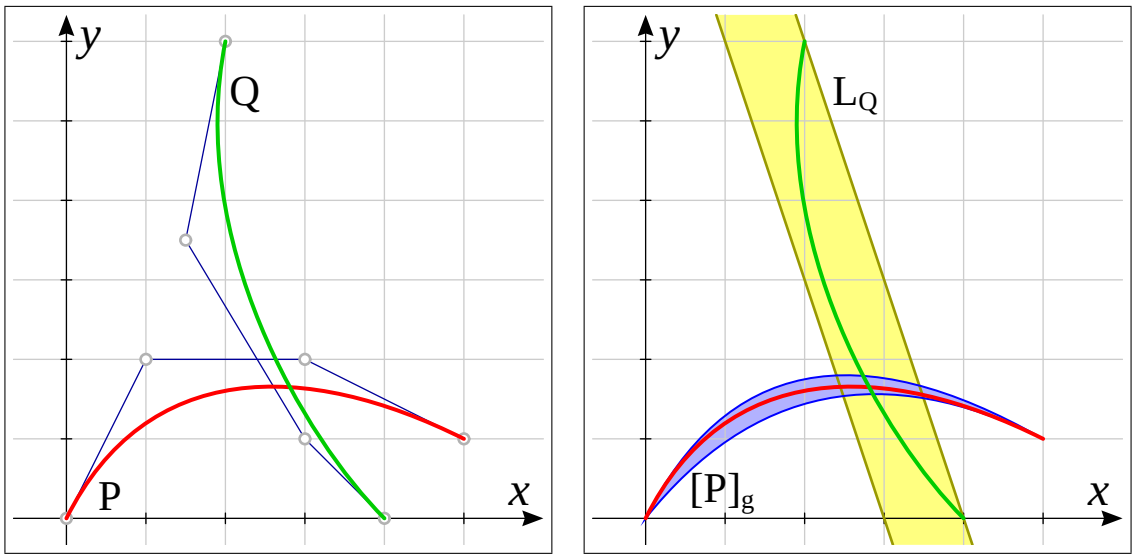
$$\text{where} \quad [\delta_{\min}, \delta_{\max}] = [\min_{0 \le i \le d} \delta(\mathbf{P}_i), \max_{0 \le i \le d} \delta(\mathbf{P}_i)]. \tag{3.10}$$

A fat line defined in this manner is the region of space between the two lines parallel to $\overline{L}$ which most tightly enclose the Bézier curve control points. By the convex hull

property, this region bounds $\mathbf{P}(t)$. For example, Figure 3.1 shows a fat line which bounds a quartic Bézier curve. Methods for obtaining tighter bounds for polynomial quadratic and cubic curves are presented in [27].

3.2.3 Geometric Interval Clipping

The two polynomial cubic Bézier curves $\mathbf{P}(t)$ and $\mathbf{Q}(u)$ in Figure 3.3a are shown in Figure 3.3b bounded by a geometric interval, designated $[\mathbf{P}]_g$, and the fat line labeled $\mathbf{L_Q}$, respectively. In this section we discuss how to identify intervals of t for which $\mathbf{P}(t)$ lies outside of $\mathbf{L_Q}$. These intervals correspond to regions of $\mathbf{P}(t)$ which cannot intersect $\mathbf{Q}(u)$.



(a) Curves $\mathbf{P}$ and $\mathbf{Q}$, to be intersected.

(b) Geometric interval $[\mathbf{P}]_g$ and fat line $\mathbf{L_Q}$.

Figure 3.3: Intersection of the Geometric interval $[\mathbf{P}]_g$ and the fat line $\mathbf{L_Q}$.

$[\mathbf{P}]_g$ is defined by the parametric equation for the hybrid Bézier curve $\hat{\mathbf{P}}(t)$:

$$\hat{\mathbf{P}}(t) = (1-t)^2\hat{\mathbf{P}}_0 + 2t(1-t)\hat{\mathbf{P}}_1(t) + t^2\hat{\mathbf{P}}_2, \tag{3.11}$$

where $\hat{\mathbf{P}}_0$, $\hat{\mathbf{P}}_1(t)$ and $\hat{\mathbf{P}}_2$ are defined as in Theorem 3.1. We may measure the signed, scaled distance from $\overline{L}$ to $\hat{\mathbf{P}}(t)$ by substituting Equation 3.11 into Equation 3.8:

$$\hat{\delta}(t) = (1-t)^2\hat{\delta}_0 + 2t(1-t)\hat{\delta}_1(t) + t^2\hat{\delta}_2, \qquad \hat{\delta}_i = \delta(\hat{\mathbf{P}}_i). \tag{3.12}$$

Note that $\hat{\mathbf{P}}_1(t)$ is the moving control point of the hybrid curve. $\hat{\delta}_1(t)$ is found by applying Equation 3.8 to the control points of $\hat{\mathbf{P}}_1(t)$ from Theorem 3.1:

$$\hat{\delta}_1(t) = \delta(\hat{\mathbf{P}}_1(t)) = \sum_{j=0}^{d-2} \delta(\hat{\mathbf{P}}_{1,j}) B_j^{d-2}(t), \tag{3.13}$$

where $B_j^{d-2}(t)$ is the Bernstein basis function (see Equation 2.16).

$\hat{\delta}(t)$ is a quadratic polynomial in the Bernstein basis. It is also a *hybrid polynomial* since $\hat{\delta}_1(t)$ is a function of t. Defining $[\hat{\delta}_1]$ as the interval containing the coefficients $\delta(\hat{\mathbf{P}}_{1,j})$ of $\hat{\delta}_1(t)$, we bound $\hat{\delta}(t)$ by an interval Bernstein polynomial:

$$[\hat{\delta}](t) = (1-t)^2\hat{\delta}_0 + 2t(1-t)[\hat{\delta}_1] + t^2\hat{\delta}_2, \tag{3.14}$$

$$[\hat{\delta}_1] = [\hat{\delta}_{1,\min}, \hat{\delta}_{1,\max}] = [\min_{0\le j\le d-2} \delta(\hat{\mathbf{P}}_{1,j}), \max_{0\le j\le d-2} \delta(\hat{\mathbf{P}}_{1,j})]. \tag{3.15}$$

Interval Bernstein polynomials (see Section 2.4) provide a convenient way to handle the variability of the moving control point $\hat{\delta}_1(t)$. Geometrically, $[\hat{\delta}](t)$ is a bound on the signed, scaled distance of $\mathbf{P}(t)$ to $\overline{L}$. The lower and upper bounds of $[\hat{\delta}](t)$ are defined by the quadratic Bernstein polynomials

$$\hat{\delta}_{\min}(t) = (1-t)^2\hat{\delta}_0 + 2t(1-t)\hat{\delta}_{1,\min} + t^2\hat{\delta}_2, \tag{3.16}$$

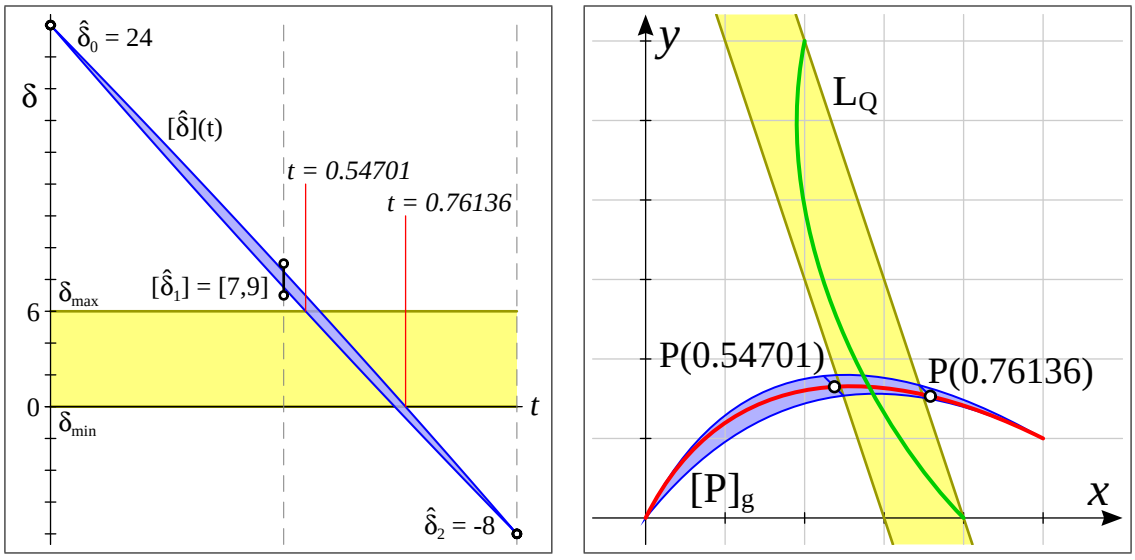
$$\hat{\delta}_{\max}(t) = (1-t)^2\hat{\delta}_0 + 2t(1-t)\hat{\delta}_{1,\max} + t^2\hat{\delta}_2. \tag{3.17}$$

The intervals of t for which $\mathbf{P}(t)$ lies outside of $\mathbf{L_Q}$ correspond to regions where $[\hat{\delta}](t) \cap [\delta_{\min}, \delta_{\max}] = \emptyset$. The end points for these intervals are made up of $t = 0$

(if $\delta_0 \in [\delta_{\min}, \delta_{\max}]$), $t = 1$ (if $\delta_2 \in [\delta_{\min}, \delta_{\max}]$) and the values $t \in [0, 1]$ for which $\hat{\delta}_{\min}(t)$ and $\hat{\delta}_{\max}(t)$ cross $\delta_{\min}$ and $\delta_{\max}$, corresponding to the roots of the equations

$$\begin{aligned} \hat{\delta}_{\min}(t) &= \delta_{\min}, & \hat{\delta}_{\min}(t) &= \delta_{\max}, \\ \hat{\delta}_{\max}(t) &= \delta_{\min}, & \hat{\delta}_{\max}(t) &= \delta_{\max}. \end{aligned} \tag{3.18}$$

Since $\hat{\delta}_{\min}(t)$ and $\hat{\delta}_{\max}(t)$ are quadratic, we may solve for these roots directly by using the quadratic formula. In practice, we have found that pseudo-conversion from the Bernstein basis to the power basis [20], combined with a numerically stable quadratic equation solver [4], is an efficient method to obtain a high quality result.



(a) $[\hat{\delta}](t)$ crossing $[\delta_{\min}, \delta_{\max}] = [0, 6]$.

(b) $\mathbf{P}(t)$ for $t \in [0.54701, 0.76136]$ is retained.

Figure 3.4: $\mathbf{P}(t) \neq \mathbf{Q}(u)$ when $[\hat{\delta}](t) < \delta_{\min}$ or $[\hat{\delta}](t) > \delta_{\max}$.

Figure 3.4a illustrates the arrangment of $[\hat{\delta}](t)$ and $[\delta_{\min}, \delta_{\max}]$ corresponding to Figure 3.3b. In this example, $\hat{\delta}_{\min}(0.54701) = \delta_{\max}$ and $\hat{\delta}_{\max}(0.76136) = \delta_{\min}$. We can be certain that $\mathbf{P}(t) \neq \mathbf{Q}(u)$ for $t < 0.54701$ and for $t > 0.76136$. This step of the GeoClip algorithm ends by subdividing the curve $\mathbf{P}(t)$ twice using the de Casteljau algorithm, clipping all portions of the curve away which lie outside the parameter interval $[0.54701, 0.76136]$. Figure 3.4b shows the clipping points.

Rational Curves

Up to this point we have assumed that $\mathbf{P}$ is a polynomial Bézier curve. One of the original purposes of hybrid curves was to approximate a rational curve using a hybrid polynomial curve [26]. It is possible, therefore, to use a polynomial hybrid curve to represent a rational curve $\mathbf{P}$. However, it is likely that our clipping bounds will be narrower if we handle rational curves directly. We now derive the rational case.

