

Surface-Surface Intersection with Validated Error Bounds

by

Harish Mukundan

B.Tech. in Naval Architecture and Ocean Engineering (2002)
Indian Institute of Technology, Madras, India

Submitted to the Department of Ocean Engineering and the Department of
Mechanical Engineering

in partial fulfillment of the requirements for the degrees of

Master of Science in Ocean Engineering

and

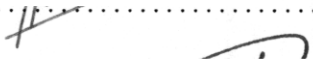
Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005

© Massachusetts Institute of Technology 2005. All rights reserved.

Author
 Department of Ocean Engineering
November 29, 2004

Certified by
Nicholas M. Patrikalakis, Kawasaki Professor of Engineering,
Professor of Ocean and Mechanical Engineering
Thesis Supervisor

Accepted by
Michael S. Triantafyllou, Professor of Ocean Engineering
Chairman, Departmental Committee on Graduate Students
Department of Ocean Engineering

Accepted by
Lallit Anand, Professor of Mechanical Engineering
Chairman, Departmental Committee on Graduate Students
Department of Mechanical Engineering

Surface-Surface Intersection with Validated Error Bounds

by

Harish Mukundan

Submitted to the Department of Ocean Engineering
and the Department of Mechanical Engineering
on November 29, 2004, in partial fulfillment of the
requirements for the degrees of
Master of Science in Ocean Engineering
and
Master of Science in Mechanical Engineering

Abstract

This thesis presents a robust method for tracing intersection curve segments between continuous rational parametric surfaces, typically rational polynomial parametric surface patches. Using a validated ordinary differential equation (ODE) system solver based on interval arithmetic, we obtain a continuous, validated upper bound for the intersection curve segment in the parametric space of each surface. Application of the validated ODE solver in the context of eliminating the pathological phenomena of straying and looping is discussed. We develop a method to achieve a continuous gap-free boundary with a definite numerically verified upper bound for the intersection curve error in parameter space. This bound in parametric space is further mapped to an upper bound for the intersection curve error in 3D model space, denoted as model space error, which assists in defining robust boundary representation models of complex three-dimensional solids. In addition, we also discuss a method for controlling this model space error so that it takes values below a predefined threshold (tolerance). Application of the above method to various examples is further demonstrated.

Thesis Supervisor: Nicholas M. Patrikalakis, Kawasaki Professor of Engineering,
Professor of Ocean and Mechanical Engineering

Acknowledgments

This thesis is the culmination of two years of my work at MIT. I recognize that this learning experience is just a formal introduction to the more informal experience called life. At this point I would like to thank so many people without whose help this thesis would never have been possible.

First of all I would like to thank my advisor Prof. Nicholas M. Patrikalakis for his motivation, mentorship and his expert guidance through the labyrinth of research. I would also like to thank Prof. Takashi Maekawa and Prof. Takis Sakkalis and Dr. Kwang Hee Ko for all their help, and many a fruitful discussions on my research. I also am grateful to Prof. C. M. Hoffmann, Prof. N. S. Nedialkov, Prof. T. J. Peters and Prof. N. F. Stewart for useful discussions and their comments on my work. I gratefully acknowledge the funding for this work which came from NSF under the grants DMS-0138098 and CCR-0231511.

I feel fortunate to have been taught by the masters in their corresponding fields at MIT, Professors T. R. Akylas, K. J. Bathe, J. J. Connor, D. C. Gossard, S. Hunter, E. Kausel, P. Koev, A. T. Patera, N. M. Patrikalakis, J. Peraire, R. Stocker, N. P. Suh, A. Techet, A. Toomre, K. J. Vandiver, D. Veneziano, J. K. White and D. K. P. Yue. I gratefully remember and acknowledge my Professors at IIT-Madras V. G. Idichandy, S. Surendran and R. Natarajan for their recommendation letters without which I would not have been at MIT in the first place.

Many thanks to Design Laboratory staff Dr. C. Evangelinos, Dr. W. Cho and Mr. F. Baker and current as well as previous members Da, Hongye, Micaela, Megumi, Namik, Stephen and Vanessa. I also thank my colleagues Dee, Ding, Ioannis, Irena, Josh, Keith, Louiz, Mei, Steve, Susan, Tadd, Tian Run, Vivek for the pleasant working environment. My friends Bhanu, Khade, Reejis, Sreekar and Xiaojing for many an arbitrary chat sessions. Thanks are due to past and present administrative staff at Ocean Engineering Steve Malley, Kathy de Zengotita and Eda Daniel. I would also like to thank the so many unknown faces who have helped me in every aspect of my life in the last two years.

I can never forget my Parents T. Mukundan and B. Lali for their unconditional support and encouragement for 24 years without which I cannot even imagine (I don't want to imagine) where I would have been. Finally I thank the Almighty for guiding me through some of the most difficult and confusing times and to keep up my morale. This quote from Bible has been a source of constant encouragement for me: *He who dwells in the shelter of the Most High will rest in the shadow of the Almighty. I will say of the LORD, "He is my refuge and my fortress, my God, in whom I trust."*

Contents

| | |
|---|-----------|
| Abstract | 3 |
| Acknowledgments | 4 |
| 1 Introduction | 13 |
| 1.1 Motivation | 13 |
| 1.2 Solution Methods for RPP-RPP Surface Intersection | 14 |
| 1.3 Robust Solution Methods: A Brief Review | 15 |
| 1.4 Solution Method | 15 |
| 1.5 Assumptions | 16 |
| 1.6 Thesis Outline | 17 |
| 2 Review of Interval Methods | 19 |
| 2.1 Definition | 19 |
| 2.2 Basic Interval Operations | 20 |
| 2.2.1 Arithmetic Operations | 20 |
| 2.2.2 Trigonometric Operations | 20 |
| 2.2.3 Other Operations | 21 |
| 2.3 Properties of Interval Operations | 21 |
| 2.4 Interval-Valued Functions | 22 |
| 2.4.1 Basic Interval Functions | 22 |
| 2.4.2 Mean Value Theorem | 22 |
| 2.5 Interval Vectors and Matrices | 23 |
| 2.6 Rounded Interval Arithmetic | 23 |
| 3 Tracing a Surface-Surface Intersection | 25 |
| 3.1 Evaluation of Starting Points for Intersection | 26 |
| 3.1.1 Transversal Intersection | 26 |
| 3.1.2 Tangential Intersection and Multiplicity | 27 |
| 3.2 Interval ODEs for Surface Intersection | 27 |
| 3.2.1 Transversal Intersection | 28 |
| 3.2.2 Tangential Intersection | 29 |

| | | |
|----------|--|-----------|
| 4 | Nonlinear ODE Solvers for Marching | 33 |
| 4.1 | Problem Statement | 33 |
| 4.2 | Overview of Existing Methods | 33 |
| 4.2.1 | Runge-Kutta Method | 34 |
| 4.2.2 | Adams-Bashforth Method | 34 |
| 4.2.3 | Taylor Series Method | 35 |
| 4.3 | Uniqueness and Existence Theorems | 35 |
| 4.3.1 | Existence Theorem | 36 |
| 4.3.2 | Uniqueness Theorem | 36 |
| 4.4 | Conventional Solution Methods and Issues | 36 |
| 4.4.1 | Inherent Errors | 36 |
| 4.4.2 | Straying or Looping | 37 |
| 4.5 | Interval Nonlinear ODE Solvers | 38 |
| 4.5.1 | Advantages of an Interval ODE Solver | 38 |
| 5 | Validated ODE Solver in Tracing Surface-Surface Intersections | 41 |
| 5.1 | Overview of the Method | 41 |
| 5.2 | Phase I Algorithm | 42 |
| 5.3 | Phase II Algorithm | 44 |
| 5.3.1 | Interval Taylor Series Method | 44 |
| 5.3.2 | Interval Hermite Obreschkoff Method | 46 |
| 5.4 | Formulation Based on Validated ODE Solver | 47 |
| 5.5 | Resolving Singularities and Preventing Straying or Looping | 48 |
| 5.6 | Complexity Analysis | 49 |
| 6 | Automatic Differentiation | 51 |
| 6.1 | Introduction | 51 |
| 6.2 | Forward Mode Automatic Differentiation | 52 |
| 6.3 | Backward Mode Automatic Differentiation | 52 |
| 6.4 | Automatic Generation of Taylor Coefficients | 53 |
| 6.5 | Complexity Analysis | 54 |
| 7 | Error Bounds in Model Space | 57 |
| 7.1 | Mapping into Model Space | 57 |
| 7.2 | Intersection in Model Space and Reduction of Model Space Error Bound | 58 |
| 7.3 | Monotonic Control of Error Bounds | 60 |
| 7.3.1 | Conservative Relation | 62 |
| 7.3.2 | Approximate Relation | 65 |
| 7.3.3 | Controlling A Priori Enclosure in a Validated ODE Solver | 65 |
| 7.3.4 | Comparison of Two Methods | 66 |
| 7.4 | Complexity Analysis | 68 |

| | | |
|----------|---|-----------|
| 8 | Examples | 69 |
| 8.1 | Transversal Intersection of Surfaces | 69 |
| 8.1.1 | Intersection of two Bicubic Bézier Surfaces | 69 |
| 8.1.2 | Intersection of two Biquadratic Bézier Surfaces | 70 |
| 8.1.3 | Intersection with Singularity (Example: 1) | 72 |
| 8.1.4 | Intersection with Singularity (Example: 2) | 73 |
| 8.1.5 | Torus-Cylinder Intersection (Trigonometric Functions) | 75 |
| 8.2 | Tangential Intersection of Surfaces | 75 |
| 8.2.1 | Planar Intersection Curve | 75 |
| 8.2.2 | Non-Planar Intersection Curve | 77 |
| 8.3 | Self-Intersection of Surfaces | 80 |
| 8.3.1 | Self-Intersection of a Bicubic Bézier Patch | 80 |
| 8.4 | Resolving Straying and Looping | 80 |
| 8.4.1 | Example 1 Depicting Resolving Straying and Looping | 82 |
| 8.4.2 | Example 2 Depicting Resolving Straying and Looping | 86 |
| 8.5 | A Difficult Case | 86 |
| 9 | Conclusions | 91 |
| 9.1 | Conclusions | 91 |
| 9.2 | Recommendations for Future Research | 93 |
| A | Tables | 95 |

List of Figures

| | | |
|-----|---|----|
| 3-1 | A given intersection can have many components as depicted in this figure which was modified from [7]. | 25 |
| 3-2 | This figure illustrates transversal intersection of two surfaces. | 28 |
| 4-1 | Phenomenon of straying or looping. | 37 |
| 5-1 | Steps involved in validated scheme for solving ODEs depicted for the case involving a single dependent variable. | 43 |
| 5-2 | The series of <i>a priori enclosures</i> in parametric spaces which enclose the true intersection curve segment. | 47 |
| 7-1 | Mapping of the pre-image of the intersection curve segment from the parameter space to the model space. Note that the boxes obtained in the parameter space of each of the surface is continuous, gap free and ordered. | 59 |
| 7-2 | Depicts how we want to obtain tolerance in parametric space from tolerance in model space. | 60 |
| 7-3 | The flow chart representing control mechanism. | 61 |
| 7-4 | Mechanism to control the size of the <i>a priori enclosure</i> in a validated ODE solver. | 66 |
| 8-1 | Transversal intersection of two bicubic Bézier surfaces corresponding to a maximum relative model space error of 0.00350. | 71 |
| 8-2 | Transversal intersection of two tensor product Bézier surface patches. This figure depicts convergence of error bounds. | 71 |
| 8-3 | An example of transversal intersection with a singular point involving tracing four separate intersection curve segments. | 73 |
| 8-4 | Transversal intersection of a hyperbolic surface and a plane involving tracing four separate intersection curve segments. | 74 |
| 8-5 | Transversal intersection of a Torus and a Cylinder. | 76 |
| 8-6 | Dependence of error in parametric space on the width of starting point for a transversal intersection involving a Torus and a Cylinder. | 77 |
| 8-7 | Tangential intersection of two cubic-quadratic Bézier patches for a maximum relative model space error = 0.0050. Note that the control points of the surfaces are chosen such that the curve of intersection lies on a plane. | 78 |

| | | |
|------|---|----|
| 8-8 | Tangential intersection of tensor product Bézier surface patches. Control points of surfaces are chosen such that the curves of intersection do not lie on a plane. | 79 |
| 8-9 | Self-intersection of a bicubic Bézier patch. | 81 |
| 8-10 | Integration using <i>ode45</i> in Matlab. Looping is seen at the region close to the singularity in the σ, t parameter space. | 83 |
| 8-11 | Integration using <i>ode113</i> in Matlab. Straying and looping is seen at the region close to the singularity in the σ, t parameter space. | 83 |
| 8-12 | Integration using a validated ODE solver, not crossing the singular region in σ, t parameter space. | 84 |
| 8-13 | Result from <i>ode113</i> in Matlab in the σ, t parameter space. This experiment is done for the case of a small perturbation of one of the surface $[\mathbf{Q}_3](u, v)$ | 84 |
| 8-14 | Result from <i>ode45</i> in Matlab in the σ, t parameter space. This experiment is done for the case of a small perturbation of one of the surface $[\mathbf{Q}_3](u, v)$ | 85 |
| 8-15 | Result from a validated ODE solver in the σ, t parameter space for a case involving a small perturbation of one of the surface $[\mathbf{Q}_3](u, v)$ | 85 |
| 8-16 | Example depicting how validated ODE solver prevents straying and looping. Figure (a) shows the surface $[\mathbf{Q}_3](u, v)$ perturbed along the positive z-direction, the intersection curve segment is correctly traced by the validated ODE solver. Figure (b) in a similar way illustrates how the validated ODE solver successfully trace the correct intersection curve segment when the perturbation is in the negative z-direction. | 87 |
| 8-17 | Resolving straying and looping of curve of intersection for the intersection of a hyperbolic surface $[\mathbf{P}_4](u, v)$ and a plane $[\mathbf{Q}_4](u, v)$. Figure (a) shows the plane ($[\mathbf{Q}_4](u, v)$) perturbed along the positive z-direction, the intersection curve segment is correctly traced by the validated ODE solver. Figure (b) in a similar way illustrates how the validated ODE solver successfully trace the correct intersection curve segment when the perturbation is in the negative z-direction. | 88 |
| 8-18 | Figure shows an intersection where the intersection becomes difficult to solve as the governing differential equation fail. | 89 |