If $\mathbf{P}$ is rational, Equation 3.11 becomes

$$\hat{\mathbf{P}}(t) = \frac{(1-t)^2\hat{w}_0\hat{\mathbf{P}}_0 + 2t(1-t)\hat{w}_1(t)\hat{\mathbf{P}}_1(t) + t^2\hat{w}_2\hat{\mathbf{P}}_2}{(1-t)^2\hat{w}_0 + 2t(1-t)\hat{w}_1(t) + t^2\hat{w}_2} \tag{3.19}$$

where $\hat{\mathbf{P}}_0$, $\hat{\mathbf{P}}_1(t)$ and $\hat{\mathbf{P}}_2$ and their corresponding weights $\hat{w}_0$, $\hat{w}_1(t)$ and $\hat{w}_2$ are defined by applying Theorem 3.1 to the control points and weights of a rational curve $\mathbf{P}$. As with BezClip, when $\mathbf{P}$ is rational we *cannot* represent the intersection of $\hat{\mathbf{P}}(t)$ and $\mathbf{L_Q}$ as $\{(x,y) = \hat{\mathbf{P}}(t) \mid \hat{\delta}(t) \in [\delta_{\min}, \delta_{\max}]\}$. Instead, we have

$$\{(x,y) = \hat{\mathbf{P}}(t) \mid 0 \in [\hat{\delta}(t,\delta_{\min}), \hat{\delta}(t,\delta_{\max})]\}, \tag{3.20}$$

where $\hat{\delta}(t,\delta_{\min}) = 0$ and $\hat{\delta}(t,\delta_{\max}) = 0$ define the intersection between $\hat{\mathbf{P}}(t)$ and the lines bounding $\mathbf{L_Q}$. We derive $\hat{\delta}(t,\alpha)$, where $\alpha \in \{\delta_{\min}, \delta_{\max}\}$, by substituting Equation 3.19 into Equation 3.8 and clearing the denominator:

$$\hat{\delta}(t,\alpha) = (1-t)^2\hat{\delta}_0(\alpha) + 2t(1-t)\hat{\delta}_1(t,\alpha) + t^2\hat{\delta}_2(\alpha), \qquad \hat{\delta}_i(\alpha) = \hat{w}_i(\delta(\hat{\mathbf{P}}_i) - \alpha). \tag{3.21}$$

Note that $\hat{\delta}(t,\alpha)$ is polynomial since we were able to clear the denominator. Therefore, the fact that $\mathbf{P}$ is rational does not add much complexity to the GeoClip algorithm. In fact, the only significant added complexity is a side effect of clearing the denominator: a rational curve $\mathbf{P}$ must be clipped independently against each of the boundary lines of $\mathbf{L_Q}$.

Although this process is essentially the same as previously outlined, we briefly review it here with proper substitutions for the rational case. Bounding $\hat{\delta}(t,\alpha)$ with an interval Bernstein polynomial, Equations 3.14 and 3.15 become

$$[\hat{\delta}](t,\alpha) = (1-t)^2\hat{\delta}_0(\alpha) + 2t(1-t)[\hat{\delta}_1(\alpha)] + t^2\hat{\delta}_2(\alpha), \tag{3.22}$$

$$[\hat{\delta}_1(\alpha)] = [\min_{0\le j\le d-2} \hat{w}_{1,j}(\delta(\hat{\mathbf{P}}_{1,j}) - \alpha), \max_{0\le j\le d-2} \hat{w}_{1,j}(\delta(\hat{\mathbf{P}}_{1,j}) - \alpha)]. \tag{3.23}$$

As with Equations 3.16 and 3.17, we derive bounding polynomials from $[\hat{\delta}](t,\alpha)$:

$$\hat{\delta}_{\min}(t,\alpha) = (1-t)^2\hat{\delta}_0(\alpha) + 2t(1-t)\hat{\delta}_{1,\min}(\alpha) + t^2\hat{\delta}_2(\alpha), \tag{3.24}$$

$$\hat{\delta}_{\max}(t,\alpha) = (1-t)^2\hat{\delta}_0(\alpha) + 2t(1-t)\hat{\delta}_{1,\max}(\alpha) + t^2\hat{\delta}_2(\alpha). \tag{3.25}$$

The intersection of these bounding polynomials with the boundary lines of $\mathbf{L_Q}$, defined by $\alpha \in \{\delta_{\min}, \delta_{\max}\}$, correspond to the roots of the equations

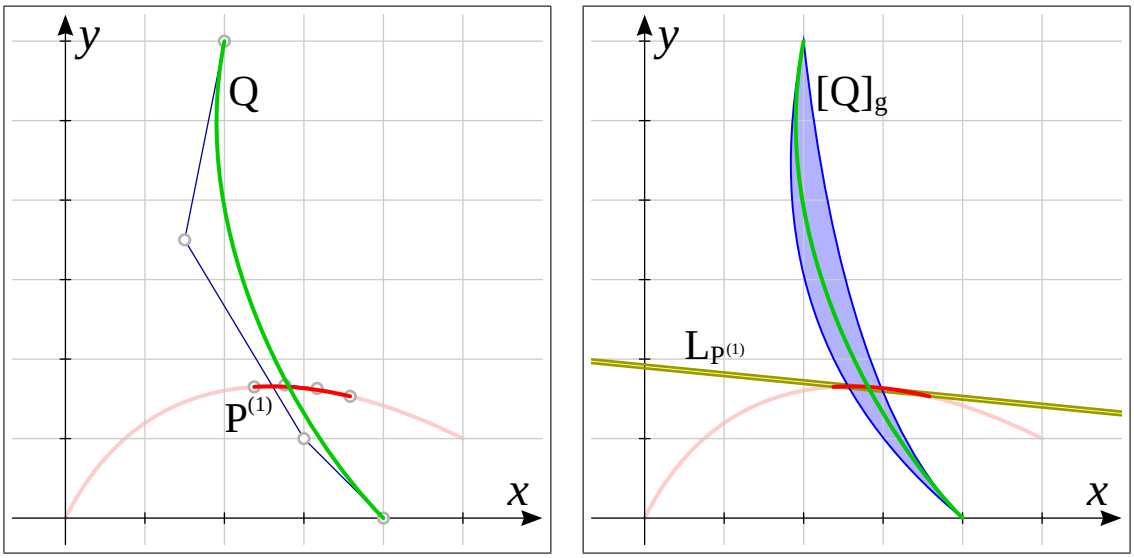
$$\begin{aligned} \hat{\delta}_{\min}(t,\delta_{\min}) &= 0, & \hat{\delta}_{\min}(t,\delta_{\max}) &= 0, \\ \hat{\delta}_{\max}(t,\delta_{\min}) &= 0, & \hat{\delta}_{\max}(t,\delta_{\max}) &= 0. \end{aligned} \tag{3.26}$$

These roots, restricted to the parameter range $t \in [0,1]$, along with the end points $t = 0$ and $t = 1$, are sufficient to define the intervals over which $\hat{\mathbf{P}}(t)$ intersects $\mathbf{L_Q}$.

3.2.4 Iterating

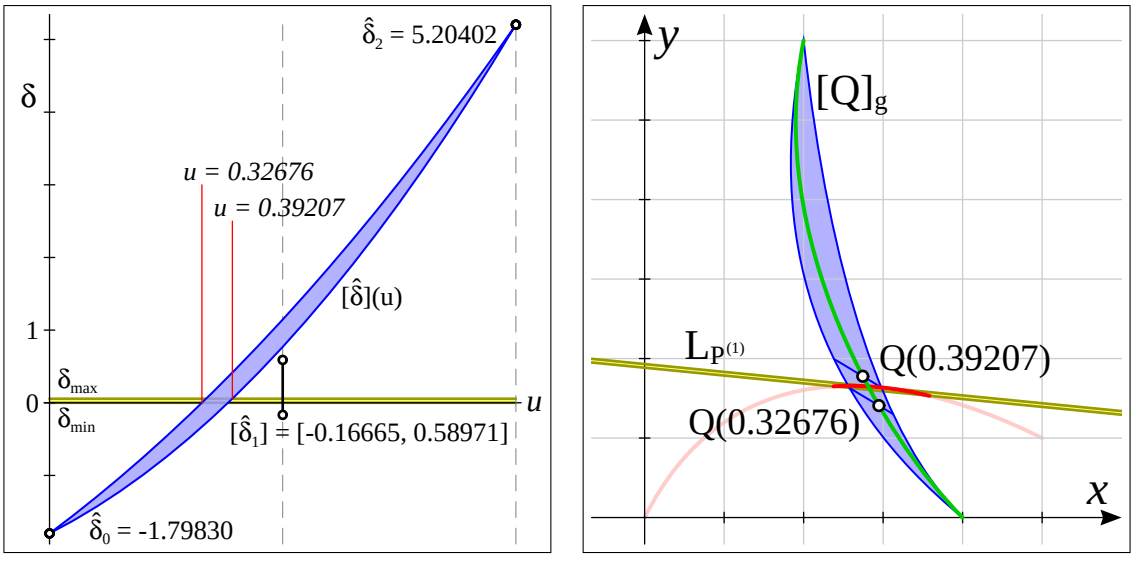
We have shown how to use the geometric interval clipping technique to clip away regions where two curves are guaranteed not to intersect. In our example, it was determined that $\mathbf{P}(t)$ did not intersect $\mathbf{Q}(u)$ outside the interval $t \in [0.54701, 0.76136]$. Figure 3.5a illustrates curves $\mathbf{P}$ and $\mathbf{Q}$ from Figure 3.3 after the first clipping step of GeoClip. $\mathbf{P}^{(1)}$ is the portion of curve $\mathbf{P}$ remaining after subdivision. The next

step is to determine where curve $\mathbf{Q}$ overlaps $\mathbf{P}^{(1)}$. We now address the process of iteratively applying geometric interval clipping to further isolate intersections.



(a) Curves $\mathbf{P}^{(1)}$ and $\mathbf{Q}$, to be intersected. (b) Geometric interval $[\mathbf{Q}]_g$ and fat line $\mathbf{L}_{\mathbf{P}^{(1)}}$.

Figure 3.5: Intersection of the Geometric interval $[\mathbf{Q}]_g$ and the fat line $\mathbf{L}_{\mathbf{P}^{(1)}}$.



(a) $[\hat{\delta}](u)$ crossing $[\delta_{\min}, \delta_{\max}] = [0, 0.05588]$. (b) $\mathbf{Q}(u)$ for $u \in [0.32676, 0.39207]$ is retained.

Figure 3.6: $\mathbf{Q}(u) \neq \mathbf{P}(t)$ when $[\hat{\delta}](u) < \delta_{\min}$ or $[\hat{\delta}](u) > \delta_{\max}$.

To determine the regions of $\mathbf{Q}(u)$ which are safe to clip, we begin by forming a geometric interval $[Q]_g$ to bound curve $\mathbf{Q}$ and a fat line $\mathbf{L}_{\mathbf{P}^{(1)}}$ to enclose $\mathbf{P}^{(1)}$, as shown in Figure 3.5b. We may now apply the geometric interval clipping technique to determine the overlap of $[Q]_g$ and $\mathbf{L}_{\mathbf{P}^{(1)}}$.

Figure 3.6a shows that interval Bernstein polynomial $[\hat{\delta}](u)$ which represents the signed, scaled distance from $\overline{L}_{\mathbf{P}^{(1)}}$ to $[Q]_g$. The overlap of $[\hat{\delta}](u)$ with the interval $\delta = [\delta_{\min}, \delta_{\max}]$ is calculated and it is determined that $\mathbf{Q}(u)$ may be safely clipped for $u < 0.32676$ and $u > 0.39207$. A subcurve $\mathbf{Q}^{(1)}$ is formed by subdividing $\mathbf{Q}(u)$ to remove these regions. The clipping points $Q(0.32676)$ and $Q(0.39207)$, which clearly lie on either side of the fat line $\mathbf{L}_{\mathbf{P}^{(1)}}$, are shown in Figure 3.6b.

i	$\mathbf{P}$ *Intersection Interval* $[t_0^i, t_1^i]$	*Width of* $[t_0^i, t_1^i]$
0	$[0, 1]$	1.0000000×10^{0}
1	$[0.547013139706567, 0.761355820929153]$	2.1434268×10^{-1}
2	$[0.626365384338657, 0.627118468877733]$	7.5308454×10^{-4}
3	$[0.626551181798084, 0.626551181835295]$	$3.7211234 \times 10^{-11}$

(a) Refinement of the interval $[t_0, t_1]$ which bounds potential intersections on $\mathbf{P}$.

i	$\mathbf{Q}$ *Intersection Interval* $[u_0^i, u_1^i]$	*Width of* $[u_0^i, u_1^i]$
0	$[0, 1]$	1.0000000×10^{0}
1	$[0.326756310053875, 0.392066137230932]$	6.5309827×10^{-2}
2	$[0.369959987612443, 0.369975295646327]$	1.5308034×10^{-5}
3	$[0.369965211165735, 0.369965211165736]$	$2.2204460 \times 10^{-16}$

(b) Refinement of the interval $[u_0, u_1]$ which bounds potential intersections on $\mathbf{Q}$.

Table 3.1: Convergence of intersection intervals for curves in Figure 3.3.

The process continues by clipping $\mathbf{P}^{(1)}(t)$ against $\mathbf{Q}^{(1)}(u)$, and so on. The details of subsequent iterations are shown in Table 3.1. In this example, the two curves have a single intersection, $\mathbf{P}(t^*) = \mathbf{Q}(u^*)$. For each iteration i shown in Table 3.1, $t^* \in [t_0^i, t_1^i]$ and $u^* \in [u_0^i, u_1^i]$. By the fourth iteration using the GeoClip technique, the parameter values of the intersection point have been approximated with over 16 digits of accuracy.

3.2.5 Multiple Intersections

So far we have only discussed the details of applying the geometric interval clipping technique to locate a single point of intersection between two Bézier curves. To complete our discussion of the GeoClip algorithm, we will now show how to isolate multiple intersections.