List of Tables

| | | |
|-----|--|----|
| 6.1 | Common Unary Operations | 53 |
| 6.2 | Common Binary Operations | 53 |
| 7.1 | The tolerance in parametric space of surface $[\mathbf{Q}_3](u, v)$ obtained from the given tolerance in model space. | 67 |
| 7.2 | The tolerance in parametric space of surface $[\mathbf{P}_1](\sigma, t)$ obtained from the given tolerance in model space. | 67 |
| 7.3 | The tolerance in parametric space of surface $[\mathbf{P}_3](\sigma, t)$ obtained from the given tolerance in model space. | 68 |
| 8.1 | Variation of the model space error with the number of steps for a bicubic Bézier intersection. | 70 |
| 8.2 | The effect of changing the width of the starting point for the number of steps required, the maximum relative model space error and the VNODE global error for a VNODE tolerance of 10^{-50} | 72 |
| 8.3 | The number of steps needed, the time taken for various VNODE tolerances to trace one of the four branches of the intersection of a hyperbolic surface and a plane. | 74 |
| 8.4 | Table comparing maximum relative model space error bound with the time taken, number of steps required and the VNODE tolerance for an intersection involving a torus and a cylinder. | 75 |
| 8.5 | Variation of the model space error with the number of steps for the tangential intersection of two surfaces. The intersection curve lies on a plane. | 78 |
| 8.6 | Variation of the model space error with the number of steps for the tangential intersection of two surfaces. The intersection curve does not lie on a plane. | 80 |
| 8.7 | Number of steps required for tracing the curve of intersection of the surface for different orders of the Taylor series. Note that the VNODE tolerance is kept constant at 1×10^{-25} | 81 |
| 8.8 | Resolving singularities of the curve of intersection for the intersection of a bi-cubic surface and a cubic-quadratic surface. Table shows the perturbations along the common normal and the corresponding number of steps needed to trace the intersection. | 86 |

| | | |
|-----|---|----|
| 8.9 | Resolving singularities of the curve of intersection for the intersection of a hyperbolic surface and a plane. We tabulate the perturbations along the common normal (z-axis) and the steps needed to trace the intersection. A constant VNODE tolerance of 1×10^{-20} was used. . . | 89 |
| A.1 | Interval Notations. | 96 |

Chapter 1

Introduction

1.1 Motivation

Surface to surface intersection is a fundamental process required to build and interrogate complex CAD models. It is needed in representing complex objects using the boundary representation (B-rep) method, in finite element discretization, computer animation, feature recognition, manufacturing simulation, numerically controlled machining, collision avoidance and scientific visualization for implicitly defined objects and for contouring multivariate functions that represent some properties of a system [28].

In general surface-surface intersection can be quite complicated. Intersection of two surfaces can have many components; open segments, closed loops and these components can also have a very complicated topological structure. Moreover the intersection can be a *transversal intersection*, a *tangential intersection* or it can result in an *overlap of surfaces*. A resolution of these different aspects of intersection is a necessity of the CAD/CAM community. These requirements could be listed objectively as:

- to obtain all components of the intersection,
- to estimate the components accurately and,
- to obtain a strict error bound for the above estimate.

A variety of methods, focusing mainly on numerical techniques have been developed (Section 1.2) to address these issues. But there are cases where completely catastrophic answers are returned without any warning, and this lack of robustness can cause severe topology violations as shown by Hu *et al.* [13]. This inconsistency prevails as a major impediment in the full automation of several design and manufacturing processes, and causes frequent system crashes, which to a large extent hinders productivity as described by Hoffmann [9, 10]. It is thus a challenge to realize the above goals within the constraints of available computational power and storage requirements.

Two types of surfaces are of main interest: *implicit algebraic* and *rational parametric*. An implicit algebraic (IA) surface is represented by a polynomial function defined as $f(\mathbf{r}) = 0$, where \mathbf{r} is the position vector of a point on the surface [29]. The rational polynomial parametric (RPP) type includes Bézier, rational Bézier, B-spline and NURBS surface patches [29], which are represented with two parameters σ and t as $\mathbf{P} = \mathbf{P}(\sigma, t)$, $0 \leq \sigma, t \leq 1$. These surfaces are popular in CAD/CAM and geometric design, and NURBS surfaces are chosen as the standard format in industry. Depending on the surfaces involved in intersection, we have three distinct classes: IA-IA, RPP-IA and RPP-RPP. The most frequent surface to surface intersection problem is the last one namely RPP-RPP, which is defined as follows:

$$\mathbf{P}(\sigma, t) \cap \mathbf{Q}(u, v), \quad (0 \leq \sigma, t \leq 1, \quad 0 \leq u, v \leq 1)$$

where $\mathbf{P}(\sigma, t) = \left(\frac{X_P(\sigma, t)}{W_P(\sigma, t)}, \frac{Y_P(\sigma, t)}{W_P(\sigma, t)}, \frac{Z_P(\sigma, t)}{W_P(\sigma, t)} \right)^T$ and $\mathbf{Q}(u, v) = \left(\frac{X_Q(u, v)}{W_Q(u, v)}, \frac{Y_Q(u, v)}{W_Q(u, v)}, \frac{Z_Q(u, v)}{W_Q(u, v)} \right)^T$. Formulation involves setting $\mathbf{P}(\sigma, t) = \mathbf{Q}(u, v)$ which leads to three nonlinear polynomial equations in four unknowns σ, t, u, v . This is an under-constrained system. The solutions are typically not isolated points but curves [30].

1.2 Solution Methods for RPP-RPP Surface Intersection

There are three major techniques for solving RPP-RPP surface intersections: *lattice methods*, *subdivision based methods* and *marching methods*. Detailed reviews can be found in [27, 28, 29].

Lattice Methods

Essentially, the lattice method reduces the dimensionality of the problem by computing intersections of a number of iso-parametric curves of one surface with the other surface followed by connecting the resulting discrete intersection points to form all the branches of the actual intersection [29]. This method has limitations in the sense that there is a possibility of missing components of the intersection, and the uncertainty in obtaining the correct topology of the intersection curves.

Subdivision Based Methods

They usually decompose the problem into simpler and similar problems which further reduce to a plane-plane intersection [29]. This is usually followed by a connection phase of the individual solutions to form the complete solution. Other subdivision based schemes include the *projected polyhedron* method employed by Sherbrooke *et al.* [39]. Subdivision methods on the other hand suffer from correct connectivity of the solution branches, missing of small loops and the presence of extraneous loops, and data proliferation.

Marching Methods

Marching methods involve generation of sequences of points of an intersection curve branch by stepping from a given point on the intersection curve in a direction prescribed by the local differential geometry [2, 3, 17, 48]. Marching methods formulate the surface intersection as an initial value problem (IVP) in the domain $0 \leq \sigma, t, u, v \leq 1$. However, such methods are by themselves *incomplete* in that they require *starting points* (initial conditions) for every branch of the solution. In this thesis, we focus on a marching method which is robust and efficient after locating starting points and performing a topological configuration of the intersection curves.

1.3 Robust Solution Methods: A Brief Review

It is well known that the problem of surface-surface intersection reduces to solving an initial value problem for ordinary differential equations (ODE) [29]. Conventional algorithms for solving a system of ODEs for example, Runge-Kutta method, Adams-Bashforth method or Taylor series method [32, 42], compute an estimate for an answer and perhaps its error estimate. Refer to Chapter 4 for a detailed treatment of solution schemes for IVPs and related issues. The user cannot tell how accurate the estimated answer may be without extensive error analysis.

The *Interval Projected Polyhedron* (IPP) algorithm [11] uses subdivision techniques coupled with robust interval arithmetic and exhaustively finds all intersection components. However as mentioned in Section 1.2, the *topology resolution* of computed intersection segments based on adjacency information is complicated. Moreover, the algorithm tends to be extremely time consuming for the case of tangential as well as higher order intersections. There is also no guarantee that the isolated intervals do contain a root, an inherent problem associated with any subdivision algorithm [12].

Grandine and Klein [7] formulate the intersection problem as a differential algebraic equation and determine the topology of the intersection curves so that it can be solved as a *boundary value problem* instead of an *initial value problem*. But the tracing of intersection curve is based on approximate methods, which compromises the robustness of the entire algorithm. Moreover the algorithm finds difficulty dealing with tangential intersections.

1.4 Solution Method

We develop a robust intersection algorithm which uses a validated interval ODE solving scheme for tracing the pre-image of the intersection given a robust evaluation of starting points in every branch of the intersection.

Interval techniques take into account three sources of errors in the numerical computation of solution to ODEs; propagation of *error in initial data*, *truncation error* caused by truncating infinite sequences of arithmetic operations after a finite number of steps and *round-off errors* inherent to computation in a floating point

environment [8]. When correctly used interval methods can compute bounds in which the correct answer is guaranteed to be enclosed [4].

The focus of the thesis is to investigate a robust marching method which will produce a continuous guaranteed bound on the error at each point using a validated interval ODE solving scheme [23]. Thus the same ODE solver can be used uniformly for both *transversal* and *tangential* intersections. An improvement for the special case of *self-intersection* is also discussed. We also relate the phenomenon of *straying* and *looping* to the criterion of a step size control based on the validation procedure in the method. The method described in this thesis enables us to realize the state of a gap-free boundary [34] using a marching scheme. We further develop tools to control this model space error.

1.5 Assumptions

Two arbitrary surfaces can intersect in a very complicated fashion. There is no limitation as to the general shape or nature of the surfaces which intersect. Given a very complicated system of space curves, we can find out surfaces, whose intersection results in the system of complicated space curves. But if we limit the class of surfaces, then we are able to predict some characteristics of the intersection curves. Hence we assume that the surfaces we use are:

1. **Regular:** The regularity of the surfaces requires the existence of a tangent plane everywhere on the surface, and the absence of self-intersections. Moreover it requires that the surfaces have neither *essential* nor *artificial* singularities [29]. The essential singularities arise from specific features of the surface geometry while the artificial singularities arise from the way in which we parametrize the surface [29].
2. **Continuous & Rational Polynomial Parametric (RPP):** Polynomial surfaces have special properties as shown by Hu *et al.* [11] with respect to overlap of surfaces over finite neighborhood. Hu *et al.* [11] describe some additional constraints on how RPP surfaces are limited in their behavior during tangential intersection. Moreover the use of RPP surfaces ensures that they are C^∞ continuous and rational [23]. This assumption is crucial for us to use the validated ODE solver. The assumption of a continuous surface is also needed for us to prove that the enclosures we obtain in the model space, contain the true curve of intersection.

At this point we note that obtaining a starting point or the topological configuration of an intersection curve is not the focus of the thesis. Hence we assume that we have identified and evaluated at least one starting point of each of the intersection curve segments and further a strict bound on the starting points. We also assume that we have isolated all singular points, a process which belongs to the step for obtaining the starting points. Thus our focus is on accurately tracing and finding a validated error bound in 3D model space for our intersection curve, given the type of intersection (eg:transversal or tangential) and a bound on the starting point.

1.6 Thesis Outline

The thesis is structured as follows: Chapter 2 briefly reviews interval arithmetic. In Chapter 3 we obtain the governing interval ODEs for the various intersection cases. Conventional ODE solvers and their limitations are discussed in Chapter 4 and in Chapter 5 we introduce the concept of a *validated ODE solver* [5, 26], its application in tracing surface intersection and discuss its use in preventing straying or looping and in resolving singularities. Chapter 6 describes *automatic differentiation* [24] which deals with a robust numerical technique for obtaining the Taylor coefficients and their derivatives. In Chapter 7 we develop a method to fulfill the condition of a continuous gap free boundary in the model space [23] and further a control mechanism to control this error bound. We perform various examples and tests using a prototype implementation of the algorithm in Chapter 8. Chapter 9 concludes this thesis with a review of the possible applications, issues and scope for future work. The notations used in this thesis is tabulated in Appendix.

Chapter 2

Review of Interval Methods

When we use a computer to make calculations involving real numbers, we have to use the finite set of floating-point numbers that the hardware makes available. In such a situation there are two main choices for the approximation of a real number by the floating-point number system. One choice is to represent a real number by using one floating-point number close enough to this real number. The second choice is to use two floating point numbers within which the original real number belongs. The latter is called an interval, and interval analysis was introduced by Moore [22] to allow digital computers working on floating point arithmetic to capture the errors automatically. Whenever an operation on real numbers is specified, the corresponding operation on their intervals is executed and a closed range which contains the resulting answer is returned.

The use of interval arithmetic is spreading out, mainly as a tool for so-called validated computations which guarantee that the solution is accurate within the bounds. It takes into account all possible sources of error from imprecise data to rounding errors due to floating point operations during computer calculations. In addition numerical techniques for ODEs based on interval arithmetic consider three sources of errors: (1) propagation of *error in initial data*, (2) *truncation error* caused by truncating infinite sequences of arithmetic operations after a finite number of steps and (3) *round-off errors* inherent to computation in floating point arithmetic [8]. When correctly used, interval methods can compute bounds in which the correct answer is guaranteed to be enclosed [4].

It is a well defined arithmetic system consisting of basic rules of operations. It was adopted into the area of *CAD* and *CAGD* a decade back. Interval geometries are defined by interval points [13], interval polynomial spline curves [36, 45] and interval spline surface patches [13, 45].

2.1 Definition

An interval number $[a]$ or $[\underline{a}, \bar{a}]$, is defined as the set of real numbers [22],

$$[a] \equiv [\underline{a}, \bar{a}] \equiv \{x | \underline{a} \leq x \leq \bar{a}\}, \quad \underline{a}, \bar{a}, x \in \mathbf{R}, \quad \underline{a} \leq \bar{a}, \quad (2.1)$$

where real numbers \underline{a} and \bar{a} refer to the lower and upper bounds, of the interval respectively.

A degenerate interval of the form $[a, a]$ is equivalent to the real number a . We denote the set of real numbers by \mathbf{R} and the set of interval numbers by \mathbf{IR} .

The centered form of an interval $[a]$ as used by Shen *et al.* [38] is given by,

$$[a] = m([a]) + \frac{1}{2}w([a])[I], \quad (2.2)$$

where $[I] = [-1, 1]$ and the operators $m([a])$ and $w([a])$ represent the midpoint and width of the interval $[a]$ respectively as defined by equation (2.3).

2.2 Basic Interval Operations

2.2.1 Arithmetic Operations

We define the following closed operations on two interval numbers $[a]$ and $[b]$,

$$\begin{aligned} [a] + [b] &= [\underline{a} + \underline{b}, \bar{a} + \bar{b}], \\ [a] - [b] &= [\underline{a} - \bar{b}, \bar{a} - \underline{b}], \\ [a][b] &= [\min\{\underline{a}\underline{b}, \bar{a}\bar{b}, \underline{a}\bar{b}, \bar{a}\underline{b}\}, \max\{\underline{a}\underline{b}, \bar{a}\bar{b}, \underline{a}\bar{b}, \bar{a}\underline{b}\}], \\ [a]/[b] &= [\min\{\underline{a}/\underline{b}, \bar{a}/\bar{b}, \underline{a}/\bar{b}, \bar{a}/\underline{b}\}, \max\{\underline{a}/\underline{b}, \bar{a}/\bar{b}, \underline{a}/\bar{b}, \bar{a}/\underline{b}\}], \quad 0 \notin [b]. \end{aligned}$$

2.2.2 Trigonometric Operations

For monotonically increasing functions (eg: $\exp a, \ln a$), we can obtain the interval arithmetic evaluation as:

$$f([a]) = [f(\underline{a}), f(\bar{a})].$$

For functions with a period, the lower bound and the upper bound are defined in a piecewise fashion, and depend on the slope. For trigonometric functions such as $\sin x$, we have:

$$\sin([\underline{a}, \bar{a}]) = [\underline{S}, \bar{S}], \text{ where,}$$

$$\underline{S} = \begin{cases} -1 & \text{if } \underline{a} \leq (2\pi n - \frac{\pi}{2}) \leq \bar{a} \quad \forall \quad n = 1, 2, \dots \\ \min(\sin \underline{a}, \sin \bar{a}), & \text{otherwise,} \end{cases}$$

$$\bar{S} = \begin{cases} 1 & \text{if } \underline{a} \leq (2\pi n + \frac{\pi}{2}) \leq \bar{a} \quad \forall \quad n = 1, 2, \dots \\ \max(\sin \underline{a}, \sin \bar{a}), & \text{otherwise.} \end{cases}$$

Similar definitions can be obtained for other trigonometric functions. A general

function is essentially subdivided into monotonic segments and we evaluate the bound for each of these segments.

2.2.3 Other Operations

For an interval $[a]$, we define width $w([a])$ midpoint $m([a])$ and magnitude $|[a]|$ of $[a]$ as,

$$\begin{aligned} w([a]) &= \bar{a} - \underline{a}, \\ m([a]) &= \frac{\bar{a} + \underline{a}}{2}, \\ |[a]| &= \max(|\bar{a}|, |\underline{a}|). \end{aligned} \tag{2.3}$$

Width and midpoint are defined component-wise for interval vectors and matrices.

2.3 Properties of Interval Operations

We have the inclusion of intervals,

$$[a] \subseteq [b] \quad \Leftrightarrow \quad \underline{a} \geq \underline{b} \quad \text{and} \quad \bar{a} \leq \bar{b}.$$

The interval arithmetic operations are *inclusion monotone*. That is, for real intervals $[a]$, $[a_1]$, $[b]$ and $[b_1]$, such that $[a] \subseteq [a_1]$ and $[b] \subseteq [b_1]$,

$$[a] \circ [b] \subseteq [a_1] \circ [b_1], \quad \circ \in \{+, -, \times, \div\}.$$

Interval addition and multiplication are associative and commutative, but the distributive law does not hold in general. That is, we can find three intervals $[a]$, $[b]$ and $[c]$ for which,

$$[a]([b] + [c]) \neq [a][b] + [a][c].$$

However, for any three intervals $[a]$, $[b]$ and $[c]$, the *sub-distributive law* holds,

$$[a]([b] + [c]) \subseteq [a][b] + [a][c]$$

and this reduces to the *distributive law*, for the cases that $[b][c] \geq 0$, if $[a]$ is a degenerate interval, or if $[b]$ and $[c]$ are symmetric. In particular, for some $\eta \in \mathbf{R}$, and intervals $[b]$ and $[c]$, we have

$$\eta([b] + [c]) = \eta[b] + \eta[c].$$

Using the basic interval arithmetic operations one can easily show that for any intervals $[a]$ and $[b]$ and degenerate interval η

$$w([a] \pm [b]) = w([a]) + w([b]),$$

$$w(\eta[a]) = |\eta|w([a]).$$

Some other useful results are

$$\begin{aligned} |[a] + [b]| &\leq |[a]| + |[b]|, \\ |[a][b]| &= |[a]||[b]|, \\ w([a] \pm [b]) &= w([a]) + w([b]), \\ w([a][b]) &\geq \max(|[b]|w([a]), |[a]|w([b])), \\ w([a][b]) &\leq |[b]|w([a]) + |[a]|w([b]). \end{aligned}$$

2.4 Interval-Valued Functions

2.4.1 Basic Interval Functions

A real continuous function on $D \subseteq \mathbf{R}^n$ is defined as,

$$f : \mathbf{R}^n \rightarrow \mathbf{R}.$$

The range of f over an interval $[s] \subseteq D$ is defined by,

$$R(f; [s]) = \{f(x) | x \in [s]\}.$$

The evaluation of f on $[s] \subseteq D$ in interval arithmetic, which we denote by $f([s])$, is obtained by replacing each occurrence of a real variable with a corresponding interval, by replacing the standard functions with enclosures of their ranges, and performing interval arithmetic operations instead of the real operations [26]. It follows from the inclusion monotone property of interval operations that the range of f , $R(f; [s])$, is always contained in the interval arithmetic evaluation $f([s])$ [26].

Moore [22] also proves the continuity of rational interval functions. We note that $f([s])$ need not be unique. It depends on the arrangement of the interval expressions. Thus, rearrangement of an interval expression may lead to tighter bounds. Moore [22] shows that evaluation based on the *centered form* in (2.2) may provide a sharper bound than the standard form (2.1) for a given expression. We will be using these concepts when we obtain strict bounds on the intersection in the 3D model space.

2.4.2 Mean Value Theorem

If f is continuously differentiable on D , and $[a] \subseteq D$, then, for any y and $b \in [a]$, $f(y) = f(b) + f'(\eta)(y - b)$ for some $\eta \in [a]$,

$$f(y) \in f_m([a], b) \equiv f(b) + f'([a])([a] - b).$$

This is the interval arithmetic form of the *mean value theorem* [22]. The mean value form, $f_m([a], b)$, is extremely popular in interval methods as it is analogous to the centered form, and hence gives tighter enclosures for the range of f [22].

Beyond the use of the centered form, the *modified affine arithmetic* [40] can be employed for generation of tighter bounds of functions.

2.5 Interval Vectors and Matrices

The above definitions for interval numbers can be extended to vectors and matrices, too. By an *interval vector* we mean a vector with interval components and by an *interval matrix* we mean a matrix with interval components [24]. The arithmetic operations involving interval vectors and matrices are defined by the standard formulae, except that interval numbers replace the real numbers and real arithmetic is replaced by interval arithmetic in the associated computations. Following the standard notations, vectors and matrices are denoted by bold type throughout the thesis. Interval vectors and interval matrices are denoted by bold type enclosed in a square bracket.

The corresponding vector or matrix equations also hold when $[a]$ and $[b]$ are interval vectors. If \mathbf{A} is an $n \times n$ real matrix and $[\mathbf{a}]$ is an n dimensional interval vector, then

$$w(\mathbf{A}[a]) = |\mathbf{A}|w([a]),$$

where $|\mathbf{A}|$ is obtained by taking absolute values on each component of \mathbf{A} .

Inclusion in the case of interval vectors and matrices are defined component-wise by,

$$[\mathbf{A}] \subseteq [\mathbf{B}] \Leftrightarrow [a_{ij}] \subseteq [b_{ij}] \quad \forall \quad i, j \in \mathbf{I}.$$

The maximum norms of an interval vector $[\mathbf{a}]$ and an interval matrix $[\mathbf{A}]$ are respectively given by,

$$\begin{aligned} \|[\mathbf{a}]\| &= 1 \leq i \leq n (|[a_i]|) \\ \|[\mathbf{A}]\| &= 1 \leq i \leq n \left(\sum_{j=1}^n |[a_{ij}]| \right) \end{aligned}$$

The equivalent vector form of the mean value theorem is hence,

$$\mathbf{f}(\mathbf{y}) \in \mathbf{f}([\mathbf{a}], \mathbf{b}) \equiv \mathbf{f}(\mathbf{b}) + \mathbf{J}(\mathbf{f}([\mathbf{a}], \mathbf{b}))([\mathbf{a}] - \mathbf{b}),$$

where $\mathbf{J}(\mathbf{f}([\mathbf{a}], \mathbf{b}))$ represents the Jacobian matrix of the vector \mathbf{f} .

2.6 Rounded Interval Arithmetic

Interval arithmetic based on floating point numbers does not guarantee conservative bound during computation [29]. Rounded interval arithmetic [1] ensures that the computed interval always contains the exact interval as shown below.

$$[a] + [b] = [\underline{a} + \underline{b} - \varepsilon_l, \bar{a} + \bar{b} + \varepsilon_u],$$

$$\begin{aligned}
[a] - [b] &= [\underline{a} - \bar{b} - \varepsilon_l, \bar{a} - \underline{b} + \varepsilon_u], \\
[a][b] &= [\min\{\underline{a}\underline{b}, \bar{a}\bar{b}, \underline{a}\bar{b}, \bar{a}\underline{b}\} - \varepsilon_l, \max\{\underline{a}\underline{b}, \bar{a}\bar{b}, \underline{a}\bar{b}, \bar{a}\underline{b}\} + \varepsilon_u], \\
[a]/[b] &= [\min\{\underline{a}/\underline{b}, \bar{a}/\bar{b}, \underline{a}/\bar{b}, \bar{a}/\underline{b}\} - \varepsilon_l, \max\{\underline{a}/\underline{b}, \bar{a}/\bar{b}, \underline{a}/\bar{b}, \bar{a}/\underline{b}\} + \varepsilon_u], \quad 0 \notin [b],
\end{aligned}$$

where ε_l and ε_u are the *units-in-last place* denoted by ulp_l and ulp_u for each separate floating point number resulting from the floating point operations. For simplicity of the discussion we safely assume that unless specifically indicated we use exact interval arithmetic in this thesis.

Chapter 3

Tracing a Surface-Surface Intersection

A given intersection curve represents a continuous trajectory in the parameter space of each of the surfaces. An intersection curve segment in the model space as shown in the Figure 3-1 has a counter part in the parametric space of each of the surfaces.

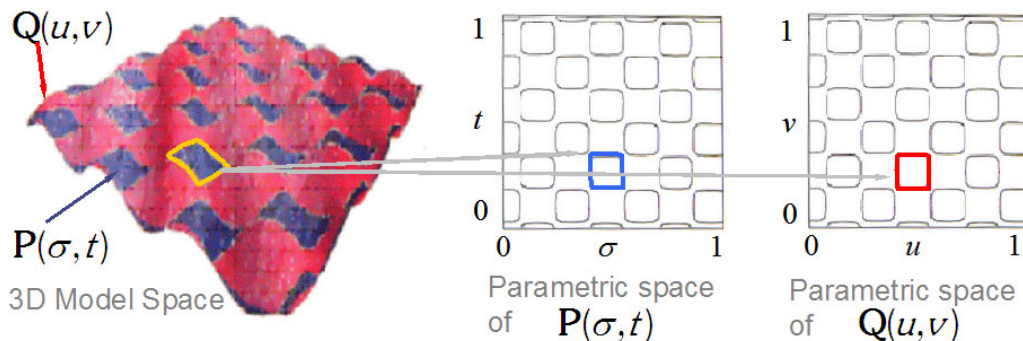


Figure 3-1: A given intersection can have many components as depicted in this figure which was modified from [7].

As shown in the Figure 3-1, the given intersection can have many components, even for surfaces which are relatively easy to represent. Some of these components may be open segments, or closed loops, and these segments can come quite close to each other. It is thus a challenge to:

- Identify each of the many segments.
- Obtain a starting point on each of the segments.
- Trace the intersection exactly or within strict error bounds.

In this chapter we formulate the surface to surface intersection (SSI) as an initial value problem (IVP). An IVP is defined if we are given the differential equations governing the problem and further the corresponding initial conditions. For the case of intersection of two surfaces the governing differential equations are a system of

ODEs. Hence this chapter will deal with obtaining the ODEs which govern the intersection. Depending on the type of intersection namely transversal or tangential we need to obtain separate ODEs. We further formulate these ODEs in interval arithmetic to obtain an interval ODE system.

3.1 Evaluation of Starting Points for Intersection

Though the focus of the thesis is not on obtaining the starting point in each intersection curve segment, we roughly describe how we plan to identify and further obtain the starting points which correspond to initial conditions. Thus we assume that we have identified different components of the intersection and have obtained at least one starting point on each segment.

3.1.1 Transversal Intersection

For identifying all connected components of the intersection a set of special points on the intersection curve can be defined. Such a set typically includes *border points*, *turning points*, *singular points* and *collinear normal points* of the intersection. These provide at least one point on any connected intersection segment or closed loops and identify all singularities [28].

Border points are points on the intersection at which at least one of the parametric variables σ, t, u, v takes a value equal to the border of the $\sigma - t$ or $u - v$ parametric domain. Computing the border points involves solving a *curve-surface* intersection such as an equation of type, $[\mathbf{P}](\sigma, 0) = [\mathbf{Q}](u, v)$. This system is solved robustly using the *interval projected polyhedron algorithm* (IPP) [29].

Turning point and singular point computation involves the first partial derivatives. Turning points essentially involve solving a system of three nonlinear polynomial equations of three variables, and computing singularities reduces to solving an over constrained system of three nonlinear polynomial equations of two variables. For the solution we use IPP algorithm.

Closed loops are usually a special case of transversal intersection, but can be more complicated. The difficulty is to recognize the presence of the closed loop, and once the presence is identified, we need a starting point for tracing the intersection curve. Collinear normal points are a subset of parallel normal points first used by Sinha *et al.* [41] in surface intersection loop detection methods. Sederberg *et al.* [35] recognized the importance of collinear normal points in detecting the existence of closed intersection loops in intersection problems of two distinct parametric surface patches. These are points on the two parametric surfaces at which the normal vectors are collinear. The collinear normal points satisfy the following equations [35].

$$\begin{aligned}
 ([\mathbf{P}_s] \times [\mathbf{P}_t]) \cdot [\mathbf{Q}_u] &\subset [ZERO], \\
 ([\mathbf{P}_s] \times [\mathbf{P}_t]) \cdot [\mathbf{Q}_v] &\subset [ZERO], \\
 ([\mathbf{P}] - [\mathbf{Q}]) \cdot [\mathbf{P}_s] &\subset [ZERO], \\
 ([\mathbf{P}] - [\mathbf{Q}]) \cdot [\mathbf{P}_t] &\subset [ZERO],
 \end{aligned} \tag{3.1}$$

where $[ZERO]$ denotes a sufficiently small interval containing zero. The system of *four* nonlinear equations (3.1) in *four* unknowns σ, t, u and v , should be solved robustly and then supplied as the starting points for the tracing algorithm.