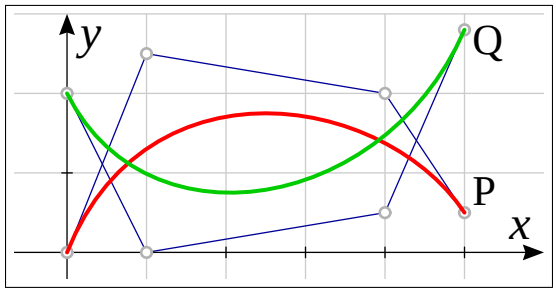


Figure 3.7: Two intersections between curves **P** and **Q**.

Figure 3.7 illustrates curves **P** and **Q** which intersect at two points. To help isolate these intersections, so that there is only one intersection between subcurves, we will use a modification of the heuristic employed by BezClip. The BezClip heuristic states that if the remaining parameter interval of either curve is not reduced by at least 20% during the clipping procedure, the curve with largest remaining interval is split in half. We then independently intersect each of the two curve halves with the curve which was not subdivided. Application of this rule allows BezClip and GeoClip to reliably locate all intersections. Figure 3.8 shows how subdividing one curve can isolate multiple intersections.

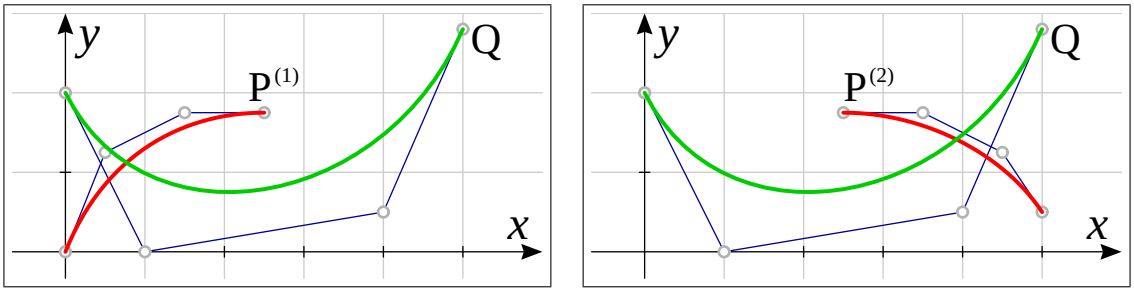


Figure 3.8: Each intersection is isolated after splitting curve **P** in half.

Experience has shown that a modest reduction in execution time is possible by using the following modified heuristic. Instead of checking the interval reduction of both curves and then, if necessary, subdividing the curve with the largest remaining interval, we split a curve as soon as it is determined that interval reduction will not exceed our threshold. This accelerates the intersection process by allowing the next clipping step to be performed against the shortest possible curve segment. Additionally, when using this approach, our experiments suggest that it is best to divide a curve in half when less than 30% of a curve's parameter length is removed by the clipping step, rather than 20% as with the original heuristic.

3.3 Timing Comparisons

An implementation of the GeoClip algorithm has been compared to implementations of the BezClip and Impl algorithms. The BezClip algorithm was chosen for comparison due to the fact that GeoClip aims to improve BezClip by extending the algorithm with geometric intervals. Impl was chosen because it appears to be the fastest curve/curve intersection algorithm for low-degree Bézier curves [27, 28].

Note that Impl requires that we identify any degree-elevated curve, such as a quadratic curve that has been represented as a cubic, and degree-reduce the curve before performing implicitization. Rather than taking this extra step, special care was taken to eliminate such curves before testing was performed. Also, timing results for Impl are omitted for curves of degree higher than five. For high-degree curves, Impl is clearly the slowest of the three algorithms and tremendous care must be taken to minimize numerical instability, rendering Impl impractical.

Algorithms such as Subdiv and KopInt were not used for testing. Other sources [27] have shown that these algorithms are slower than BezClip. The algebraic pruning technique [15] was not included as its performance is reportedly comparable to BezClip in simple cases.

The first set of tests was designed to give an indication of how the Impl, BezClip and GeoClip algorithms perform relative to each other when applied to curves of various degree. Sets of 1,000 curves were generated by randomly selecting control points in the unit square. Then all intersections between every combination of curves in the set were calculated using the available algorithms. This process was repeated at least three times for each degree and the results were averaged together. Relative execution times for these tests are reported in Table 3.2.

Degree	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Impl	1.00	1.00	1.35	3.42	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*
BezClip	2.12	1.66	1.43	1.44	1.46	1.49	1.53	1.58	1.62
GeoClip	1.17	1.12	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 3.2: Relative computation times for polynomial Bézier curve intersection.

While it is clear that timing results can change somewhat due to many factors, such as implementation details, optimizations, compilers and the host machine, these results convey the general relative performance of each algorithm. One implementation detail that should be noted is the use of the Bcom polynomial root-finding algorithm from [29] for finding roots in the Impl algorithm. All tests were run on a 1.33 GHz Apple PowerBook G4. Intersections were computed to 14 digits of accuracy using double-precision arithmetic.

Degree	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Impl	1.00	1.00	1.26	2.89	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*
BezClip	2.78	1.98	1.52	1.51	1.50	1.48	1.50	1.51	1.51
GeoClip	1.39	1.21	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 3.3: Relative computation times for rational Bézier curve intersection.

Table 3.2 indicates that GeoClip is at least 30% faster than Impl and BezClip for degree four and higher. While GeoClip is significantly faster than BezClip for all degrees, the Impl algorithm is still faster for degrees two and three. Table 3.3 shows similar results for rational curves. Sets of rational Bézier curves were generated

in the same manner as before, with weights being randomly chosen in the interval $[0, 10]$ for each control point. Note that in both Table 3.2 and Table 3.3 Impl could not be applied to curves of degree six and higher.

Randomly generated curves are useful for giving a good, overall feel for the performance of these algorithms without making many assumptions about the form the curves will take. However, we would expect that curves designed for use by people would have lower curvature, rarely self-intersect and, being simpler, intersect other curves in fewer places when compared to automatically generated curves. While GeoClip is competitive with Impl for quadratic and cubic curves, we have also tested intersection performance using polynomial cubic curves created by hand to see if we can expect any improvement when applying GeoClip to real-world data.

Table 3.4 shows relative execution times for calculating all intersections between sets of 5,000 human-designed cubic polynomial Bézier curves. These curves were extracted from the glyph definitions of 36 standard PostScript fonts, including some fonts that contain symbols and icons. The curves were then normalized by centering them within the unit square, taking care to retain proportions. Just over 20,000 unique curves were obtained which were distributed into four data sets of 5,000 curves each, labeled *A*, *B*, *C* and *D*.

Data Set	*A*	*B*	*C*	*D*	*Total*
Impl	1.005	1.014	1.022	1.000	1.005
BezClip	1.661	1.624	1.628	1.497	1.593
GeoClip	1.000	1.000	1.000	1.021	1.000

Table 3.4: Relative computation time for cubic Bézier curves, by data set.

When testing with these data sets the small difference in speed between Impl and GeoClip becomes insignificant. This is not unexpected when we consider that Impl simply converts every intersection problem to a root-finding problem. There is little difference in the work required to find a single point of intersection and multiple

intersections. GeoClip, on the other hand, will more efficiently clip away portions of each curve when the curves can be more precisely bounded by the quadratic curves of a geometric interval and when there are fewer intersections.

Analysis of the data sets A, B, C and D shows that over 99% of the curve pairs result in only a single intersection. Table 3.5 shows the relative amount of time spent finding intersections between curves in these data sets separated into groups of zero to three intersections. Too few pairings of curves resulted in four or more intersections to accurately measure performance.

Intersections	*0*	*1*	*2*	*3*
Impl	1.000	1.008	1.000	1.000
BezClip	2.322	1.526	4.978	6.580
GeoClip	1.062	1.000	1.632	1.972

Table 3.5: Relative computation time for cubic Bézier curves, by intersection count.

Our experience is that the GeoClip algorithm is significantly faster than the other tested algorithms when applied to curves of degree four or higher. GeoClip is still competitive with Impl for quadratic and cubic curves and is always faster than BezClip. Furthermore, if it is expected that there will usually only be one point of intersection between a pair of curves, GeoClip may beat Impl. Difficulties with Impl, such as numerical stability issues and the need to check for degree elevated curves, make GeoClip an attractive choice as a general-purpose curve/curve intersection algorithm.

3.4 Observations and Conclusions

In this chapter we have presented the details of the GeoClip curve/curve intersection algorithm. In Section 3.2 we showed that geometric intervals can be used to extend the design of BezClip [27] to create a simple, yet fast and robust algorithm. This algorithm, GeoClip, exhibits cubic convergence, improving on the observed

quadratic convergence of BezClip. Section 3.3 provides evidence that GeoClip is generally faster than BezClip for curves of any degree. GeoClip is also faster than Impl for degree four and higher and competitive for quadratic and cubic curves.

BezClip uses the convex hull of the curve as a kind of linear bound on the curve. The geometric interval we use in this chapter is formed by transforming a Bézier curve into a quadratic hybrid curve. Logically, the next step would appear to be to use some kind of cubic bound in conjunction with the Bézier clipping technique. We have experimented with this possibility and found that the added complexity of a cubic geometric interval outweighs any advantage. Timing tests suggest that a cubic bound would yield performance somewhere between GeoClip and BezClip, at best. This suggests that the choice of a quadratic geometric interval is a good balance between complexity and rate of convergence.

Geometric intervals provide a powerful method to exploit the geometry of Bézier curves for finding intersections. The same approach can be applied to ray/surface intersection to similar effect, as shown in Chapter 4. GeoClip could also easily be extended to become a general-purpose polynomial root finder. Such a system would bear some similarity to [3], but our geometric interval method would be employed to form quadratic bounds for determining clipping ranges, which may provide some unique advantages.

Chapter 4

Ray/surface Intersection

Chapter 3 describes an algorithm for locating points of intersection between two Bézier curves. This algorithm, labeled GeoClip, utilizes geometric intervals to isolate curve regions which can be safely discarded; showing that they do not participate in an intersection. This chapter presents an extension to this approach for finding ray/surface intersections. That is, we will show how to use geometric intervals to isolate points where a ray intersects a Bézier surface patch. We will continue to use the label GeoClip for the ray/surface version of the geometric interval clipping algorithm, distinguishing between the two algorithms by context.

Section 4.1 reviews several existing algorithms for solving the ray/surface intersection problem. Section 4.2 presents the ray/surface GeoClip algorithm in detail. Section 4.3 provides timing comparisons between GeoClip and a few other algorithms. Section 4.4 is dedicated to observations and conclusions.

4.1 Previous Work

Several algorithms for ray/surface intersection, including [32, 33], utilize subdivision techniques. A ray is tested against the convex hull of a Bézier patch and, if they intersect, the patch is subdivided. The process of recursively subdividing the surface and testing sub-surface convex hulls against the ray continues until points of intersection have been approximated sufficiently well. This amounts to a binary search and,

therefore, convergence is linear. [33] improves upon [32] by mapping the problem to two dimensions, reducing the cost of using the de Casteljau algorithm to subdivide the Bézier patch.

There is also a wealth of literature that takes advantage of numerical methods to solve the ray/surface intersection problem. Toth [30] describes an algorithm that combines bounding volumes and an interval version of Newton's iteration to isolate safe starting positions for a standard Newton solver. The use of interval arithmetic allows Toth's algorithm to make guarantees about the ability of the standard Newton solver to converge on a solution. Barth and Stürzlinger [2] also use bounding volumes and Newton's method, but the robustness of interval arithmetic is abandoned for an increase in speed. Their method makes no convergence guarantees, but, in practice, intersections are only rarely missed. Martin et al. [16] brings together the work of many previous Newton's iteration-based numerical solvers to form one coherent system that is efficient and reliable.

Kajiya [12] takes an algebraic approach, mapping the ray/surface intersection problem to the task of finding the roots of a univariate polynomial. Kajiya's approach formulates the polynomial in the power basis, while other sources [23, 24] show that it is possible to keep the polynomial in the Bernstein basis, enhancing numerical stability. This approach parallels the Impl [28] curve/curve intersection algorithm discussed in Chapter 3, therefore we will also apply the Impl label to this algorithm.

Nishita et al. [19] extend the Bézier clipping technique [27] to ray/surface intersection. The label BezClip will be used to identify this ray/surface extension. BezClip borrows and improves upon a technique from [33], using two planes that intersect along the ray to project a Bézier patch to $\mathbb{R}^2$. The intersection of the ray with the patch corresponds to the location of the origin after projection. A line through the origin replaces the fat lines used in [27] and the surface is clipped against this line alternately in the s and t parameter directions, removing portions of the

surface that do not contain the origin. This process is repeated until the origin, and therefore the ray/surface intersection, has been located with sufficient precision. BezClip combines the simplicity of subdivision techniques with the efficiency of the numerical methods, making the algorithm a popular choice.

4.2 The GeoClip Ray/surface Intersection Algorithm

We extend the ray/surface BezClip technique to formulate our geometric interval clipping algorithm, GeoClip. Projection to $\mathbb{R}^2$ is done in the same manner and clipping against a line through the origin is retained as the mechanism for discarding portions of the surface that do not intersect the ray. The major contribution of GeoClip is the introduction of geometric intervals, which significantly improve the algorithm's ability to find regions of the surface which do not overlap the origin. Due to this enhancement, GeoClip generally exhibits *cubic* convergence without much added complexity, whereas BezClip is thought to converge quadratically.

Section 4.2.1 reviews the projection step defined in [19]. Section 4.2.2 introduces a method for forming geometric intervals for Bézier surface patches. Sections 4.2.3 and 4.2.4 describe the clipping process. Section 4.2.5 explains how to handle multiple intersections.

4.2.1 Projection to $\mathbb{R}^2$

The parametric equation

$$\tilde{\mathbf{P}}(s,t) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\tilde{\mathbf{P}}_{i,j}w_{i,j}B_i^n(s)B_j^m(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{i,j}B_i^n(s)B_j^m(t)} \tag{4.1}$$

defines a degree $n \times m$ rational Bézier surface patch $\tilde{\mathbf{P}}(s,t)$. For now, we shall concern ourselves with patches defined in Cartesian three-space, which have control points $\tilde{\mathbf{P}}_{i,j} = (\tilde{x}_{i,j}, \tilde{y}_{i,j}, \tilde{z}_{i,j})$ and corresponding weights $w_{i,j}$. The tilde above some

symbols is to distinguish them from the projected (x, y) space we will use later. See Section 2.3 for a review of Bézier surface patches.

A ray may be defined as the intersection of two planes with implicit equations

$$a_k\tilde{x} + b_k\tilde{y} + c_k\tilde{z} + d_k = 0, \quad k \in \{1, 2\}, \tag{4.2}$$

where (a_k, b_k, c_k) is normal to plane k and $d_k = -a_k\tilde{x} - b_k\tilde{y} - c_k\tilde{z}$ for any $(\tilde{x}, \tilde{y}, \tilde{z})$ in the plane. Experience indicates that orthogonal planes work best.

Any point on $\tilde{\mathbf{P}}(s, t)$ that lies on both planes must also lie along the ray. The intersection of the patch and plane k is found by substituting Equation 4.1 into Equation 4.2 and clearing the denominator:

$$\rho^k(s, t) = \sum_{i=0}^{n}\sum_{j=0}^{m} \rho^k_{i,j} B^n_i(s) B^m_j(t) = 0, \qquad \rho^k_{i,j} = w_{i,j}(a_k\tilde{x} + b_k\tilde{y} + c_k\tilde{z} + d_k). \tag{4.3}$$

We now project $\tilde{\mathbf{P}}(s, t)$ to $\mathbb{R}^2$, forming a polynomial Bézier surface $\mathbf{P}(s, t)$ in Cartesian two-space:

$$\mathbf{P}(s, t) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{P}_{i,j} B^n_i(s) B^m_j(t), \qquad \mathbf{P}_{i,j} = (x_{i,j}, y_{i,j}) = (\rho^1_{i,j}, \rho^2_{i,j}). \tag{4.4}$$

An example projection is shown in Figure 4.1. Note that $\mathbf{P}$ is *always* polynomial, even if $\tilde{\mathbf{P}}$ is rational. If $\tilde{\mathbf{P}}$ is polynomial (all $w_{i,j} = 1$), and the planes intersecting along the ray are orthogonal, then $\mathbf{P}$ is a simple orthographic projection of $\tilde{\mathbf{P}}$ along the ray.

This transformation projects plane 1 to the y axis and plane 2 to the x axis. The ray is projected to the origin, $\mathbf{0}$. After projection, the ray/surface intersection problem amounts to finding

$$\{(s, t) \mid \mathbf{P}(s, t) = \mathbf{0};\ 0 \le s, t \le 1\}. \tag{4.5}$$

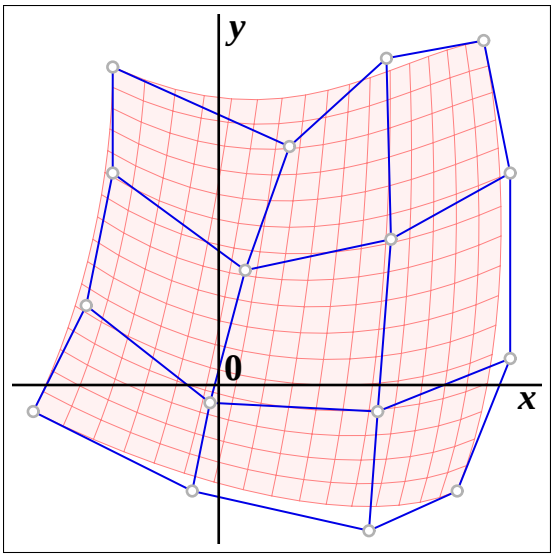


Figure 4.1: Bézier patch $\mathbf{P}$, the projection of $\tilde{\mathbf{P}}$.

4.2.2 Geometric Intervals

A geometric interval $[\mathbf{P}]_g$ for the surface $\mathbf{P}$ can be formed by transforming the patch into a *hybrid Bézier surface*. The hybrid surface is equivalent to $\mathbf{P}$. However, moving control points allow us to arbitrarily reformulate either the s or t parameter direction as quadratic. By recasting the surface in this manner we will be able to more efficiently determine which regions of the surface do not intersect the ray.

We first derive $\hat{\mathbf{P}}^{\mathbf{s}}$, a hybrid surface transforming the s parameter direction of $\mathbf{P}$ so that it is quadratic. Figure 4.2a illustrates $\hat{\mathbf{P}}^{\mathbf{s}}$ corresponding to Figure 4.1.

Theorem 4.1. *Given a Bézier surface* $\mathbf{P}(s,t)$ *of degree* $n \times m$, $n \geq 2$, *with control points* $\mathbf{P}_{i,j}$, $i \in \{0,\ldots,n\}$, $j \in \{0,\ldots,m\}$, *there exists an equivalent hybrid surface* $\hat{\mathbf{P}}^{\mathbf{s}}$ *with fixed control points* $\hat{\mathbf{P}}^{\mathbf{s}}_{0,j} = \mathbf{P}_{0,j}$ *and* $\hat{\mathbf{P}}^{\mathbf{s}}_{2,j} = \mathbf{P}_{n,j}$ *and moving control points* $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j}(s)$. $\hat{\mathbf{P}}^{\mathbf{s}}$ *is given by the equation*

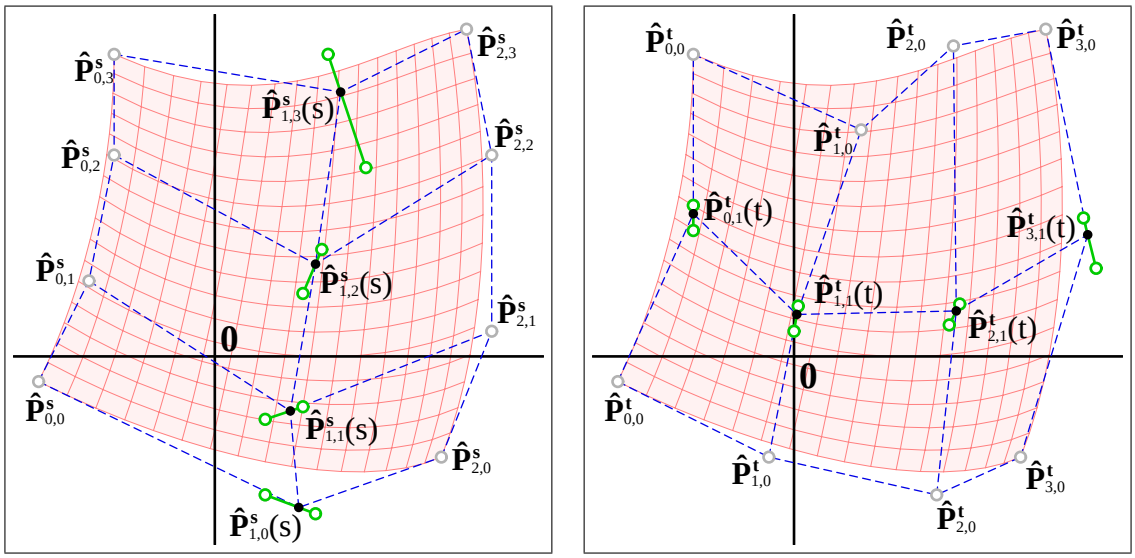
$$\hat{\mathbf{P}}^{\mathbf{s}}(s,t) = \sum_{j=0}^{m} B_j^m(t) \left((1-s)^2 \hat{\mathbf{P}}^{\mathbf{s}}_{0,j} + 2s(1-s)\hat{\mathbf{P}}^{\mathbf{s}}_{1,j}(s) + s^2 \hat{\mathbf{P}}^{\mathbf{s}}_{2,j} \right). \tag{4.6}$$

Each moving control point $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j}(s)$ *is a Bézier curve defined by control points* $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j,k}$, $k \in \{0, \ldots, n-2\}$. *The points* $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j,k}$ *are formed by applying Theorem 3.1 to row* j *of the control net belonging to* $\mathbf{P}$.

Proof. By regrouping the equation for $\mathbf{P}(s,t)$,

$$\mathbf{P}(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{P}_{i,j} B_i^n(s) B_j^m(t) = \sum_{j=0}^{m} B_j^m(t) \left(\sum_{i=0}^{n} \mathbf{P}_{i,j} B_i^n(s) \right), \tag{4.7}$$

we isolate the expression $\sum_{i=0}^{n} \mathbf{P}_{i,j} B_i^n(s)$, which denotes the Bézier curve made up of the control points along row j of the control net of $\mathbf{P}$. Applying Theorem 3.1 to this curve gives us Equation 4.6. □



(a) Hybrid surface $\hat{\mathbf{P}}^{\mathbf{s}}$ equivalent to Figure 4.1. (b) Hybrid surface $\hat{\mathbf{P}}^{\mathbf{t}}$ equivalent to Figure 4.1.

Figure 4.2: Application of Theorems 4.1 and 4.2 to a projected surface patch $\mathbf{P}(s,t)$.

$\mathbf{P}$ may also be transformed into the hybrid surface $\hat{\mathbf{P}}^{\mathbf{t}}$, which reformulates $\mathbf{P}$ as quadratic in the t parameter direction, as illustrated in Figure 4.2b.

Theorem 4.2. *Given a Bézier surface* $\mathbf{P}(s,t)$ *of degree* $n \times m$, $m \geq 2$, *with control points* $\mathbf{P}_{i,j}$, $i \in \{0, \ldots, n\}$, $j \in \{0, \ldots, m\}$, *there exists an equivalent hybrid surface*

$\hat{\mathbf{P}}^{\mathbf{t}}$ *with fixed control points* $\hat{\mathbf{P}}^{\mathbf{t}}_{i,0} = \mathbf{P}_{i,0}$ *and* $\hat{\mathbf{P}}^{\mathbf{t}}_{i,2} = \mathbf{P}_{i,m}$ *and moving control points* $\hat{\mathbf{P}}^{\mathbf{t}}_{i,1}(t)$. $\hat{\mathbf{P}}^{\mathbf{t}}$ *is given by the equation*

$$\hat{\mathbf{P}}^{\mathbf{t}}(s,t) = \sum_{i=0}^{n} B_i^n(s) \left((1-t)^2 \hat{\mathbf{P}}^{\mathbf{t}}_{i,0} + 2t(1-t)\hat{\mathbf{P}}^{\mathbf{t}}_{i,1}(t) + t^2 \hat{\mathbf{P}}^{\mathbf{t}}_{i,2} \right) \tag{4.8}$$

Each moving control point $\hat{\mathbf{P}}^{\mathbf{t}}_{i,1}(t)$ *is a Bézier curve defined by control points* $\hat{\mathbf{P}}^{\mathbf{t}}_{i,1,k}$, $k \in \{0, \ldots, m-2\}$. *The points* $\hat{\mathbf{P}}^{\mathbf{t}}_{i,1,k}$ *are formed by applying Theorem 3.1 to column* i *of the control net belonging to* $\mathbf{P}$.

Proof. Mirroring the approach taken in the proof of Theorem 4.1, we regroup the equation for $\mathbf{P}(s,t)$:

$$\mathbf{P}(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{P}_{i,j} B_i^n(s) B_j^m(t) = \sum_{i=0}^{n} B_i^n(s) \left(\sum_{j=0}^{m} \mathbf{P}_{i,j} B_j^m(t) \right). \tag{4.9}$$

The isolated expression, $\sum_{j=0}^{m} \mathbf{P}_{i,j} B_j^m(t)$, corresponds to the Bézier curve defined by the control points along column i of the control net of $\mathbf{P}$. Applying Theorem 3.1 to this curve gives us Equation 4.8. □

$[\mathbf{P}]_g$ may be defined by either $\hat{\mathbf{P}}^{\mathbf{s}}(s,t)$ or $\hat{\mathbf{P}}^{\mathbf{t}}(s,t)$, denoted $[\mathbf{P}]^s_g$ or $[\mathbf{P}]^t_g$, respectively. In Sections 4.2.3 and 4.2.4 we show how $[\mathbf{P}]^s_g$ and $[\mathbf{P}]^t_g$ are used to iteratively isolate all intersection points.