An alternate way to detect closed intersection loops is to use topological methods. Bounding pyramids can be used to ensure the nonexistence of closed loops in surface to surface intersection [29]. Robust evaluation in this context thus reduces to solving the above system of equations (3.1) using algorithms such as *interval projected polyhedron algorithm*.

3.1.2 Tangential Intersection and Multiplicity

Obtaining at least one starting point in each intersection curve segment for a tangential intersection is not an easy task. This is because of a variety of complex shapes the intersection curve can follow. But if we assume rational polynomial surfaces, then we can use a theorem by Hu *et al.* [11] for simplification.

Theorem 1 *If a tangential contact curve of two polynomial surfaces does not contain a loop then they must start from a border point and end at another border point.*

This implies that if two surfaces are polynomials then there are two cases.

Case 1: The intersection can start at a border and end at another border point.

Case 2: The intersection contains at least one loop.

This theorem says that if the tangential intersection curve contains a loop, then inside the loop, there must be a collinear normal point which is not an intersection point of those two surfaces [11]. We at this point note that this theorem can be applied only to ideal mathematical surfaces. Application of this theorem however in the context of floating point arithmetic requires more study.

3.2 Interval ODEs for Surface Intersection

The intersection of two interval parametric surfaces $[\mathbf{P}](\sigma, t)$ and $[\mathbf{Q}](u, v)$ can be described as an interval vector equation given by,

$$[\mathbf{P}](\sigma, t) = [\mathbf{Q}](u, v). \quad (3.2)$$

We can reformulate equation (3.2) as a system of *ordinary differential equations*(ODE) which are arc length parametrized. Our approach is to use a marching scheme to find out the curve of intersection by solving this system of interval ODEs obtained by Hu *et al.* [11],

$$\begin{aligned} \sigma' &= \frac{d\sigma}{ds} = \frac{Det([\mathbf{c}], [\mathbf{P}_t], [\mathbf{N}^{\mathbf{P}}])}{[\mathbf{N}^{\mathbf{P}}] \cdot [\mathbf{N}^{\mathbf{P}}]}, & t' &= \frac{dt}{ds} = \frac{Det([\mathbf{P}_\sigma], [\mathbf{c}], [\mathbf{N}^{\mathbf{P}}])}{[\mathbf{N}^{\mathbf{P}}] \cdot [\mathbf{N}^{\mathbf{P}}]}, \\ u' &= \frac{du}{ds} = \frac{Det([\mathbf{c}], [\mathbf{Q}_v], [\mathbf{N}^{\mathbf{Q}}])}{[\mathbf{N}^{\mathbf{Q}}] \cdot [\mathbf{N}^{\mathbf{Q}}]}, & v' &= \frac{dv}{ds} = \frac{Det([\mathbf{Q}_u], [\mathbf{c}], [\mathbf{N}^{\mathbf{Q}}])}{[\mathbf{N}^{\mathbf{Q}}] \cdot [\mathbf{N}^{\mathbf{Q}}]}, \end{aligned} \quad (3.3)$$

where Det denotes the determinant and,

$$[\mathbf{N}^{\mathbf{P}}] = [\mathbf{P}_{\sigma}] \times [\mathbf{P}_t], \quad [\mathbf{N}^{\mathbf{Q}}] = [\mathbf{Q}_u] \times [\mathbf{Q}_v],$$

are the normal vectors of $[\mathbf{P}]$ and $[\mathbf{Q}]$ respectively. $[\mathbf{c}]$ is the marching direction s is the arc length parameter.

Equations (3.3) are true for any surface-surface intersection involving parametrically defined surfaces, provided we correctly represent the marching direction (tangent to the intersection curve), and the surfaces and their derivatives. Based on the intersection type the marching direction has to be computed differently.

3.2.1 Transversal Intersection

For a transversal intersection, the direction of marching $[\mathbf{c}]$, is perpendicular to the normal vectors of both surfaces (refer Figure 3-2). This direction can be obtained as follows [11]:

$$[\mathbf{c}] = \pm \frac{[\mathbf{N}^{\mathbf{P}}] \times [\mathbf{N}^{\mathbf{Q}}]}{\|[\mathbf{N}^{\mathbf{P}}] \times [\mathbf{N}^{\mathbf{Q}}]\|}. \quad (3.4)$$

$$\begin{aligned} \text{Green arrow} \quad & \mathbf{N}^{\mathbf{P}} = \mathbf{P}_{\sigma} \times \mathbf{P}_t \quad \text{Normal to } \mathbf{P}(\sigma, t) \\ \text{Blue arrow} \quad & \mathbf{N}^{\mathbf{Q}} = \mathbf{Q}_u \times \mathbf{Q}_v \quad \text{Normal to } \mathbf{Q}(u, v) \\ \text{Red arrow} \quad & \mathbf{c} = \frac{\mathbf{N}^{\mathbf{P}} \times \mathbf{N}^{\mathbf{Q}}}{\|\mathbf{N}^{\mathbf{P}} \times \mathbf{N}^{\mathbf{Q}}\|} \end{aligned}$$

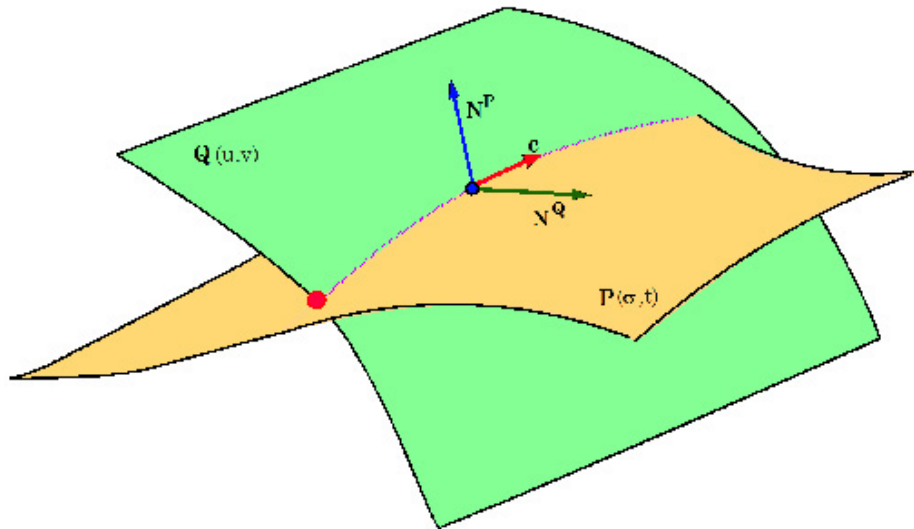


Figure 3-2: This figure illustrates transversal intersection of two surfaces.

Self-Intersection

Differential equations for tracing self-intersection curves are formulated such that the curve of self-intersection is arc length parametrized. The marching direction coincides with the tangential direction of the self-intersection curve $[\mathbf{c}]$ of the surface which is perpendicular to the two normal vectors $[\mathbf{N}^{\mathbf{P}}](s, t)$ and $[\mathbf{N}^{\mathbf{P}}](u, v)$ where $(s, t) \neq (u, v)$. The marching direction can hence be written as,

$$[\mathbf{c}] = \frac{[\mathbf{N}^{\mathbf{P}}](s, t) \times [\mathbf{N}^{\mathbf{P}}](u, v)}{|[\mathbf{N}^{\mathbf{P}}](s, t) \times [\mathbf{N}^{\mathbf{P}}](u, v)|}. \quad (3.5)$$

Robust computation of the curve of self-intersection is usually based on IPP [29]. It is very inefficient to solve surface intersection problems with IPP, as the key difficulty here being the removal of the trivial solutions from the real solutions $(s, t) = (u, v)$. The advantage of the marching scheme we propose is that we do not have to remove the trivial solutions and that it fits into our concept of a uniform approach, where we use the same set of differential equations with appropriate changes for the type of intersection.

3.2.2 Tangential Intersection

Obtaining the marching direction for tangential intersection is based on Ye and Maekawa [49] and they use the higher derivatives of the surfaces involved to compute it. Note that we cannot use equation (3.4) to get the marching direction because normals to both surfaces are parallel.

The unit tangent vector $[\mathbf{c}]$ must lie on the common tangent plane of $[\mathbf{P}](\sigma, t)$ and $[\mathbf{Q}](u, v)$. The tangent plane can be defined using the linear combination of the partial derivatives ($[\mathbf{P}_\sigma]$, $[\mathbf{P}_t]$, $[\mathbf{Q}_u]$ and $[\mathbf{Q}_v]$) of each of the surfaces $[\mathbf{P}](\sigma, t)$ and $[\mathbf{Q}](u, v)$, i.e.

$$[\mathbf{c}] = [\mathbf{P}_\sigma]\sigma' + [\mathbf{P}_t]t' = [\mathbf{Q}_u]u' + [\mathbf{Q}_v]v'. \quad (3.6)$$

Using the concept of a curve on a surface, and using the fact that the normals of the surfaces are the same denoted by $[\mathbf{N}]$, we can show that the normal curvatures of both surfaces are equal, which can be rewritten in terms of the *second fundamental form coefficients* of both surfaces ($[L^{\mathbf{P}}]$, $[M^{\mathbf{P}}]$, $[N^{\mathbf{P}}]$ and $[L^{\mathbf{Q}}]$, $[M^{\mathbf{Q}}]$, $[N^{\mathbf{Q}}]$) as,

$$[L^{\mathbf{P}}](\sigma')^2 + 2[M^{\mathbf{P}}]\sigma't' + [N^{\mathbf{P}}](t')^2 = [L^{\mathbf{Q}}](u')^2 + 2[M^{\mathbf{Q}}]u'v' + [N^{\mathbf{Q}}](v')^2. \quad (3.7)$$

This is a quadratic equation in (σ', t', u', v') . By taking the cross product of both sides of equation (3.6) with $[\mathbf{Q}_u]$ and $[\mathbf{Q}_v]$, and projecting the resulting equations onto the common surface normal vector $[\mathbf{N}]$, u' and v' can be represented as the following linear combination of σ' and t' :

$$u' = [a_{11}]\sigma' + [a_{12}]t', \quad (3.8)$$

$$v' = [a_{21}]\sigma' + [a_{22}]t', \quad (3.9)$$

where we obtain $[a_{11}]$, $[a_{12}]$, $[a_{21}]$ and $[a_{22}]$ as follows:

$$\begin{aligned} [a_{11}] &= \frac{([\mathbf{P}_\sigma] \times [\mathbf{Q}_v]) \cdot [\mathbf{N}]}{([\mathbf{Q}_u] \times [\mathbf{Q}_v]) \cdot [\mathbf{N}]} = \frac{Det([\mathbf{P}_\sigma], [\mathbf{Q}_v], [\mathbf{N}])}{\sqrt{[E^Q][G^Q] - ([F^Q])^2}}, \\ [a_{12}] &= \frac{([\mathbf{P}_t] \times [\mathbf{Q}_v]) \cdot [\mathbf{N}]}{([\mathbf{Q}_u] \times [\mathbf{Q}_v]) \cdot [\mathbf{N}]} = \frac{Det([\mathbf{P}_t], [\mathbf{Q}_v], [\mathbf{N}])}{\sqrt{[E^Q][G^Q] - ([F^Q])^2}}, \\ [a_{21}] &= \frac{([\mathbf{Q}_u] \times [\mathbf{P}_\sigma]) \cdot [\mathbf{N}]}{([\mathbf{Q}_u] \times [\mathbf{Q}_v]) \cdot [\mathbf{N}]} = \frac{Det([\mathbf{Q}_u], [\mathbf{P}_\sigma], [\mathbf{N}])}{\sqrt{[E^Q][G^Q] - ([F^Q])^2}}, \\ [a_{22}] &= \frac{([\mathbf{Q}_u] \times [\mathbf{P}_t]) \cdot [\mathbf{N}]}{([\mathbf{Q}_u] \times [\mathbf{Q}_v]) \cdot [\mathbf{N}]} = \frac{Det([\mathbf{Q}_u], [\mathbf{P}_t], [\mathbf{N}])}{\sqrt{[E^Q][G^Q] - ([F^Q])^2}}. \end{aligned}$$

Here $([E^Q], [G^Q], [F^Q])$ are the *first fundamental form coefficients* of the surface $[\mathbf{Q}]$.

Substituting (3.8) and (3.9) into (3.7), then we obtain a quadratic equation of the form,

$$[b_{11}](\sigma')^2 + 2[b_{12}](\sigma')(t') + [b_{22}](t')^2 = 0, \quad (3.10)$$

where,

$$\begin{aligned} [b_{11}] &= [a_{11}]^2[L^Q] + 2[a_{11}][a_{21}][M^Q] + [a_{21}]^2[N^Q] - [L^P], \\ [b_{12}] &= [a_{11}][a_{12}][L^Q] + ([a_{11}][a_{22}] + [a_{21}][a_{12}])[M^Q] + [a_{21}][a_{22}][N^Q] - [M^P], \\ [b_{22}] &= [a_{12}]^2[L^Q] + 2[a_{12}][a_{22}][M^Q] + [a_{22}]^2[N^Q] - [N^P]. \end{aligned}$$

There are four distinct cases to the solution of (3.10) depending upon the discriminant ($[d] = [b_{12}]^2 - [b_{11}][b_{22}]$).

- ($\bar{d} < 0$): The surfaces have an isolated tangential contact point.
- ($\bar{d} > 0$): We have the phenomenon of branching, i.e. $[\mathbf{c}]$ is not uniquely defined.
- ($0 \in [d]$ and $0 \in [b_{11}], [b_{12}], [b_{22}]$): The intersection of surfaces $[\mathbf{P}]$ and $[\mathbf{Q}]$ cannot be evaluated by this method or they have a contact of at least second order (i.e., curvature continuous).
- ($0 \in [d]$ and $0 \notin [b_{11}]^2 + [b_{12}]^2 + [b_{22}]^2$): The marching direction vector is defined. Thus, $[\mathbf{P}]$ and $[\mathbf{Q}]$ are said to intersect tangentially at the neighborhood.

The marching direction is obtained, depending on $[b_{11}]$, $[b_{12}]$ and $[b_{22}]$, as follows. If $0 \notin [b_{11}]$, $\frac{\sigma'}{t'} = [\nu] = -\frac{[b_{12}]}{[b_{11}]}$, the marching direction is given by,

$$[\mathbf{c}] = \frac{[\nu][\mathbf{P}_\sigma] + [\mathbf{P}_t]}{|[\nu][\mathbf{P}_\sigma] + [\mathbf{P}_t]|}. \quad (3.11)$$

If $0 \in [b_{11}]$ and $0 \notin [b_{22}]$, $\frac{t'}{\sigma'} = [\mu] = -\frac{[b_{12}]}{[b_{22}]}$, then the marching direction is given by,

$$[\mathbf{c}] = \frac{[\mathbf{P}_\sigma] + [\mu][\mathbf{P}_t]}{|[\mathbf{P}_\sigma] + [\mu][\mathbf{P}_t]|}. \quad (3.12)$$

Chapter 4

Nonlinear ODE Solvers for Marching

4.1 Problem Statement

Tracing the intersection of two RPP surfaces using a marching method essentially reduces to solving a system of ordinary differential equations (ODEs) (3.3). Given initial conditions, which correspond to a starting point, we can in principle integrate the system of ODEs to obtain a series of points in the parameter space of each of the surfaces which represent an approximation to the intersection curve segment.

The system of ordinary differential equations (3.3) with the initial condition represents an initial-value problem (IVP), which can be written in vector form as,

$$\mathbf{y}'(s) = \mathbf{f}(\mathbf{y}(s)), \quad \mathbf{y}(s_0) = \mathbf{y}_0,$$

where,

$$\mathbf{y}'(s) = \begin{bmatrix} \sigma' & t' & u' & v' \end{bmatrix}^T \quad \text{and} \quad \mathbf{y}_0 = \begin{bmatrix} \sigma_0 & t_0 & u_0 & v_0 \end{bmatrix}^T.$$

The system of ODEs with starting point is a regular, autonomous IVP. Our use of rational polynomial parametric surfaces which are C^∞ continuous makes sure that $\mathbf{f}(\mathbf{y}(s))$ is well behaved and is at least C^k continuous, where k is defined in Chapter 5. The initial conditions for the ODEs are obtained by solving a system of nonlinear polynomial equations if the surfaces are RPP. This is usually done numerically and has an associated error.

4.2 Overview of Existing Methods

Any numerical scheme, yielding a solution for a physical system represented by an IVP should first check for the *existence* and then the *uniqueness* of the solution before returning an approximation, or a bound for it [16]. The similar idea of existence and uniqueness is applied while solving a system of linear equations having many

unknowns. This, however, is not a common practice in the conventional solution schemes for IVPs.

A typical solution procedure is to use an approximate, point based algorithm [32] like Runge-Kutta method, Taylor series method or Adams-Bashforth technique for solving the ODEs corresponding to the surface-surface intersection problem mentioned in [28, 29] at discrete values of the arc length parameter s .

4.2.1 Runge-Kutta Method

The scheme of Runge Kutta method [16] is as follows, we define the following intermediate variables for the j^{th} step

$$\begin{aligned} \mathbf{k}_1 &= h_j \mathbf{f}(s, \mathbf{y}_j), \\ \mathbf{k}_2 &= h_j \mathbf{f}\left(s + \frac{h_j}{2}, \mathbf{y}_j + \frac{\mathbf{k}_1}{2}\right), \\ \mathbf{k}_3 &= h_j \mathbf{f}\left(s + \frac{h_j}{2}, \mathbf{y}_j + \frac{\mathbf{k}_2}{2}\right), \\ \mathbf{k}_4 &= h_j \mathbf{f}(s + h_j, \mathbf{y}_j + \mathbf{k}_3), \\ \mathbf{y}(s_{j+1}) &= \mathbf{y}(s_j) + \left(\frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6}\right), \\ s_{j+1} &= s_j + h_j. \end{aligned}$$

This method is easy to implement on a computer and is accurate up to 4th order, namely the error per step is on the order of h^5 .

4.2.2 Adams-Bashforth Method

Adams-Bashforth method is a predictor-corrector method. Hence it is done in two main steps.

Step I : The predictor step

- Fit a cubic interpolating polynomial to the function $\mathbf{f}(s, \mathbf{y})$ through the points $\mathbf{y}_j, \mathbf{y}_{j-1}, \mathbf{y}_{j-2}$ and \mathbf{y}_{j-3} . We can see that 4 points are needed to fit a 3rd order polynomial.
- Integrate this simple function over the interval \mathbf{y}_j to \mathbf{y}_{j+1} , giving

$$\mathbf{y}_{j+1}^* = \mathbf{y}_j + \frac{h_j}{24}(55\mathbf{f}(\mathbf{y}_j) - 59\mathbf{f}(\mathbf{y}_{j-1}) + 37\mathbf{f}(\mathbf{y}_{j-2}) - 9\mathbf{f}(\mathbf{y}_{j-3})). \quad (4.1)$$

Step II : The corrector step

- Fit a cubic polynomial $\mathbf{f}(s, \mathbf{y})$ through the points $\mathbf{y}_{j+1}, \mathbf{y}_j, \mathbf{y}_{j-1}$, and \mathbf{y}_{j-2} .

- Integrate the resultant polynomial giving,

$$\mathbf{y}_{j+1} = \mathbf{y}_j + \frac{h_j}{24}(9\mathbf{f}^*(\mathbf{y}_{j+1}) + 19\mathbf{f}(\mathbf{y}_j) - 5\mathbf{f}(\mathbf{y}_{j-1}) + \mathbf{f}(\mathbf{y}_{j-2})). \quad (4.2)$$

where $\mathbf{f}^*(\mathbf{y}_{j+1}) = f$ evaluated at \mathbf{y}_{j+1}^* .

Algorithm

- Use RK method to determine $\mathbf{y}_3, \mathbf{y}_2$ and \mathbf{y}_1 . \mathbf{y}_0 is known.
- Calculate \mathbf{y}_{j+1}^* using (4.1).
- Evaluate $\mathbf{f}^*(\mathbf{y}_{j+1})$ and calculate \mathbf{y}_{j+1} using equation (4.2).
- Increment the parameter $s_{j+1} = s_j + h_j$.

The Adams-Bashforth method is also of 4^{th} order and hence error is of order h^5 . This method is faster than the 4^{th} order RK method since only two new functional evaluations are needed in each step. This method is however not self starting. Therefore RK method is usually used to initiate this multi-step method [32].

4.2.3 Taylor Series Method

Taylor's series [32, 16] in a single variable s is given by,

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h_j\mathbf{f}(\mathbf{y}_j) + \frac{h_j^2}{2!}\mathbf{f}'(\mathbf{y}_j) + \dots$$

A Taylor's formula is written as,

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h_j\mathbf{f}(\mathbf{y}_j) + \frac{h_j^2}{2!}\mathbf{f}'(\mathbf{y}_j) + \dots + \frac{h_j^k}{k!}\mathbf{f}^k(\mathbf{y}_j) + \mathbf{R}_j^k,$$

where the remainder term (error term) \mathbf{R}_j^k is given by,

$$\mathbf{R}_j^k = \frac{h_j^{k+1}}{(k+1)!}\mathbf{f}^{k+1}(\mathbf{y}_j^*) \text{ for some } \mathbf{y}_j^* = \mathbf{y}(s^*) \text{ such that } s^* \in [s_j, s_{j+1}].$$

The usual practice is to truncate the Taylor's formula after k terms to obtain a very good approximation to the solution. Thus we obtain the Taylor's method of order $k+1$.

4.3 Uniqueness and Existence Theorems

Solution of an IVP for a nonlinear system of ODEs can have 3 different cases.

1. No solution.
2. Exactly one solution.

3. More than one solution.

These lead to the following fundamental questions.

Existence: Under what conditions does an initial value problem have at least one solution.

Uniqueness: Under what conditions does that problem have a unique solution, only one solution.

The theorems which state the conditions are called the *existence theorem* and *uniqueness theorem* respectively. Uniqueness is of importance, for instance, if we attempt to predict the future behavior of a physical system governed by an initial value problem. Our model may be complicated, so that we have to apply a numerical method for obtaining an approximate solution. But before doing so, we should make sure that the model will yield a unique solution.

In general for a system of ODEs,

$$\mathbf{y}'(s) = \mathbf{f}(s, \mathbf{y}(s)), \quad \mathbf{y}(s_0) = \mathbf{y}_0,$$

we have the following theorems which give conditions on the existence and uniqueness of the solution. We at this point note that existence and uniqueness are defined in the neighborhood of a point under consideration.

4.3.1 Existence Theorem

If $\mathbf{f}(s, \mathbf{y}(s))$ is defined and continuous at $(s_0, \mathbf{y}(s_0))$, then there exists solution for the system of ODEs for a neighborhood close to $(s_0, \mathbf{y}(s_0))$.

4.3.2 Uniqueness Theorem

We define the Jacobian as $\mathbf{J} = \{J_{ij}\} = \left\{ \frac{\partial f_i}{\partial y_j} \right\}$. For a unique solution at $(s_0, \mathbf{y}(s_0))$ the Jacobian \mathbf{J} should exist and should be continuous at $(s_0, \mathbf{y}(s_0))$.

4.4 Conventional Solution Methods and Issues

The ODE solvers discussed in the previous section and other conventional solvers suffer from some very serious deficiencies. These are discussed in this section.

4.4.1 Inherent Errors

Truncation Errors

Truncation errors are caused by truncating infinite sequences of arithmetic operations after a finite number of steps. A typical example is truncating an infinite Taylor series after finite number of terms.

Rounding Errors

This pathology is caused due to computation in a floating point environment. Computation using floating point arithmetic is performed based on fixed grids. A given number which may be rational or irrational is rounded to the closest number on this grid thus introducing a small but nonzero error. This approximation over a number of operations can lead to a significant error.

Errors in Initial Data

Obtaining the starting points in general for a surface intersection problem involves solving a system of nonlinear equations numerically. This essentially results in a non zero error due to the previous inconsistencies of truncation and rounding. A small error in the initial condition for nonlinear differential equations can lead to a chaotic behavior.

Moreover in many cases a solid model is obtained after reverse engineering using some scanning devices. The output from these devices is a point cloud which may not have a unique representation corresponding to any surface. In such a case we may have an interval b-spline surfaces approximation of the point cloud [45]. Thus there is an inherent error in representing the surfaces themselves. An approximation of the surface hence could introduce errors in the intersection of the surfaces.

4.4.2 Straying or Looping

The conventional solution methods discussed in Section 4.2 are usually robust and reliable for most applications, but it is easy to find examples for which they return inaccurate results [27], especially in the presence of closely spaced features as shown in Figure 4-1. This is because the algorithms to control the step size are based on controlling just the error alone. A step size control which also verifies the existence and uniqueness before predicting the step size can prevent the solution from straying from one branch to another within that step. Looping is a result of straying from one branch to the other and back, thus going into an infinite loop.

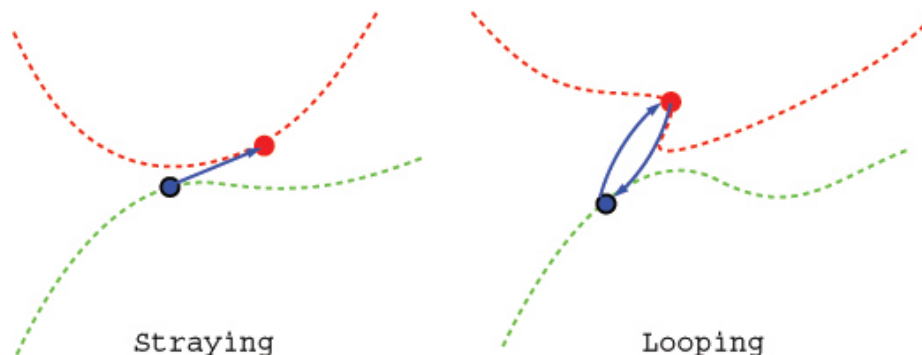


Figure 4-1: Phenomenon of straying or looping.

4.5 Interval Nonlinear ODE Solvers

Robust tracing of surface to surface intersection is one of the important problems that is addressed by Hu *et al.* [11]. The differential equations (3.3) represent a system of autonomous initial-value problem (IVP), written in vector form:

$$\mathbf{y}'(s) = \mathbf{f}([\mathbf{y}(s)]), \quad \mathbf{y}(s_0) = \mathbf{y}_0,$$

where,

$$\mathbf{f}([\mathbf{y}(s)]) = \begin{bmatrix} \sigma' \\ t' \\ u' \\ v' \end{bmatrix}, \quad \mathbf{y}_0 = \begin{bmatrix} [\sigma_0] \\ [t_0] \\ [u_0] \\ [v_0] \end{bmatrix}.$$

It is not easy to evolve an interval version of the existing algorithms for example, the Runge-Kutta method for solving a system of ODEs. Suppose we convert this approximate method to an interval method and we are concerned about the accuracy of the solutions. The resulting interval answers are useless for the following reasons [46].

1. They enclose the Runge-Kutta approximation to the solution, not the solution itself.
2. The results are very wide to be of any practical use and grow exponentially.

In order to enclose the solution, we must extend the algorithm to include an inclusion of the truncation error term. Thus naive interval versions of point algorithms do not guarantee the inclusion of the solution.

Developing a good interval algorithm [19] often involves computing a point approximation, followed by computing an interval inclusion near the approximation.

4.5.1 Advantages of an Interval ODE Solver

1. Interval Representation of Surfaces

Surfaces can be represented as interval surfaces which are specifically useful for the cases of robust reverse engineering, where the uncertainty in the surfaces is represented by means of interval surfaces. The same is true if we want to accommodate for the perturbations in the surfaces themselves. Exact surfaces can be represented as degenerate interval surfaces. Interval ODE solvers can accommodate these perturbations and return bounds which are truly conservative.

2. Accommodation of Errors due to Rounding

Use of rounded interval arithmetic ensures that the bounds obtained are conservative. The true result is bounded within the returned interval.

3. Inclusion of the Errors in Initial Condition

The initial condition which corresponds to a starting point is usually obtained as a solution of a system of nonlinear polynomial equations. There are a lot of accuracy issues when we try to solve these equations. Application of any approximate scheme for solving this system will result in an initial condition quite close to the actual solution but may not be represented exactly. Application of an algorithm for solving a system of nonlinear polynomials based on interval arithmetic like *Interval Projected Polyhedron* or *Interval Newton* guarantees that all the starting points are correctly obtained within strict error bounds. Thus the uncertainty in the starting point can be expressed as an interval and hence provided as the interval initial conditions to the interval ODE solver.