4.2.3 Geometric Interval Clipping

In this section we show how the geometric interval $[\mathbf{P}]^s_g$ is used to identify parameter intervals in s where $\mathbf{P}(s,t) \neq \mathbf{0}$. We begin by defining a line $\mathbf{L}_s$ through $\mathbf{0}$ parallel to $\mathbf{V}_0 + \mathbf{V}_1$, where $\mathbf{V}_0 = \mathbf{P}_{0,m} - \mathbf{P}_{0,0}$ and $\mathbf{V}_1 = \mathbf{P}_{n,m} - \mathbf{P}_{n,0}$. If we represent $\mathbf{L}_s$ with the implicit equation

$$\delta(x,y) = ax + by + c = 0, \tag{4.10}$$

then $\delta(x, y)$ is the signed distance from $\mathbf{L}_s$ to any point (x, y) scaled by $\sqrt{a^2 + b^2}$. Therefore, the signed, scaled distance from $\mathbf{L}_s$ to $\hat{\mathbf{P}}^{\mathbf{s}}(s, t)$ defined in Theorem 4.1 is

$$\hat{\delta}^s(s,t) = \sum_{j=0}^{m} B_j^m(t) \left((1-s)^2 \hat{\delta}^s_{0,j} + 2s(1-s)\hat{\delta}^s_{1,j}(s) + s^2 \hat{\delta}^s_{2,j} \right), \tag{4.11}$$

where $\hat{\delta}^s_{0,j}$, $\hat{\delta}^s_{1,j}(s)$ and $\hat{\delta}^s_{2,j}$ are distance measures from $\mathbf{L}_s$ to $\hat{\mathbf{P}}^{\mathbf{s}}_{0,j}$, $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j}(s)$ and $\hat{\mathbf{P}}^{\mathbf{s}}_{2,j}$, respectively. $\hat{\delta}^s_{0,j}$ and $\hat{\delta}^s_{2,j}$ are found by directly applying Equation 4.10 to the fixed control points $\hat{\mathbf{P}}^{\mathbf{s}}_{0,j} = (x_{0,j}, y_{0,j})$ and $\hat{\mathbf{P}}^{\mathbf{s}}_{2,j} = (x_{2,j}, y_{2,j})$ of $\hat{\mathbf{P}}^{\mathbf{s}}(s, t)$:

$$\hat{\delta}^s_{0,j} = \delta(\hat{\mathbf{P}}^{\mathbf{s}}_{0,j}) = ax_{0,j} + by_{0,j} + c, \qquad \hat{\delta}^s_{2,j} = \delta(\hat{\mathbf{P}}^{\mathbf{s}}_{2,j}) = ax_{2,j} + by_{2,j} + c. \tag{4.12}$$

However, $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j}(s)$ is a Bézier curve, so we must apply the function $\delta(x, y)$ to each of its control points $\hat{\mathbf{P}}^{\mathbf{s}}_{1,j,k} = (x_{1,j,k}, y_{1,j,k})$ to produce $\hat{\delta}^s_{1,j}(s)$:

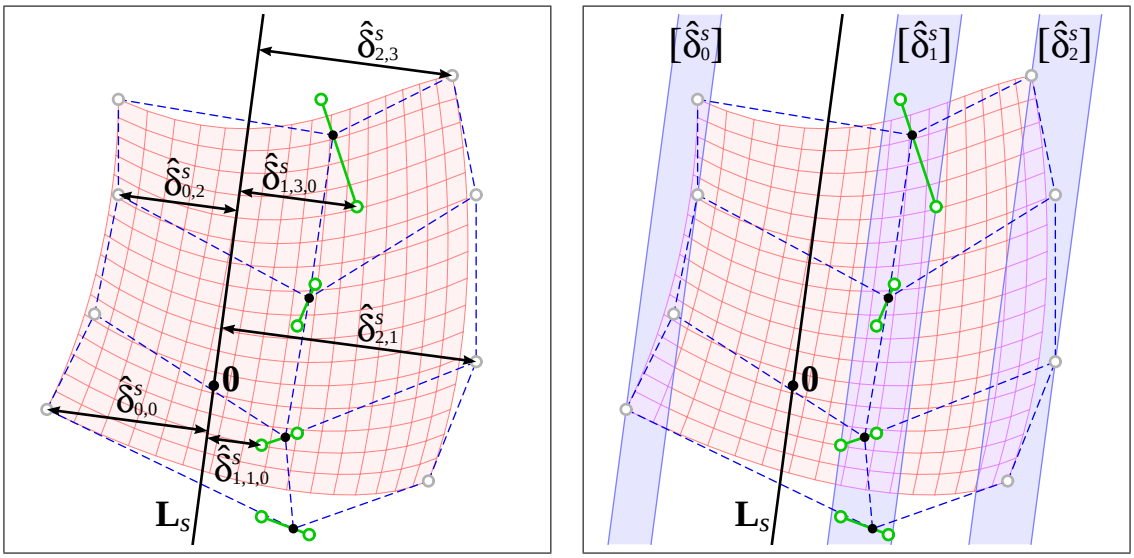
$$\hat{\delta}^s_{1,j}(s) = \sum_{0 \le k \le n-2} B_k^{n-2}(s) \hat{\delta}^s_{1,j,k}, \qquad \hat{\delta}^s_{1,j,k} = \delta(\hat{\mathbf{P}}^{\mathbf{s}}_{1,j,k}) = ax_{1,j,k} + by_{1,j,k} + c. \tag{4.13}$$

The replacement of $\mathbf{P}$ with the equivalent surface $\hat{\mathbf{P}}^{\mathbf{s}}$ now becomes important. The geometric interval transformation enables us to bound the distance from $\mathbf{L}_s$ to $\mathbf{P}(s, t)$ with an interval quadratic Bernstein polynomial $[\hat{\delta}^s](s)$:

$$[\hat{\delta}^s](s) = (1-s)^2 [\hat{\delta}^s_0] + 2s(1-s)[\hat{\delta}^s_1] + s^2 [\hat{\delta}^s_2], \tag{4.14}$$

where $[\hat{\delta}^s_0]$, $[\hat{\delta}^s_1]$ and $[\hat{\delta}^s_2]$ are intervals containing $\hat{\delta}^s_{0,j}$, $\hat{\delta}^s_{1,j}(s)$ and $\hat{\delta}^s_{2,j}$, respectively:

$$\begin{aligned}
[\hat{\delta}^s_0] &= [\hat{\delta}^s_{0,\min}, \hat{\delta}^s_{0,\max}] = [\min_{0 \le j \le m} \hat{\delta}^s_{0,j}, \max_{0 \le j \le m} \hat{\delta}^s_{0,j}], \\
[\hat{\delta}^s_1] &= [\hat{\delta}^s_{1,\min}, \hat{\delta}^s_{1,\max}] = [\min_{\substack{0 \le j \le m \\ 0 \le k \le n-2}} \hat{\delta}^s_{1,j,k}, \max_{\substack{0 \le j \le m \\ 0 \le k \le n-2}} \hat{\delta}^s_{1,j,k}], \\
[\hat{\delta}^s_2] &= [\hat{\delta}^s_{2,\min}, \hat{\delta}^s_{2,\max}] = [\min_{0 \le j \le m} \hat{\delta}^s_{2,j}, \max_{0 \le j \le m} \hat{\delta}^s_{2,j}].
\end{aligned} \tag{4.15}$$



(a) $\hat{\delta}^s$ measures which define $\hat{\delta}^s_{i,\min}$ and $\hat{\delta}^s_{i,\max}$.

(b) Visualization of $[\hat{\delta}^s_0]$, $[\hat{\delta}^s_1]$ and $[\hat{\delta}^s_2]$.

Figure 4.3: In this example, $[\hat{\delta}^s_0] = [\hat{\delta}^s_{0,0}, \hat{\delta}^s_{0,2}]$, $[\hat{\delta}^s_1] = [\hat{\delta}^s_{1,1,0}, \hat{\delta}^s_{1,3,0}]$ and $[\hat{\delta}^s_2] = [\hat{\delta}^s_{2,3}, \hat{\delta}^s_{2,1}]$.

Figure 4.3 illustrates the gathering of distance measures $\hat{\delta}^s_{0,j}$, $\hat{\delta}^s_{1,j,k}$ and $\hat{\delta}^s_{2,j}$ to form the intervals $[\hat{\delta}^s_0]$, $[\hat{\delta}^s_1]$ and $[\hat{\delta}^s_2]$. For the example hybrid surface $\hat{\mathbf{P}}^{\mathbf{s}}$ from Figure 4.2a, we find $[\hat{\delta}^s_0] = [\hat{\delta}^s_{0,0}, \hat{\delta}^s_{0,2}] = [-206, -152]$, $[\hat{\delta}^s_1] = [\hat{\delta}^s_{1,1,0}, \hat{\delta}^s_{1,3,0}] = [70, 150]$ and $[\hat{\delta}^s_2] = [\hat{\delta}^s_{2,3}, \hat{\delta}^s_{2,1}] = [248, 326]$. Note that these values are scaled by the length of the normal vector used to define $\mathbf{L}_s$.

Equation 4.14 exhibits the useful property $\delta(\mathbf{P}(s,t)) \in [\hat{\delta}^s](s)$ for $0 \leq s \leq 1$. Since $\mathbf{L}_s$ contains $\mathbf{0}$, the roots of $[\hat{\delta}^s](s)$, $0 \leq s \leq 1$, are intervals containing values of s for which $\delta(\mathbf{P}(s,t)) = 0$ and, therefore, $\mathbf{P}(s,t) = \mathbf{0}$. While it is possible to use interval arithmetic and the quadratic formula to find the roots of $[\hat{\delta}^s](s)$, the dependency problem (see Section 2.1.3) leads to excessive overestimation. There is, however, a simple geometric solution, as illustrated in Figure 4.4.

The roots of $[\hat{\delta}^s](s)$ are intervals with end points made up of $t = 0$ (if $0 \in [\hat{\delta}^s_0]$), $t = 1$ (if $0 \in [\hat{\delta}^s_2]$) and the values $t \in [0,1]$ corresponding to the roots of the equations

$$\hat{\delta}_{\min}(s) = (1-s)^2\hat{\delta}^s_{0,\min} + 2s(1-s)\hat{\delta}^s_{1,\min} + s^2\hat{\delta}^s_{2,\min} = 0, \tag{4.16}$$

$$\hat{\delta}_{\max}(s) = (1-s)^2\hat{\delta}^s_{0,\max} + 2s(1-s)\hat{\delta}^s_{1,\max} + s^2\hat{\delta}^s_{2,\max} = 0. \tag{4.17}$$

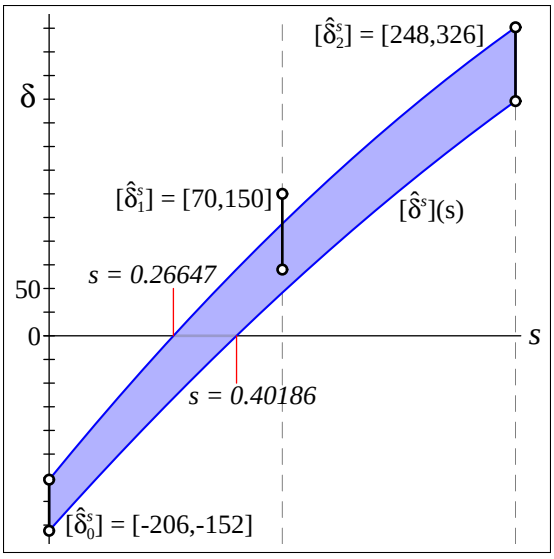


Figure 4.4: $[\hat{\delta}^s](s)$ crosses the s axis at $s \in [0.26647, 0.40186]$.

The polynomials $\hat{\delta}_{\min}(s)$ and $\hat{\delta}_{\max}(s)$ are the lower and upper bounds of $[\hat{\delta}^s](s)$, respectively. The roots of $\hat{\delta}_{\min}(s)$ and $\hat{\delta}_{\max}(s)$ (along with the values $t = 0$ and $t = 1$, where required) are taken pairwise, in numerical order, as the end points of intervals. These intervals are the roots of $[\hat{\delta}^s](s)$, of which there will be no more than two. Note that any double roots of $\hat{\delta}_{\min}(s)$ and $\hat{\delta}_{\max}(s)$ must be counted twice.

To obtain accurate values for the roots of $\hat{\delta}_{\min}(s)$ and $\hat{\delta}_{\max}(s)$, it is important to use a numerically stable method for solving the quadratic formula which can be easily applied to a polynomial in the Bernstein basis. In practice, we have found that pseudo-conversion from the Bernstein basis to the power basis [20], combined with a numerically stable quadratic equation solver [4], is an efficient method to obtain a high quality result.