Chapter 5

Validated ODE Solver in Tracing Surface-Surface Intersections

5.1 Overview of the Method

Standard numerical methods for solving IVPs for ODEs as described in Chapter 4 attempt to compute an approximate solution that satisfies a user-specified tolerance at discrete points. In this chapter we briefly explain and apply a validated interval solution scheme for correctly tracing the surface-surface intersection in the parametric space of the surfaces. A more detailed treatment is given by Nedialkov [24]. The methods discussed in Section 4.2 are usually robust and reliable for most applications, but it is easy to find examples for which they return inaccurate results, especially when two solutions are close to each other within the tolerance for error, causing straying or looping [27]. This is because the algorithms to control the step size are based on controlling just the error alone [23]. A step size control which also verifies the existence and uniqueness before predicting the step size can prevent the solution from straying from one branch to another within that step [23, 30, 31]. Also refer to Sections 5.5 and 8.4 for a detailed review on straying and looping.

A validated interval scheme for ODEs not only produces a guaranteed error bound on the true solution, but also verifies uniqueness of the solution for the ODE system within that bound. Each step in a validated interval solution scheme for solving IVPs for ODEs can guarantee:

1. The existence of the solution: i.e. if solution exists in that step within the enclosure.
2. The uniqueness of the solution: i.e. if we have a unique solution in that step within the enclosure.

The uniqueness of the solution for a given parametrization of the surface, can eliminate looping or straying, which is an inherent problem (refer Section 8.4) in most other solvers [23].

The validated solution scheme for solving IVPs for ODEs can be traced back to Moore [22], Krückeberg [18], Eijgenraam [5] and Löhner [20]. One efficient way is

to find a bound for the *Taylor's formula* for the successive step, even enclosing the truncation error term. Please refer to Appendix A.1 for a comprehensive list of all the notations used in this chapter.

Let us assume that we have a vector interval ODE system of the form,

$$\frac{d\mathbf{y}}{ds} = \mathbf{f}([\mathbf{y}(s)]), \quad [\mathbf{y}(s_0)] = [\mathbf{y}_0]. \quad (5.1)$$

Our goal at this point is to compute boxes in the parameter space, which enclose the pre-image of a given intersection curve segment at every point. For every step, we have an initial interval $[\mathbf{y}_j]$, obtained from the previous step. This is illustrated in Figure 5-1-1. We aim at computing the enclosure $[\tilde{\mathbf{y}}_j]$ of the family of solutions $\mathbf{y}(s; s_0, [\mathbf{y}_0])$ passing through $[\mathbf{y}_0]$ for each step h_j such that,

$$\mathbf{y}(s; s_j, [\mathbf{y}_j]) \equiv \begin{bmatrix} \sigma & t & u & v \end{bmatrix}^T \subseteq \begin{bmatrix} [\tilde{\sigma}] & [\tilde{t}] & [\tilde{u}] & [\tilde{v}] \end{bmatrix}^T \equiv [\tilde{\mathbf{y}}_j], \quad \forall s \in [s_j, s_{j+1}],$$

where $\mathbf{y}(s; s_j, [\mathbf{y}_j])$ represents the family of curves passing through $[\mathbf{y}_j]$ satisfying equation (3.3). We call such a bound $[\tilde{\mathbf{y}}_j]$, a *a priori enclosure*, and try to obtain this bound on the parameters σ, t, u, v for the j^{th} step $h_j = (s_{j+1} - s_j)$.

The validated scheme for solving ODEs is usually done in two phases [22, 26]. In the *phase I algorithm*, we find out an enclosure $[\mathbf{y}_j]$ and a corresponding step size h_j for unique solution. In the second phase, called *phase II algorithm*, we obtain the initial interval for the subsequent step $[\mathbf{y}_{j+1}]$, thus proceeding the integration without considerable increase in the width of the bounds. The integration can be terminated when we have an inclusion of the endpoints.

5.2 Phase I Algorithm

This phase in a validated solving scheme for ODEs involves:

- Choosing an *a priori* bound and a step size based on *validation criterion*.
- Checking the existence and uniqueness of the solution of IVP within the *a priori enclosure* for the above step size.

Thus the goal is to compute enclosures $[\tilde{\mathbf{y}}_j]$ on the family of the solutions $\mathbf{y}(s; s_0, [\mathbf{y}_0])$ for the IVP corresponding to the intersection under consideration,

$$\mathbf{y}(s; s_j, [\mathbf{y}_j]) \subseteq [\tilde{\mathbf{y}}_j], \quad \forall s \in [s_j, s_{j+1}],$$

where $\mathbf{y}(s; s_j, [\mathbf{y}_j])$ represents the family of curves passing through $[\mathbf{y}_j]$ satisfying equation (3.3) and s is the independent variable which in our case is the arc length parameter. We call such a bound $[\tilde{\mathbf{y}}_j]$, an *a priori enclosure*, and try to obtain this bound for the j^{th} step $h_j = (s_{j+1} - s_j)$. For validating the solution for a pair of step size and an *a priori enclosure*, we can use various methods like the *constant enclosure method* [5], the *polynomial enclosure method* [21] or the *Taylor series method* [4]. The

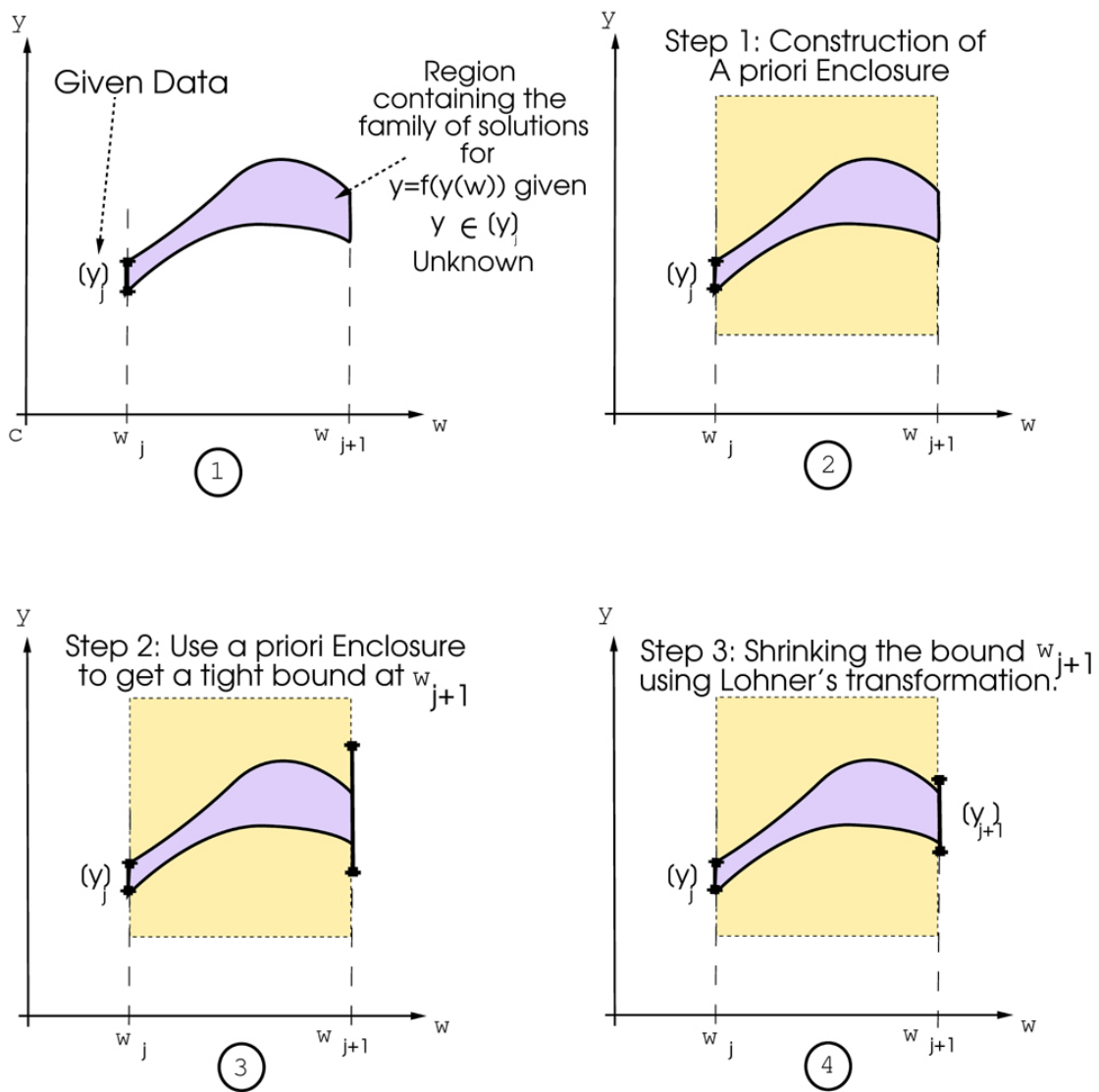


Figure 5-1: Steps involved in validated scheme for solving ODEs depicted for the case involving a single dependent variable.

Taylor series method is preferred to a constant step size method since it can allow for longer step sizes [24] and can be written as follows:

$$[\tilde{\mathbf{y}}_j(s)] \supseteq [\mathbf{y}_j] + \sum_{i=1}^{k-1} [\mathbf{y}_j]_i (s - s_j)^i + [\tilde{\mathbf{y}}_j]_k (s - s_j)^k, \quad (5.2)$$

where k is the order of the Taylor series used and $[\mathbf{y}_j]_i$ is the i^{th} Taylor coefficient evaluated at $[\mathbf{y}_j]$. We numerically solve for the corrected step size h_j , given an initial guess for an *a priori enclosure* as shown by Nedialkov [24]. At this point we have made an assumption that $\mathbf{f}([\mathbf{y}(s)])$ is well behaved and is C^k continuous.

5.3 Phase II Algorithm

Phase II of a validated solution scheme for ODEs involves:

- Propagation of the solution.
- Reducing the phenomenon of wrapping.

Using the *a priori enclosure* $[\tilde{\mathbf{y}}_j]$ from *phase I algorithm*, *phase II algorithm* computes a tighter enclosure $[\mathbf{y}_{j+1}]$ at s_{j+1} ,

$$\begin{aligned} \mathbf{y}(s_{j+1}; s_j, [\mathbf{y}_j]) &\subseteq [\mathbf{y}_{j+1}], \\ \text{such that, } [\mathbf{y}_{j+1}] &\subseteq [\tilde{\mathbf{y}}_j] \text{ at } s_{j+1}. \end{aligned}$$

This phase is important, as it will help in proceeding the integration scheme. i.e., we get the initial interval $[\mathbf{y}_{j+1}]$ for the successive step. The key difficulty we face in phase II algorithm is the *wrapping effect*. Löhner [19] defines wrapping as *undesirable overestimation of a solution set of an iteration or recurrence which occurs if this solution set is replaced by a superset of some simpler structure and this superset is then used to compute the enclosures for the next step which may eventually lead to an exponential growth of overestimation*. By containing wrapping we prevent the exponential growth in the width of the interval solution at s_{j+1} .

5.3.1 Interval Taylor Series Method

Moore's method [22] computes $[\mathbf{y}_{j+1}]$ using an interval version of the Taylor's formula,

$$\begin{aligned} [\mathbf{y}_{j+1}] &= [\mathbf{y}_j] + \sum_{i=1}^{k-1} [\mathbf{y}_j]_i h_j^i + [\tilde{\mathbf{y}}_j]_k h_j^k, \\ &= [\mathbf{y}_j] + \sum_{i=1}^{k-1} \mathbf{f}^{[i]}([\mathbf{y}_j]) h_j^i + \mathbf{f}^{[k]}([\tilde{\mathbf{y}}_j]) h_j^k, \end{aligned} \quad (5.3)$$

where, $[\mathbf{y}_j]_i \equiv \mathbf{f}^{[i]}([\mathbf{y}_j])$ is also another notation for the i^{th} Taylor coefficient. The disadvantage of using equation (5.3) directly is that the widths of $[\mathbf{y}_j]$, always increase

with j , even if the width of the true interval solution contracts as shown in Figure 5-1-3. Much smaller bounds can be achieved if we use the mean-value theorem to $\mathbf{f}^{[i]}$ [26]. i.e.,

$$\mathbf{f}^{[i]}([\mathbf{y}_j]) = \mathbf{f}^{[i]}(\hat{\mathbf{y}}_j) + \mathbf{J}(\mathbf{f}^{[i]}; [\mathbf{y}_j], \hat{\mathbf{y}}_j)([\mathbf{y}_j] - \hat{\mathbf{y}}_j),$$

and rewriting equation (5.3) to obtain,

$$[\mathbf{y}_{j+1}] = \hat{\mathbf{y}}_j + \sum_{i=1}^{k-1} \mathbf{f}^{[i]}(\hat{\mathbf{y}}_j) + \mathbf{f}^{[k]}(\mathbf{y}; s_j, s_{j+1})h_j^k + \{\mathbf{I} + \sum_{i=1}^{k-1} \mathbf{J}(\mathbf{f}^{[i]}; [\mathbf{y}_j], \hat{\mathbf{y}}_j)h_j^i\}([\mathbf{y}_j] - \hat{\mathbf{y}}_j). \quad (5.4)$$

We normally choose $\hat{\mathbf{y}}_j = m([\mathbf{y}_j])$, the mid point of $[\mathbf{y}_j]$ and \mathbf{I} is the identity matrix.

The robust generation of Taylor series coefficients and their Jacobian's are done with a technique called *automatic differentiation* [22, 43], a detailed treatment of which can be found in Chapter 6.

Controlling Wrapping

To limit *wrapping* researchers have proposed many methods starting with a local coordinate transformation by Moore [22] enclosing the solution at each step as a linear transformation of the interval vector, and constraining the error to convex polytopes by Stewart [44]. The most promising one is a *QR factorization method* developed by Löhner [20], which is also a solution to a linear transformation of an interval vector.

Let, $\mathbf{A}_0 = \mathbf{I}$, $\hat{\mathbf{y}}_0 = m([\mathbf{y}_0])$, $[\mathbf{r}_0] = [\mathbf{y}_0] - \hat{\mathbf{y}}_0$ and let us write, $[\mathbf{z}_{j+1}] = h_j^k \mathbf{f}^{[k]}([\tilde{\mathbf{y}}_j])$. Also let:

$$\hat{\mathbf{y}}_{j+1} = \hat{\mathbf{y}}_j + \sum_{i=1}^{k-1} h_j^i \mathbf{f}^{[i]}(\hat{\mathbf{y}}_j) + m([\mathbf{z}_{j+1}]),$$

and,

$$[\mathbf{S}_j] = \mathbf{I} + \sum_{i=1}^{k-1} h_j^i \mathbf{J}(\mathbf{f}^{[i]}; [\mathbf{y}_j]),$$

where, $j \geq 0$, and \mathbf{I} is the identity matrix.

Then equation (5.4) can be rewritten as,

$$[\mathbf{y}_{j+1}] = \hat{\mathbf{y}}_j + \sum_{i=1}^{k-1} h_j^i \mathbf{f}^{[i]}(\hat{\mathbf{y}}_j) + [\mathbf{z}_{j+1}] + ([\mathbf{S}_j] \mathbf{A}_j)[\mathbf{r}_j].$$

The initial condition for the next step is chosen to be the unwrapped region, and a measure of its width written as,

$$[\mathbf{r}_{j+1}] = \mathbf{A}_{j+1}^{-1}([\mathbf{S}_j] \mathbf{A}_j)[\mathbf{r}_j] + \mathbf{A}_{j+1}^{-1}([\mathbf{z}_{j+1}] - m([\mathbf{z}_{j+1}])), \quad (5.5)$$

where $\mathbf{A}_{j+1} \in \mathbf{R}^{n \times n}$ is a point matrix which is nonsingular for $j = 0, 1 \dots$ and yet to

be determined.

The reduction in wrapping depends on the choice of \mathbf{A}_{j+1} . Löhner chooses \mathbf{A}_{j+1} as shown below. Let $\hat{\mathbf{A}}_{j+1} = m([\mathbf{S}_j]\mathbf{A}_j)$, and we factorize $\hat{\mathbf{A}}_{j+1} = \mathbf{Q}_{j+1}\mathbf{R}_{j+1}$, where \mathbf{Q}_{j+1} is orthogonal and \mathbf{R}_{j+1} is upper triangular. We now assign, $\mathbf{A}_{j+1} = \mathbf{Q}_{j+1}$, and substitute in equation (5.5).

One simple way of explaining the success of the QR factorization method in reducing wrapping is that we enclose the solution on each step, in a moving coordinate system that *matches* the solution set [14]. The output obtained from the validated ODE solver is a set of boxes in the parameter space which are continuous, and discrete intervals for parameters σ, t, u, v at specific s_j 's.

5.3.2 Interval Hermite Obreschkoff Method

An alternative to the well established Taylor series method used in phase II algorithm is a relatively new method proposed by Nedialkov [14, 24], which is based on the *Hermite Obreschkoff* scheme.

The following formula is the basis for the Interval Hermite Obreschkoff method.

$$\sum_{i=0}^q (-1)^i c_i^{q,p} h_j^i \mathbf{f}^{[i]}([\mathbf{y}_{j+1}]) = \sum_{i=0}^p c_i^{p,q} h_j^i \mathbf{f}^{[i]}([\mathbf{y}_j]) + \underbrace{(-1)^q \frac{q!p!}{(p+q)!} h_j^{p+q+1} \frac{\mathbf{f}^{[p+q+1]}([\tilde{\mathbf{y}}_j])}{(p+q+1)!}}_{\text{Error term}}, \quad (5.6)$$

where $c_i^{q,p} = c_{i-1}^{q,p} \frac{q-i+1}{q+p-i+1}$ and $c_0^{q,p} = 1$. For an IVP for ODE we know $[\mathbf{y}_j]$ at each step, and we need $[\mathbf{y}_{j+1}]$. From the equation (5.6) we can see that the LHS is a polynomial in $[\mathbf{y}_{j+1}]$, and we can evaluate RHS, a polynomial in $[\mathbf{y}_j]$. Obtaining $[\mathbf{y}_{j+1}]$ involves solving the nonlinear system of equations, once the error term is known [24].

Phase II algorithm for obtaining strict bounds $[\mathbf{y}_{j+1}]$, with IHO uses a predictor-corrector method. In the predictor phase we compute an enclosure of the solution at s_{j+1} , and in the corrector phase we improve the enclosure by enclosing the solution of (5.6) using a Newton like step.

1. Computing the coefficients $c_i^{q,p}$.
2. Predicting an enclosure $[\mathbf{y}_{j+1}^{(0)}]$ at s_{j+1} . This is essentially a part of the Löhner method [24].
3. Correcting the enclosure $[\mathbf{y}_{j+1}^{(0)}]$ to compute an improved enclosure $[\mathbf{y}_{j+1}]$.

A much more detailed description and derivation of the IHO method is provided by Nedialkov [24]. Nedialkov explains why an IHO method might out-perform an ITS method. It is shown in [24] that the IHO method is more stable and produces smaller enclosures than an ITS method with the same step-size and order. We might use either ITS or IHO methods for *phase II* in the validated ODE solving scheme.

5.4 Formulation Based on Validated ODE Solver

We solve the ODEs given by the equation (3.3) using a validated ODE solver given initial conditions. Our use of rational polynomial parametric surfaces which are C^∞ continuous makes sure that $\mathbf{f}([\mathbf{y}(s)])$ is well behaved and is at least C^k continuous. *Phase I algorithm* verifies the existence and uniqueness of the intersection curve segment, and a successful validation results in a step size h_j and a corresponding *a priori enclosure* $[\tilde{\mathbf{y}}_j]$, which in the context of surface intersection is,

$$[\tilde{\mathbf{y}}_j] \equiv \left[\begin{array}{cccc} [\tilde{\sigma}] & [\tilde{t}] & [\tilde{u}] & [\tilde{v}] \end{array} \right]^T.$$

Phase II algorithm now finds a tight estimate of the bound on the parameter for a specific s_{j+1} ,

$$[\mathbf{y}_{j+1}] \equiv \left[\begin{array}{cccc} [\sigma_{j+1}] & [t_{j+1}] & [u_{j+1}] & [v_{j+1}] \end{array} \right]^T.$$

This tighter bound acts as the initial condition for the next step, and hence helps in marching along the intersection curve, without significant increase of the error in the evaluation of the intersection curve segment. The intersection curve is obtained as a series of connected *a priori enclosures* (boxes) in the parameter space, which enclose the exact curve of intersection in the parameter space as shown in Figure 5-2 [23].

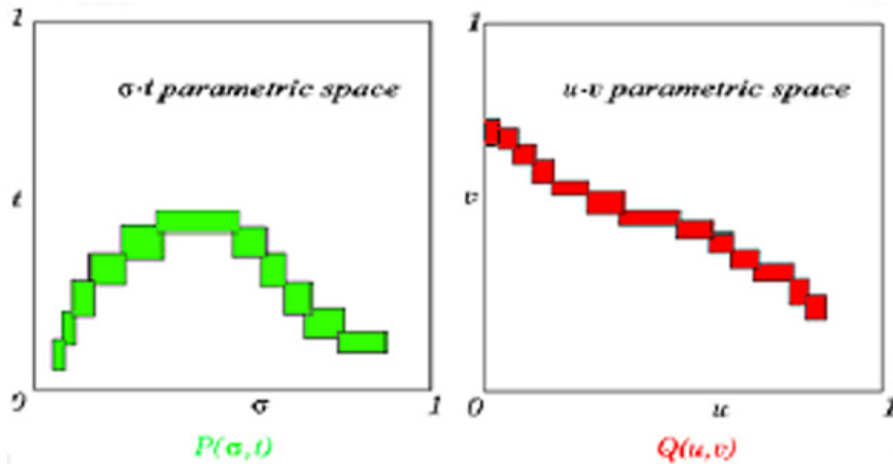


Figure 5-2: The series of *a priori enclosures* in parametric spaces which enclose the true intersection curve segment.

5.5 Resolving Singularities and Preventing Straying or Looping

Any numerical scheme, yielding a solution for a physical system represented by an IVP should first check for the *existence* and the *uniqueness* of the solution before returning an approximation, or a bound for it [16]. This however, is not a common practice in the conventional solution schemes for IVPs. A typical solution procedure is to use an approximate, point based algorithm [32] like Runge-Kutta method, Taylor series method or Adams-Bashforth technique for solving ODEs as mentioned in [29, 28]. These methods are usually robust and reliable for most applications, but it is easy to find examples for which they return inaccurate results [30, 31], especially in the neighborhood of a singularity or in the presence of closely spaced features as shown in Figure 4-1 when they are employed for surface to surface intersection problem. Also refer to Section 8.4 where we have performed examples clearly depicting straying and looping. Such abnormalities happen because the algorithms to control the step size are based on controlling just the error alone. On the other hand, a validated ODE solver verifies the *existence* and *uniqueness* of the solution for the ODE system within the *a priori* bound before determining the step size. We employ this idea to successfully resolve the cases involving singularities in phase space where the criterion of existence and uniqueness is not satisfied, and the cases of near singularity, where solutions from different branches exist quite close together, thereby tracing the correct solution.

If the solution exists and is unique for a given step size h_j and an *a priori enclosure* $[\tilde{\mathbf{y}}_j]$, the criterion (5.2) based on Taylor series holds [24]. Without loss of generality, we consider the case of $k = 1$, namely, the *constant enclosure method* [24]:

$$[\tilde{\mathbf{y}}_j(s)] \supseteq [\mathbf{y}_j] + \mathbf{f}([\tilde{\mathbf{y}}_j])h_j. \quad (5.7)$$

Let us assume that the surfaces $[\mathbf{P}](\sigma, t)$ and $[\mathbf{Q}](u, v)$ intersect transversally in such a way that they have two distinct branches and that these branches lie close to each other in a given region in the parameter space. For such regions the denominator of equation (3.4) $||[\mathbf{N}^{\mathbf{P}}] \times [\mathbf{N}^{\mathbf{Q}}]||$ gives a value close to 0 and in the event of both curve segments intersecting each other, it contains 0. The evaluation of $\mathbf{f}([\tilde{\mathbf{y}}_j])$ based on the equation (3.3) blows up, returning a smaller and smaller step size and correspondingly smaller $[\tilde{\mathbf{y}}_j]$ to satisfy the criterion (5.7). In the event of $0 \in ||[\mathbf{N}^{\mathbf{P}}] \times [\mathbf{N}^{\mathbf{Q}}]||$, the criterion is never satisfied, and this condition is reported as a singularity [23].

This validated step size strategy can hence not only prevent straying or looping but also successfully resolve the singularities of intersection curve segments. Examples that fully demonstrates this capability of the validated ODE solver is performed in Section 8.4.

5.6 Complexity Analysis

As documented by Nedialkov [24] the most expensive part of the interval Taylor series method is in generating the high-order Jacobians and matrix-matrix multiplications. Further there might be significant memory overheads especially when implemented using operator overloading involving many memory allocations and reallocations, and also in reading and storing Taylor coefficients and their Jacobians.

If n is the number of equations, we can assume that the cost of evaluating $\frac{\partial f^{[i]}}{\partial y}$ is roughly n times the cost of evaluating $f^{[i]}$ [25]. Thus for generating $k-1$ Jacobians in an interval Taylor series method one require $c_f n N k^2 + O(n N k)$ arithmetic operations, where c_f is the ratio of multiplications and divisions in N arithmetic operations for the evaluation of f [25]. In a similar way if we let $p = q$ and $k = p + q + 1$. We need to generate $p = (k-1)/2$ terms for the forward solution and $q = p = (k-1)/2$ terms for the backward one. The corresponding work is then, $\frac{c_f n N k^2}{2} + O(n N k)$.

Löhner's method with QR factorization technique requires computing an enclosure of the inverse of a point matrix, which in fact is a floating-point approximation of an orthogonal matrix [25]. Based on the assumption that $N \approx n^2$ [25], the work done per step for ITS method is given by,

$$c_f n^3 k^2 + O(n^3 k).$$

Similarly the work done per step for IHO is given by,

$$\frac{c_f n^3 k^2}{2} + O(n^3 k).$$

Chapter 6

Automatic Differentiation

6.1 Introduction

A method of divided difference has been widely used as a tool to replace symbolic manipulation for numerical derivative computation. However, using divided difference we introduce truncation errors which affect further computations and hence the scheme is no more robust.

Automatic differentiation (AD) is an alternative to the above methods. AD uses the chain rule to compute the derivatives of composite functions. AD evaluates a function and its derivatives using the same code and common temporary values. If the code is optimized the derivatives are optimized as well. The resulting differentiation is accurate up to round-off errors. If we calculate using interval arithmetic, we obtain enclosures of the true derivatives. It is quite easy to implement in computer languages allowing operator overloading. Stauning [24] gives an introduction to this method.

Let us assume that we have a rational function (one in only rational operations like $+$, $-$, $*$, $/$, \sin , \exp , etc.). We can decompose the expression for a general function $f(x)$, $x = (x_1, \dots, x_m)$, into a list of equations representing the function,

$$\begin{aligned}\tau_i(x) &= g_i(x) = x_i & i &= 1 \dots m, \\ \tau_i(x) &= g_i(\tau_1(x) \dots \tau_{i-1}(x)) & i &= (m+1) \dots l.\end{aligned}$$

All functions $\tau_i(x)$ and $g_i(x)$ are scalar functions and only one elementary operation occurs in each of the functions $g_i(x)$. We call such a list of functions a *code list*. Elementary functions have an *arity* of 0, 1 and 2, corresponding to a constant (eg: u), unary (eg: $\sin(u)$) or a binary (eg: $u+v$) operation.

Assume that $f : \mathbf{R}^m \rightarrow \mathbf{R}^n$ is a rational function given by an expression which is decomposable into a *code-list* given by the functions g_i , and assume that the functions are differentiable and we can obtain their derivatives, then by chain rule for composite functions we can obtain,

$$\frac{\partial \tau_i}{\partial \tau_j} = \delta_{ij} + \sum_{k=j}^{i-1} \frac{\partial g_i}{\partial \tau_k} \frac{\partial \tau_k}{\partial \tau_j},$$

$$\delta_{ij} = \begin{bmatrix} 1 & i = j \\ 0 & i \neq j \end{bmatrix}.$$

Let us now define the matrices,

$$Dg = \begin{bmatrix} 0 & 0 & 0 & \cdots \\ \left(\frac{\partial g_2}{\partial \tau_1}\right) & 0 & 0 & 0 \\ \left(\frac{\partial g_3}{\partial \tau_1}\right) & \left(\frac{\partial g_3}{\partial \tau_2}\right) & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

and the matrix $D\tau$ is defined as,

$$D\tau = \begin{bmatrix} 0 & 0 & 0 & \cdots \\ \left(\frac{\partial \tau_2}{\partial \tau_1}\right) & 0 & 0 & 0 \\ \left(\frac{\partial \tau_3}{\partial \tau_1}\right) & \left(\frac{\partial \tau_3}{\partial \tau_2}\right) & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

We formulate this as a matrix equation,

$$\begin{aligned} D\tau &= I + DgD\tau, \\ (I - Dg)D\tau &= I. \end{aligned}$$

We know that $(I - Dg)$ is not singular, hence,

$$\begin{aligned} (I - Dg)D\tau &= D\tau(I - Dg) = I, \\ D\tau &= (I - Dg)^{-1}, \end{aligned} \tag{6.1}$$

or

$$\begin{aligned} (I - Dg)^T D\tau^T &= I, \\ D\tau^T &= ((I - Dg)^T)^{-1}. \end{aligned} \tag{6.2}$$

The matrix Dg is usually very sparse and this is utilized in solving the systems.