For each interval root $[s_0, s_1]$ of $[\hat{\delta}^s](s)$, regions of the projected surface $\mathbf{P}(s,t)$ for which $s < s_0$ or $s > s_1$ are clipped away using the de Casteljau algorithm. For example, $[\hat{\delta}^s](s)$ in Figure 4.4 has a single root at approximately $[0.26647, 0.40186]$. In this example, portions of $\mathbf{P}(s,t)$ below $s = 0.26647$ and above $s = 0.40186$ are clipped since $\mathbf{P}(s,t) \neq \mathbf{0}$ for $s < 0.26647$ or $s > 0.40186$. Figure 4.5 shows $\mathbf{P}(s,t)$ after it has been clipped.

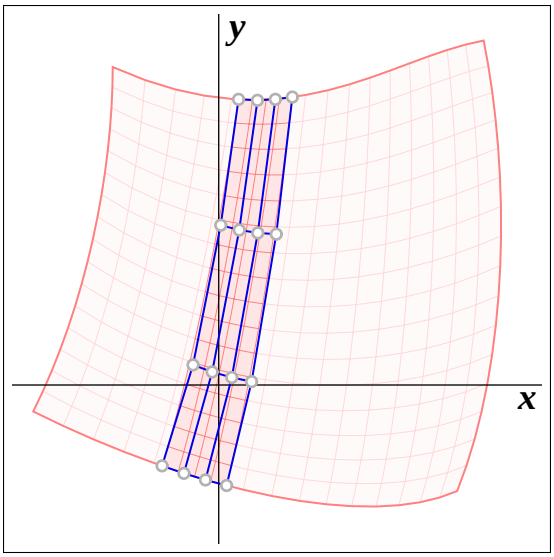


Figure 4.5: $\mathbf{P}(s,t)$ is clipped to $s \in [0.26647, 0.40186]$ during the first clipping step.

Note that we may identify up to two roots for $[\hat{\delta}^s](s)$. If there are two roots, then two separate clipping steps are performed, resulting in two new subsurfaces. Each subsurface is then processed independently, as each bounds separate potential intersection points. If $[\hat{\delta}^s](s)$ has no roots, then no intersection exists.

We use the phrase *geometric interval clipping in* s to describe the process of using the interval Bernstein polynomial $[\hat{\delta}^s](s)$ to identify regions in the s parameter direction of $\mathbf{P}(s,t)$ which can be safely clipped. Geometric interval clipping in t is the analogous process whereby $[\hat{\delta}^t](t)$ is constructed from $\hat{\mathbf{P}}^{\mathbf{t}}(s,t)$ to find clipping values in t. Section 4.2.4 shows how alternate application of geometric interval clipping in s and t converges on points where $\mathbf{P}(s,t) = \mathbf{0}$.

4.2.4 Iterating

Geometric interval clipping in t is accomplished by applying the steps detailed in Section 4.2.3 to the hybrid surface $\hat{\mathbf{P}}^{\mathbf{t}}(s,t)$ (Theorem 4.2). We begin this section by briefly outlining this process.

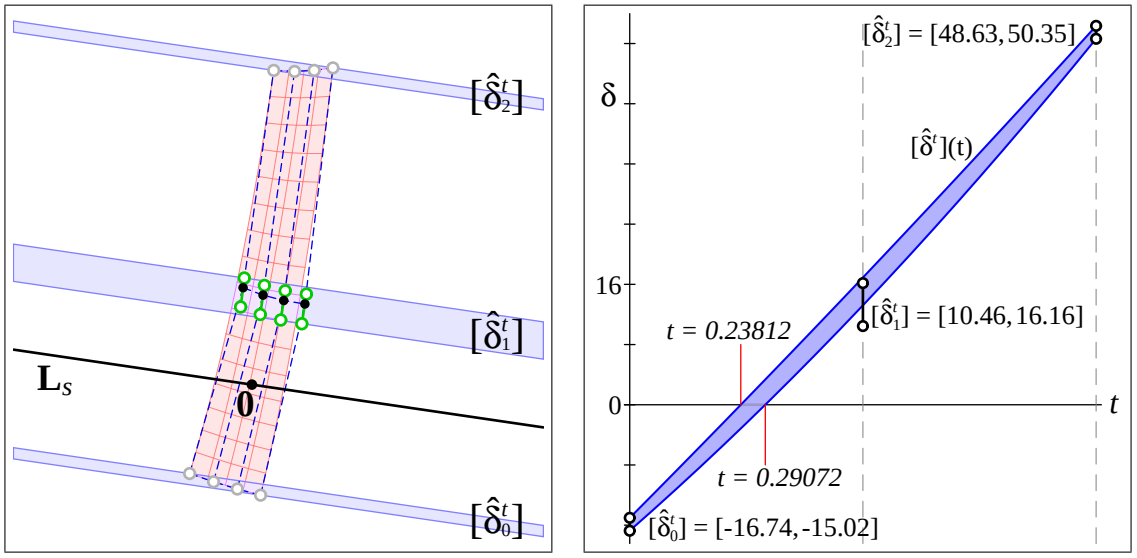
First, we define the line $\mathbf{L}_t$ as a line through $\mathbf{0}$ parallel to $(\mathbf{P}_{n,0} - \mathbf{P}_{0,0}) + (\mathbf{P}_{n,m} - \mathbf{P}_{0,m})$. As in Equation 4.10, we use the function $\delta(x,y)$ to denote the signed,

scaled distance from $\mathbf{L}_t$ to an arbitrary point (x, y). Therefore, the signed, scaled distance from $\mathbf{L}_t$ to $\hat{\mathbf{P}}^{\mathbf{t}}(s,t)$ is

$$\hat{\delta}^t(s,t) = \sum_{i=0}^{n} B_i^n(s) \left((1-t)^2 \hat{\delta}_{i,0}^t + 2t(1-t)\hat{\delta}_{i,1}^t(t) + t^2 \hat{\delta}_{i,2}^t \right), \tag{4.18}$$

$$\hat{\delta}_{i,1}^t(t) = \sum_{0 \le k \le m-2} B_k^{m-2}(t) \hat{\delta}_{i,1,k}^t, \tag{4.19}$$

where $\hat{\delta}_{i,0}^t = \delta(\hat{\mathbf{P}}_{i,0}^{\mathbf{t}})$, $\hat{\delta}_{i,1,k}^t = \delta(\hat{\mathbf{P}}_{i,1,k}^{\mathbf{t}})$ and $\hat{\delta}_{i,2}^t = \delta(\hat{\mathbf{P}}_{i,2}^{\mathbf{t}})$.



(a) Visualization of $[\hat{\delta}_0^t]$, $[\hat{\delta}_1^t]$ and $[\hat{\delta}_2^t]$.

(b) $[\hat{\delta}^t](t)$ has a root at $t \in [0.23812, 0.29027]$.

Figure 4.6: The t clipping values are found by solving for the roots of $[\hat{\delta}^t](t)$.

We may find intervals bounding the control points of $\hat{\delta}^t(s,t)$ to form $[\hat{\delta}^t](t)$, an interval quadratic Bernstein polynomial bounding the distance from $\mathbf{L}_t$ to $\mathbf{P}(s,t)$:

$$[\hat{\delta}^t](t) = (1-t)^2[\hat{\delta}_0^t] + 2t(1-t)[\hat{\delta}_1^t] + t^2[\hat{\delta}_2^t], \tag{4.20}$$

$$\begin{aligned}
[\hat{\delta}_0^t] &= [\hat{\delta}_{0,\min}^t, \hat{\delta}_{0,\max}^t] = [\min_{0 \le i \le n} \hat{\delta}_{i,0}^t, \max_{0 \le i \le n} \hat{\delta}_{i,0}^t], \\
[\hat{\delta}_1^t] &= [\hat{\delta}_{1,\min}^t, \hat{\delta}_{1,\max}^t] = [\min_{\substack{0 \le i \le n \\ 0 \le k \le m-2}} \hat{\delta}_{i,1,k}^t, \max_{\substack{0 \le i \le n \\ 0 \le k \le m-2}} \hat{\delta}_{i,1,k}^t], \\
[\hat{\delta}_2^t] &= [\hat{\delta}_{2,\min}^t, \hat{\delta}_{2,\max}^t] = [\min_{0 \le i \le n} \hat{\delta}_{i,2}^t, \max_{0 \le i \le n} \hat{\delta}_{i,2}^t].
\end{aligned} \tag{4.21}$$

The intervals $[\hat{\delta}_0^t]$, $[\hat{\delta}_1^t]$ and $[\hat{\delta}_2^t]$ are represented in Figure 4.6a as bars bounding the distance of each row of control points to $\mathbf{L}_t$. The interval polynomial $[\hat{\delta}^t](t)$ is shown in Figure 4.6b. The interval roots of $[\hat{\delta}^t](t)$ determine the clipping values along the t parameter direction. In this example, the de Casteljau algorithm is applied to clip away the surface for $t < 0.23812$ and $t > 0.29027$.

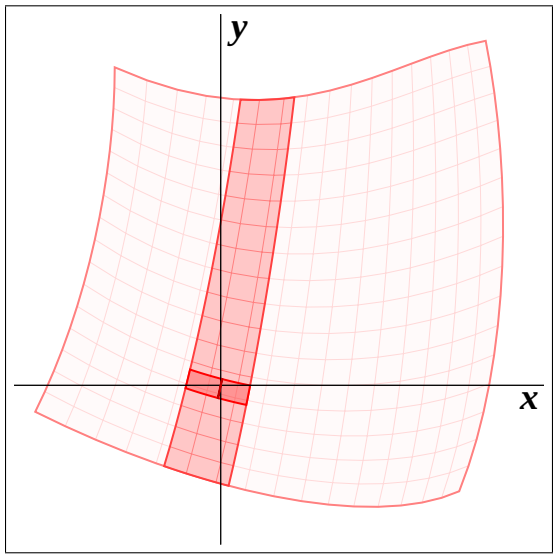


Figure 4.7: Convergence to **0**.

By alternately applying geometric interval clipping in the s and t parameter directions, the portions of the surface which are not intersected by the ray are clipped away. This process produces a nested sequence of rectangular regions in parameter space; each level more tightly bounding the parameter locations of the intersection points, as illustrated in Figure 4.7. Once the width and height of an enclosing region approximate the parameters (s, t) of the intersection point to within tolerance, the center of the region is reported as an intersection.

Note that each region may immediately contain several sub-regions if, for example, $[\hat{\delta}^s](s)$ or $[\hat{\delta}^t](t)$ has multiple roots. The refinement process is repeated recursively for all subregions until each area has converged to **0** or has been shown to contain no intersections.

4.2.5 Multiple Intersections

If a ray intersects a Bézier patch at a single point, then the GeoClip algorithm, as presented so far, is sufficient to locate the parameter values for the intersection point. The final component of the GeoClip algorithm is a method for reliably isolating multiple ray/surface intersections.

The same heuristic employed by BezClip [19] is used to handle multiple intersections. That is, the width of a parameter interval must be reduced by at least 20% during a clipping step or the surface is split in half along the current parameter direction. Each half is then processed recursively. For example, suppose that geometric interval clipping in s is being performed. If $[\delta^s](s)$ has a single root at $s \in [s_{\min}, s_{\max}]$, then the patch is split in half along the s parameter direction if $[s_{\min}, s_{\max}]$ covers more than 80% of the remaining curve segment. The heuristic is applied to geometric interval clipping in t in an obviously similar manner.

4.3 Timing Comparisons

Implementations of BezClip and Impl have been tested to determine how GeoClip compares to these popular algorithms. Since GeoClip is an extension of BezClip, our tests are intended to provide a reasonable means of evaluating the extent to which geometric intervals improve performance. Impl is a very fast algorithm which takes a very different approach to the ray/surface intersection problem. Testing Impl gives us some idea of how GeoClip compares to more diverse schemes. Subdivision and numerical methods were not included as other sources [19] indicate that BezClip performs well when compared to these algorithm classes.

Our tests focus on bicubic Bézier patches since they are the most widely used Bézier surface. The first test we conducted is designed to evaluate how each algorithm performs when applied to randomly generated Bézier surfaces. A collection of 100,000

bicubic patches in $\mathbb{R}^2$ is generated by randomly selecting 16 control points (x, y) per patch, where $x, y \in [-1, 1]$. If each planar patch is thought of as the projection of a patch in $\mathbb{R}^3$ along a ray, as described in Section 4.2.1, then the origin of the plane corresponds to the point where the ray intersects the surface. Impl, BezClip and GeoClip are each applied to the set of 100,000 randomly generated patches to determine the parametric location(s) of the origin. Average relative runtimes for this test are presented in Table 4.1.

Algorithm	Impl	BezClip	GeoClip
Relative Time	1.000	2.044	1.568

Table 4.1: Relative computation times for randomly generated bicubic patches.

This test makes it apparent that GeoClip is significantly faster than BezClip. We have also performed similar tests using patches of varying degree. These tests indicate that GeoClip is faster than BezClip for bilinear, biquadratic and biquartic Bézier surfaces as well. Performance is approximately equal for biquintic patches. For higher degrees, BezClip outperforms GeoClip.

Another striking result from Table 4.1 is that both BezClip and GeoClip appear to be slower than Impl when applied to a set of cubic Bézier surfaces with randomized control vertices. Must useful patches, however, are much smoother than a randomly generated patch will tend to be. Applying these algorithms to ray tracing would be a more realistic method of evaluation. Therefore, we have integrated each of these algorithms into a simple ray tracer, enabling them to be used to render bicubic Bézier patches composing a scene.

Using this ray tracer, we rendered the scene depicted in Figure 4.8 which contains the classic Utah teapot. This scene was rendered using Impl, BezClip and GeoClip to calculate each ray/surface intersection. The total time spent calculating intersections was recorded and averaged over several trials. GeoClip took the

least amount of time with BezClip and Impl taking about 28% and 43% longer, respectively. Ray tracing timing results are summarized in Table 4.2.

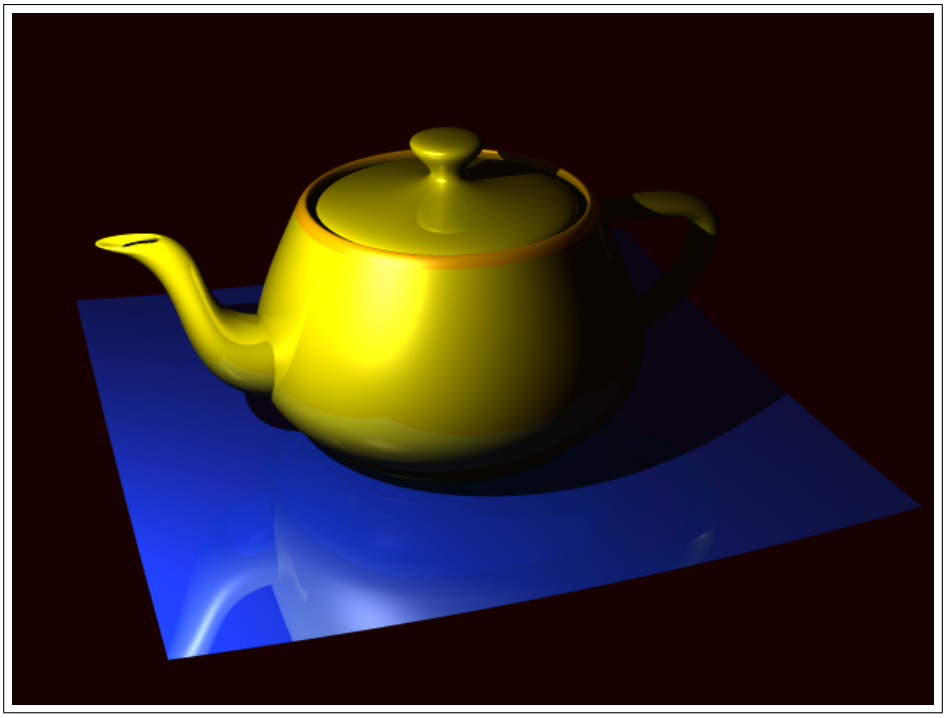
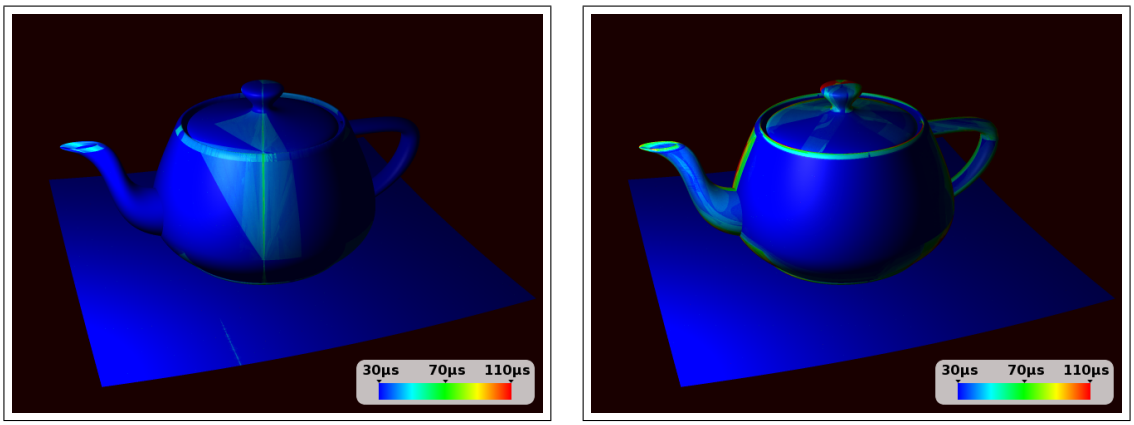


Figure 4.8: Utah teapot rendering.

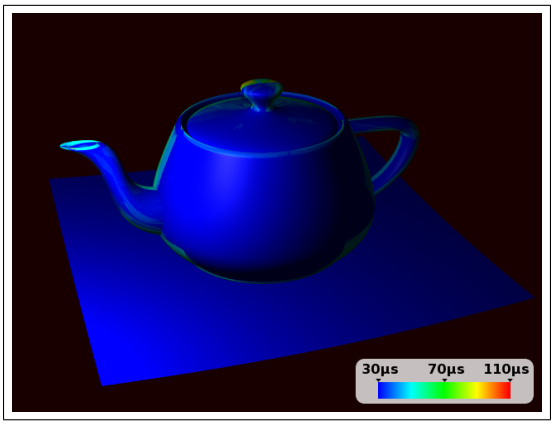
These ray tracing timings seem to contradict results obtained using randomly generated patches. Based on Table 4.1, one might expect Impl to take the least time. To get a better idea of why this isn't the case, we have produced a series of images to help us visualize the time required to calculate primary ray intersections. Figure 4.9 uses a color gradient to code each pixel by the time required to find the intersections with the first patch along the ray through each pixel.

In Figure 4.9, BezClip and GeoClip exhibit difficulties near patch silhouette edges. In these areas the ray is close to the outline of the object, producing multiple intersections that are parametrically close together. This effect can be seen clearly on the knob of the lid or the opening of the spout when using BezClip (Figure 4.9b). Comparing this to Figure 4.9c, we can see that GeoClip also has difficulties at silhouette edges, but the problem is less pronounced. The performance of the Impl



(a) Impl can slow down in smooth regions.

(b) BezClip has trouble at silhouette edges.

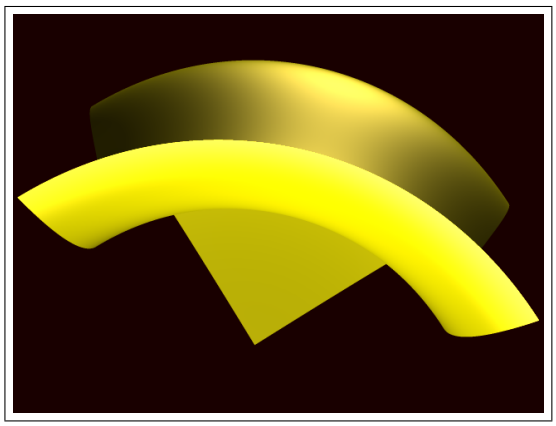


(c) GeoClip has less trouble at silhouette edges.

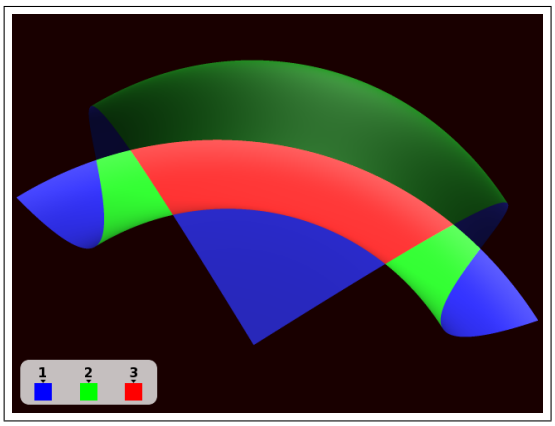
Figure 4.9: Time taken to calculate primary ray intersections with visible patches.

algorithm seems to be more strongly affected by factors other than silhouette edges which are not readily apparent in Figure 4.9a. However, it is clear that Impl can be significantly slower than GeoClip and BezClip in smooth regions, such as the body of the teapot.

Figure 4.10a depicts a single Bézier patch taken from the knob of the teapot's lid. The patch is positioned so that a single primary ray may intersect the surface up to three times. Color coding is used in Figure 4.10b to illustrate how many times the ray through a given pixel intersects the Bézier patch. This configuration was selected to highlight the circumstances in which BezClip and GeoClip would have the most difficulty. In this case, we would expect Impl to outperform GeoClip.

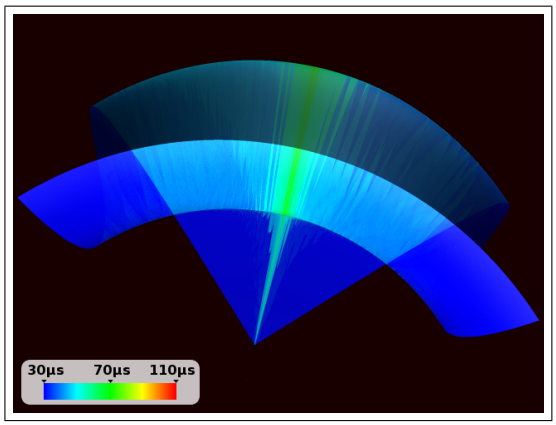


(a) A Bézier patch from the teapot's knob.

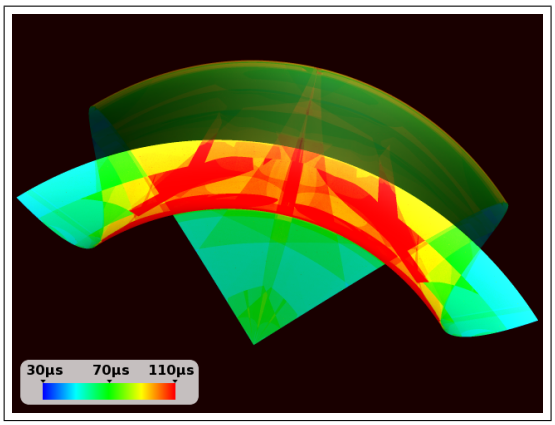


(b) Color coded by number of intersections.

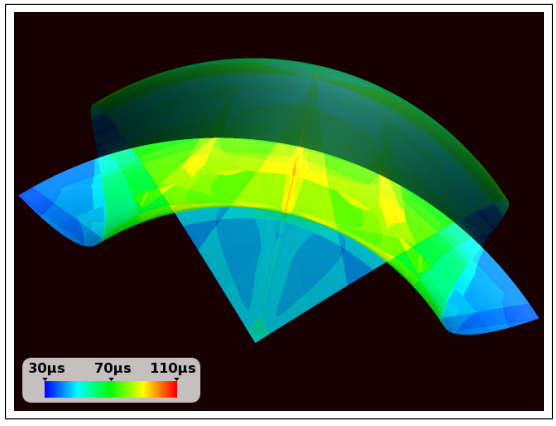
Figure 4.10: A patch from the teapot's knob, oriented to have multiple intersections.



(a) Impl timing results.



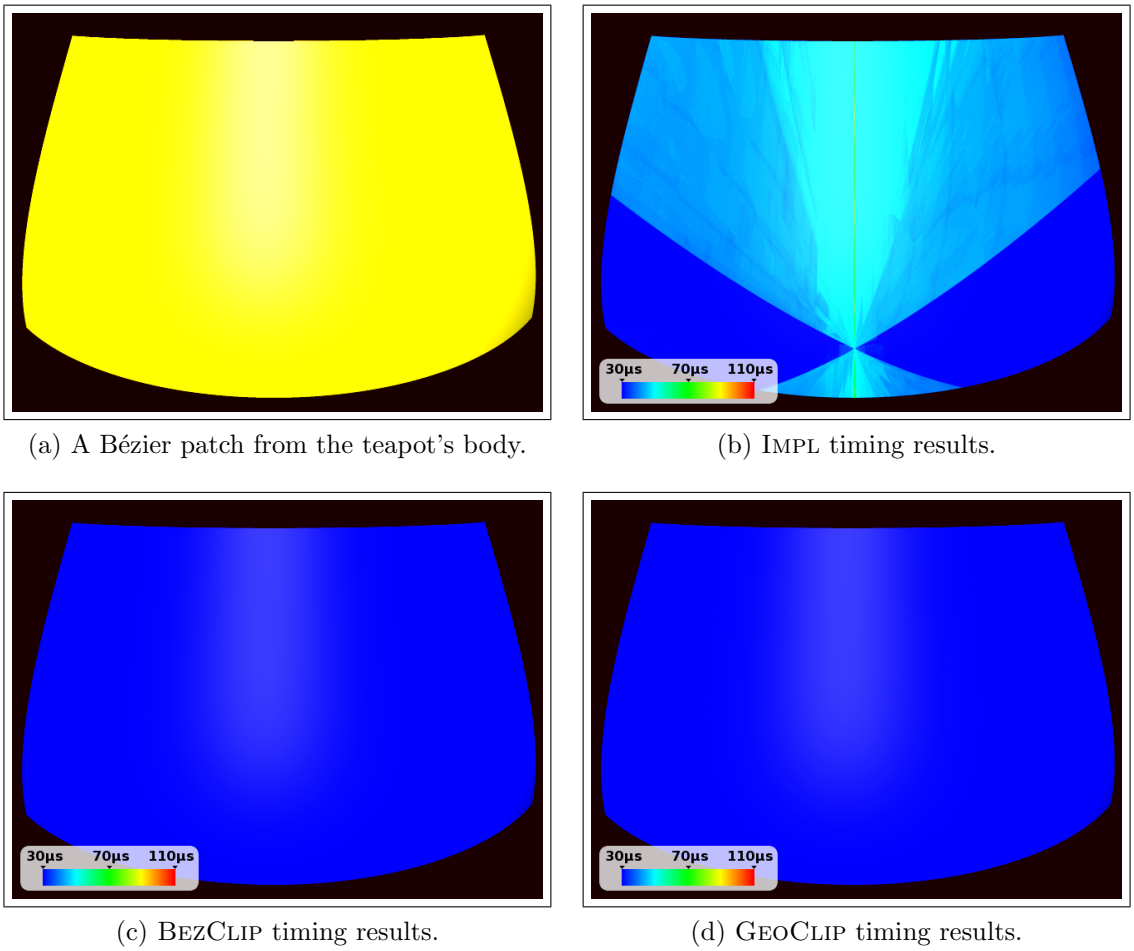
(b) BezClip timing results.



(c) GeoClip timing results.

Figure 4.11: Time taken to calculate primary ray intersections with the knob patch.

Now consider Figure 4.11. In this figure, the color coding scheme used in Figure 4.9 is applied to the knob patch to denote the time taken to calculate each ray/surface intersection. BezClip, shown in Figure 4.11b, has the most difficulty with this scene, slowing down significantly in regions where the ray intersects the surface three times. Silhouette edges also present a problem for BezClip. Though noticeably improved, a similar pattern can be discerned in the timing results for GeoClip (Figure 4.11c). The same regions tend to be troublesome, but GeoClip is almost universally faster than BezClip in this example. Despite the advantage that GeoClip has over BezClip, Figure 4.11a clearly shows that Impl handles this case better than the other algorithms.



(a) A Bézier patch from the teapot's body.

(b) Impl timing results.

(c) BezClip timing results.

(d) GeoClip timing results.

Figure 4.12: Time taken to calculate primary ray intersections with a body patch.

The final ray tracing example we will consider is shown in Figure 4.12. The Bézier surface used for this case comes from the teapot body and is never intersected more than once by a primary ray. Figures 4.12b, 4.12c and 4.12d illustrate the time taken to calculate each intersection when using Impl, BezClip and GeoClip, respectively. The fact that Impl has the most difficulty here is consistent with our earlier observation that BezClip and GeoClip tend to do better in regions that lack multiple intersections and silhouette edges.

Figure	*Figure 4.8*	*Figure 4.10a*	*Figure 4.12a*
Impl	1.427	1.000	2.026
BezClip	1.283	1.777	1.252
GeoClip	1.000	1.388	1.000

Table 4.2: Relative ray/surface intersection computation times.

The relative time required to calculate all ray/surface intersections in each ray tracing example is summarized in Table 4.2. The ray tracing tests, as well as the randomly generated patch tests, empirically show that GeoClip improves upon BezClip when calculating the intersection of a ray and a bicubic Bézier surface. This is the primary goal of the BezClip algorithm. Additionally, Table 4.2 indicates that GeoClip is the fastest of the three tested algorithms for ray tracing scenes where most rays intersect a surface only once. In this scenario, GeoClip appears to be at least 25% faster than BezClip and up to twice as fast as Impl.

4.4 Observations and Conclusions

The GeoClip ray/surface intersection algorithm has been detailed in this chapter. Section 4.2 explains how geometric intervals are used to extend BezClip [19] to form a new algorithm that is both simple and robust. In Section 4.3 we show that this algorithm, GeoClip, takes less time than BezClip to find intersections using randomly generated patches. Additionally, we provide evidence that GeoClip is

faster than BezClip for ray tracing bicubic Bézier patches. GeoClip also performs favorably in comparison to the Impl algorithm for ray tracing bicubic surfaces.

The geometric interval used for GeoClip in this chapter is constructed by transforming a Bézier patch into a hybrid surface that is quadratic in one parameter direction. Since using a quadratic hybrid surface proved beneficial, it is logical to consider extending the technique by using cubic hybrid surfaces. In experimenting with this possibility, we found that any potential benefit to this scheme is outweighed by the complexity of clipping using a cubic geometric interval. Timing results suggest that such an algorithm would be slower than BezClip, and only slightly faster than Impl, for ray tracing simple scenes. This indicates that the quadratic geometric interval used for GeoClip strikes the proper balance between complexity and convergence rate.

GeoClip improves upon BezClip by leveraging geometric intervals to exploit the geometry of Bézier surfaces for finding ray/surface intersections. Since GeoClip is an extension of BezClip, a potentially fruitful avenue for improving GeoClip might be to consider applying techniques that have previously been successfully applied to BezClip. For example, Wang [31] extends BezClip using ray coherence to speed up ray tracing. Future research might also apply these methods to GeoClip to improve performance.

Chapter 5

Conclusions and Future Directions

This work has introduced the concept of the geometric interval. We have shown that geometric interval formulations for bounding Bézier curves and surfaces can be used effectively for finding intersections. A brief summary of our results is presented in Section 5.1. A discussion of ideas for extending geometric interval techniques into new areas follows in Section 5.2.

5.1 Summary of Results

In Chapter 3, we show that a Bézier curve of cubic or higher degree can be transformed into an equivalent hybrid Bézier curve. The properties of the hybrid curve, when combined with interval analysis techniques, make it attractive for locating curve/curve intersections. Therefore, we refer to the hybrid curve as a geometric interval.

The GeoClip algorithm is formed by extending the curve/curve BezClip algorithm [27] using geometric intervals. A quadratic bound derived from the hybrid curve allows GeoClip to improve upon the convergence rate of BezClip without adding much per-iteration complexity. In our tests, GeoClip takes less time than BezClip to find all intersection points between randomly generated Bézier curves, regardless of degree. For quadratic and cubic curves, GeoClip is often faster than the Impl algorithm when the curves intersect at a single point. GeoClip is always faster than Impl for curves of degree four and higher.

Chapter 4 extends the principles developed in Chapter 3 to ray/surface intersections. The geometric interval for curves is adapted to handle transforming a Bézier surface patch into a hybrid surface that is quadratic along one parameter direction. This hybrid patch serves as a geometric interval for Bézier surfaces.

A new GeoClip algorithm is derived by extending BezClip for ray/surface intersections [19] using geometric interval techniques. The ray/surface GeoClip algorithm shares many properties with its curve/curve counterpart. In timing tests, GeoClip, once again, generally takes less time to find all intersections than the BezClip algorithm it improves upon. GeoClip also compares favorably to Impl, especially in our ray tracing tests where a ray intersects a single Bézier patch multiple times for only a relatively small percentage of pixels.

5.2 Future Directions

The geometric interval techniques used to form the GeoClip algorithms from BezClip may also be applicable to other areas. For example, it would be trivial to adapt the geometric interval for curves to Bernstein polynomials. A polynomial geometric interval might hold some unique benefits for root finding algorithms. While we have not tested this approach, the principles for doing so are all present in this work.

Curve/surface intersection is another area where future research could possibly apply geometric interval methods. Using curve and surface geometric intervals, it may be possible to efficiently narrow the parameter intervals where a curve meets a surface. Locating curve/surface intersection points is a vital building block for more complex algorithms, such as finding the intersection boundary between two surface patches.

Bibliography

[1] Ball, A. A. Consurf. part one: introduction of the conic lofting tile;. *Computer-Aided Design 6*, 4 (1974), 243–249.

[2] Barth, W., and Stürzlinger, W. Efficient ray tracing for Bézier and B-spline surfaces. *Computers & Graphics 17*, 4 (1993), 423–430.

[3] Bartoň, M., and Jüttler, B. Computing roots of polynomials by quadratic clipping. *Computer Aided Geometric Design 24*, 3 (2007), 125–141.

[4] Blinn, J. F. How to solve a quadratic equation. part 2. *Computer Graphics and Applications, IEEE 26*, 2 (2006), 82–87.

[5] Böhm, W., Farin, G., and Kahmann, J. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design 1*, 1 (1984), 1–60.

[6] Ceberio, M., and Granvilliers, L. Horner's rule for interval evaluation revisited. *Computing 69*, 1 (2002), 51–81.

[7] Coons, S. A. Surfaces for computer-aided design of space forms. Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA, 1967.

[8] de Casteljau, P. Outillage méthodes calcul. Tech. rep., André Citroen Automobiles SA, 1959.

[9] Hansen, E. Interval forms of Newton's method. *Computing 20* (1978), 153–16.

[10] Hickey, T. J., Ju, Q., and van Emden, M. H. Interval arithmetic: From principles to implementation. *Journal of the ACM 48*, 5 (2001), 1038–1068.

[11] Horner, W. G. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London 109* (1819), 308–335.

[12] Kajiya, J. T. Ray tracing parametric patches. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1982), ACM Press, pp. 245–254.

[13] Koparkar, P. A., and Mudur, S. P. A new class of algorithms for the processing of parametric curves. *Computer-Aided Design 15*, 1 (1983), 41–45.

[14] Lane, J. M., and Riesenfeld, R. F. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions of Pattern Analysis and Machine Intelligence 2* (1980), 35–46.

[15] Manocha, D., and Krishnan, S. Algebraic pruning: a fast technique for curve and surface intersection. *Computer Aided Geometric Design 14*, 9 (1997), 823–845.

[16] Martin, W., Cohen, E., Fish, R., and Shirley, P. Practical ray tracing of trimmed NURBS surfaces. *Journal of Graphics Tools 5*, 1 (2000), 27–52.

[17] Moore, R. E. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[18] Neumaier, A. *Interval Methods for Systems of Equations*, vol. 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, UK, 1990.

[19] Nishita, T., Sederberg, T. W., and Kakimoto, M. Ray tracing trimmed rational surface patches. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), ACM Press, pp. 337–345.

[20] Pavlidis, T. *Algorithms for graphics and image processing*. W. H. Freeman & Co., Computer Science Press, New York, NY, USA, 1982.

[21] Ratz, D. Inclusion isotone extended interval arithmetic. Tech. Rep. D-76128, Institut für Angewandte Mathematik, Universität Karlsruhe, May 1996.

[22] Saito, T., Wang, G.-J., and Sederberg, T. W. Hodographs and normals of rational curves and surfaces. *Computer Aided Geometric Design 12*, 4 (1995), 417–430.

[23] Sederberg, T. W. *Implicit and parametric curves and surfaces for computer aided geometric design*. PhD thesis, Purdue University, 1983.

[24] Sederberg, T. W., Anderson, D. C., and Goldman, R. N. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics and Image Processing 28* (1984), 72—84.

[25] Sederberg, T. W., and Farouki, R. T. Approximation by interval Bézier curves. *IEEE Computer Graphics and Applications 12*, 5 (1992), 87–95.

[26] Sederberg, T. W., and Kakimoto, M. *NURBS for Curve and Surface Design*. SIAM, 1991, ch. Approximating Rational Curves Using Polynomial Curves, pp. 149–158.

[27] Sederberg, T. W., and Nishita, T. Curve intersection using Bézier clipping. *Computer Aided Design 22*, 9 (1990), 538–549.

[28] Sederberg, T. W., and Parry, S. R. Comparison of three curve intersection algorithms. *Computer Aided Design 18*, 1 (1986), 58–64.

[29] Spencer, M. R. *Polynomial Real Root Finding in Bernstein Form*. PhD thesis, Brigham Young University, 1994.

[30] Toth, D. L. On ray tracing parametric surfaces. *SIGGRAPH Computer Graphics 19*, 3 (1985), 171–179.

[31] Wang, S.-W., Shih, Z.-C., and Chang, R.-C. An improved rendering technique for ray tracing Bézier and B-spline surfaces. *The Journal of Visualization and Computer Animation 11*, 4 (2000), 209–219.

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

[33] Woodward, C. *Ray tracing parametric surfaces by subdivision in viewing plane*. Springer-Verlag New York, Inc., New York, NY, USA, 1989, pp. 273–287.