6.2 Forward Mode Automatic Differentiation

From equation (6.1) we solve for the i^{th} column to get all the derivatives with respect to τ_i . To find all the derivatives with respect to all arguments of f , we solve for the first m columns of $D\tau$. This method is called the *Forward mode automatic differentiation (FAD)* [24].

6.3 Backward Mode Automatic Differentiation

The equation (6.2) can be solved using backward substitution. If we solve the equation with respect to the i^{th} column in $D\tau^T$, we obtain derivatives of τ_i with respect to all

| | |
|-------------------------|--------------------------------------|
| $g_i(u, v)$ | $\frac{\partial g_i}{\partial u}(u)$ |
| $+u$ | 1 |
| $-u$ | -1 |
| $exp(u)$ | $exp(u)$ |
| $log(u)$ | $\frac{1}{u}$ |
| $\sqrt{}(u)$ | $\frac{1}{2g_i(u)}$ |
| $sin(u)$ | $cos(u)$ |
| $cos(u)$ | $-sin(u)$ |
| $tan(u)$ | $1 + g_i^2(u)$ |
| $asin(u)$ | $\frac{1}{\sqrt{1+g_i^2(u)}}$ |
| $acos(u)$ | $\frac{-1}{\sqrt{1-g_i^2(u)}}$ |
| $atan(u)$ | $\frac{1}{1+g_i^2(u)}$ |

Table 6.1: Common Unary Operations

| $g_i(u, v)$ | $\frac{\partial g_i}{\partial u}(u, v)$ | $\frac{\partial g_i}{\partial v}(u, v)$ |
|---------------|---|---|
| $u + v$ | 1 | 1 |
| $u - v$ | 1 | -1 |
| $u \cdot v$ | v | u |
| $\frac{u}{v}$ | $\frac{1}{v}$ | $\frac{-g_i(u, v)}{v}$ |
| u^v | vu^{v-1} | $g_i(u, v) \ln(u)$ |

Table 6.2: Common Binary Operations

arguments of $f(\tau_j$ for $1 \leq j \leq m$). This method is called *Backward mode automatic differentiation (BAD)*. The reason why the BAD is famous is that we are capable of computing all partial derivatives of a scalar function $f : C^1(\mathbf{R}^m, \mathbf{R}^n)$, in the equation (6.2 just by solving with respect to one column of $D\tau^T$).

Thus the rule of thumb is,

- Use FAD for obtaining the Jacobian if $m \leq n$.
- Use BAD for obtaining the Jacobian if $m > n$.

In the tables 6.1 and 6.2, we summarize the most commonly used operations and standard functions and their derivatives.

6.4 Automatic Generation of Taylor Coefficients

We will try to give a brief introduction for generating the Taylor coefficients recursively. We denote the i^{th} Taylor coefficient of a function $y(w)$ evaluated at some point

w_j by

$$(y_j)_i = \frac{1}{i!} \frac{d^i}{dw^i} y(w_j).$$

Now consider the autonomous ODE system,

$$y'(w) = f(y), \quad y(w_j) = y_j.$$

We introduce the sequence of functions,

$$\begin{aligned} f^{[0]}(y) &= y, \\ f^{[i]}(y) &= \frac{1}{i} \left(\frac{\partial f^{[i-1]}}{\partial y} f \right)(y). \end{aligned}$$

Thus we can write,

$$\begin{aligned} (y_j)_0 &= f^{[0]}(y_j) = y_j, \\ (y_j)_i &= f^{[i]}(y_j) = \frac{1}{i} \left(\frac{\partial f^{[i-1]}}{\partial y} f \right)(y_j), \\ (y_j)_i &= f^{[i]}(y_j) = \frac{1}{i} (f(y_j))_{i-1}, \quad \text{for } i \geq 1, \end{aligned}$$

where $(f(y_j))_{i-1}$ is the $(i-1)^{st}$ coefficient of f evaluated at y_j . Thus we have a method to recursively evaluate $(y_j)_i$, for $i \geq 1$.

Now for the interval version of the evaluation of the Taylor coefficients, let

$$y(w_j) = y_j \in [y_j]$$

Thus we need to have a procedure to compute the point Taylor coefficients of $y(w)$ and perform the computations in interval arithmetic with $[y_j]$ instead of y_j . We denote the i^{th} interval Taylor coefficient of $y(w)$ at w_j by,

$$[y_j]_i = f^{[i]}([y_j]).$$

A similar concept can be extended to vector functions where we do the operations component-wise. We can further use the same concept in generating Jacobians of the Taylor series coefficients.

6.5 Complexity Analysis

As documented by Nedialkov [24] the most expensive part of the interval Taylor series method is in generating the high-order Jacobians and matrix-matrix multiplications. If n is the number of equations, we can assume that the cost of evaluating $\frac{\partial f^{[i]}}{\partial y}$ is roughly n times the cost of evaluating $f^{[i]}$ [25]. Thus for generating $k-1$ Jacobians in an interval Taylor series method one requires $c_f n N k^2 + O(n N k)$ arithmetic operations,

where c_f is the ratio of multiplications and divisions in N arithmetic operations for the evaluation of f [25].

Chapter 7

Error Bounds in Model Space

The significance of the *a priori enclosure* in interval analysis has been limited as a way to enclose the truncation error term, in the Taylor's formula for obtaining each successive step, thus providing a method for obtaining a bound for the solution to the ODE system at s_{j+1} . We realize that the *a priori enclosure* $[\tilde{\mathbf{y}}_j]$ actually bounds the solution $\mathbf{y}(s; s_j, [\mathbf{y}_j])$ over the entire step h_j . This series of *a priori enclosures* obtained in the parameter space is used to obtain a series of boxes (enclosures) in model space which enclose the true intersection curve in the model space. In this chapter we will deal with obtaining these bounds, answer the question as to why the mapping process ensures a guaranteed model space error bound. We further obtain a way to reduce the error in the model space (*model space error bound*). We also explain a method to monotonically control the size of the series of boxes obtained by the validated ODE solver.

7.1 Mapping into Model Space

We obtain $[\tilde{\mathbf{y}}_j], \forall j$, i.e. a series of *a priori enclosures* in the parametric spaces of the surfaces as shown in the Figure 5-2. This bound in parameter space is continuous because the intersection curve segment is a continuous trajectory in the parametric space. For each of the parameters $\sigma - t$ and $u - v$ corresponding to each surface, the pre-image of the curve of intersection is enclosed in the union of the boxes corresponding to *a priori enclosures* [23]. Also refer to Figure 7-1.

The series of *a priori enclosures* in the $\sigma - t$ and $u - v$ parametric space of each surface is mapped to the model space. We develop the following Theorem 2 to prove that the union of the bounds in the model space guarantees to contain the true intersection curve segment.

Theorem 2 Let $[\sigma(s)]$ and $[t(s)]$ be mappings defined by,

$$[\sigma(s)], [t(s)] : \mathbf{IR} \rightarrow \mathbf{IR}^2,$$

such that they are continuous in $s \in [s_0, s_{end}]$. Suppose $[\mathbf{P}](\sigma(s), t(s))$ is a continuous

rational interval function defined by,

$$[\mathbf{P}](\sigma, t) : \mathbf{IR}^2 \rightarrow \mathbf{IR}^3.$$

Then the mapping $[\mathbf{P}](s) = [\mathbf{P}](\sigma(s), t(s)) : \mathbf{IR} \rightarrow \mathbf{IR}^3$ is continuous in \mathbf{IR}^3 for $s \in [s_0, s_{end}]$.

The proof directly follows from the continuity of rational interval functions proved by Moore [22]. Majority of mapping in CAD practice including polynomials is continuous and rational, and hence we realize the goal of a *continuous gap-free bound* on the curve of intersection in the model space, given continuous bounds on its pre-image.

Thus mapping using RPP patches ensures guaranteed error bounds. At this point we have two series of boxes in the model space each of which enclose the true curve of intersection in the model space.

7.2 Intersection in Model Space and Reduction of Model Space Error Bound

The union of the boxes obtained in model space by mapping the enclosures of the pre-image of the curve of intersection bounds the true curve of intersection. The series of boxes obtained from each of the parametric spaces $\sigma - t$ and $u - v$ contain the true intersection curve segment. We may further prove using Theorem 3 that the true intersection curve segment actually lies in the region obtained by the intersection of the two separate bounds.

Theorem 3 *Let $[\mathbf{c}_P(s)]$ and $[\mathbf{c}_Q(s)]$ be curves of intersection in the model space obtained by mapping the pre-images of the bounds to the intersection curve from each of the surface patches. Also let us assume that $\mathbf{c}_f(s)$ is the actual curve of intersection of the two surfaces. Then $\mathbf{c}_f(s)$ lies in the region in the model space obtained by the intersection of $[\mathbf{c}_P(s)]$ and $[\mathbf{c}_Q(s)]$.*

Proof. From the definition of interval arithmetic we know,

$$\begin{aligned} \mathbf{c}_f(s) &\in [\mathbf{c}_P(s)], \\ \mathbf{c}_f(s) &\in [\mathbf{c}_Q(s)]. \end{aligned}$$

Hence we say,

$$\mathbf{c}_f(s) \in [\mathbf{c}_P(s)], \text{ and } \mathbf{c}_f(s) \in [\mathbf{c}_Q(s)].$$

From elementary set theory,

$$\begin{aligned} (\mathbf{c}_f(s) \in [\mathbf{c}_P(s)]) \cap (\mathbf{c}_f(s) \in [\mathbf{c}_Q(s)]), \\ \Rightarrow \mathbf{c}_f(s) \in ([\mathbf{c}_P(s)] \cap [\mathbf{c}_Q(s)]). \end{aligned}$$

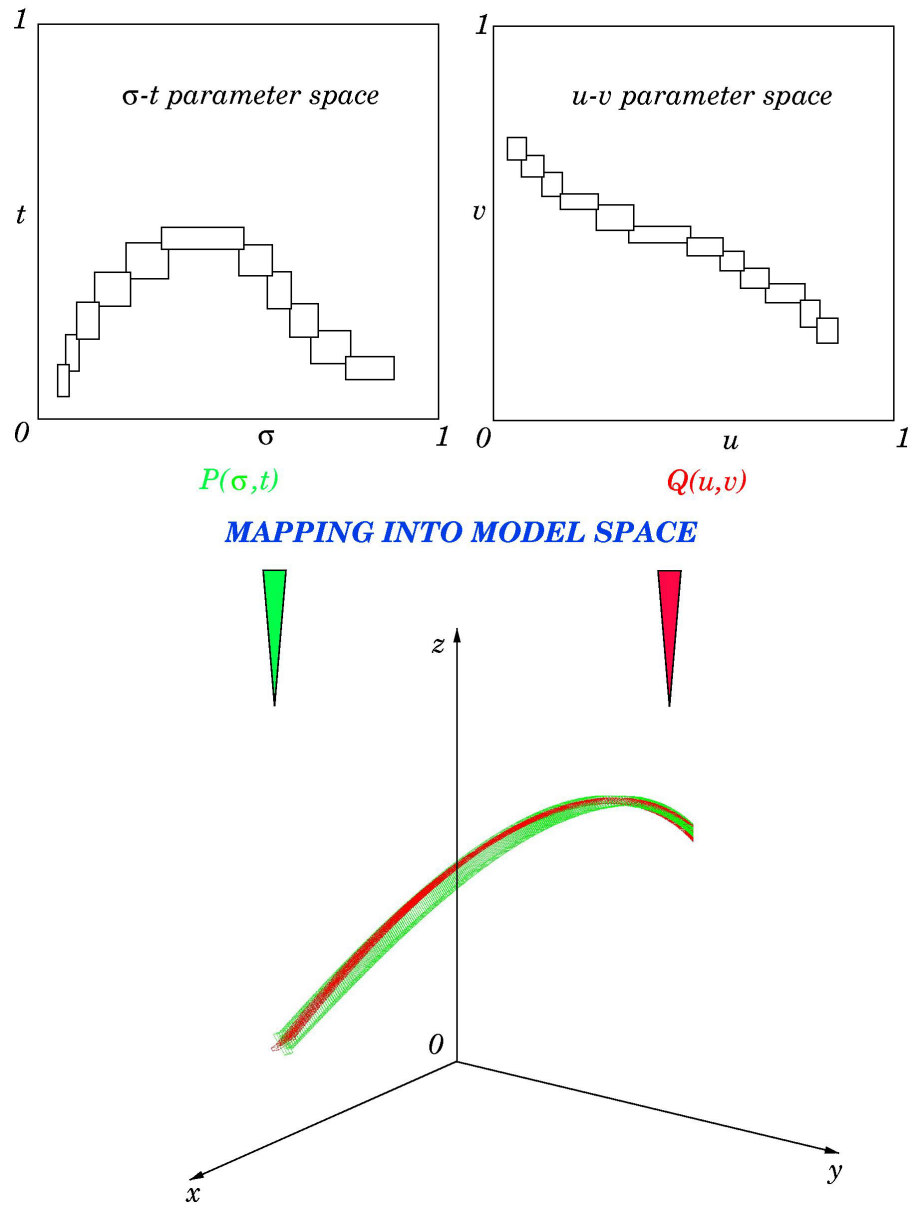


Figure 7-1: Mapping of the pre-image of the intersection curve segment from the parameter space to the model space. Note that the boxes obtained in the parameter space of each of the surface is continuous, gap free and ordered.

Performing intersection in model space may reduce the error bounds. The reduction in error bound usually depends on the relative orientation of the surfaces close to intersection as well as the relative sizes of the model space bounds obtained from each of the surfaces. Thus if the bounds from the surfaces are relatively of the same size, then the reduction is significant for a transversal intersection case compared to a tangential intersection case.

7.3 Monotonic Control of Error Bounds

The continuous error bound which we obtain is essentially the *a priori enclosures* in the parametric space of the surfaces which are further mapped to the model space. The error control based just on a validated interval arithmetic scheme is applicable only to the error at certain definite s_j 's. What we require is to develop a method such that given a tolerance in the model space we can limit the size of the *a priori enclosures* obtained by the validated ODE solvers. This essentially requires the following steps.

1. The model space tolerance is transformed into a conservative tolerance in the parametric spaces as shown by the Figure 7-2. For this we propose two methods.

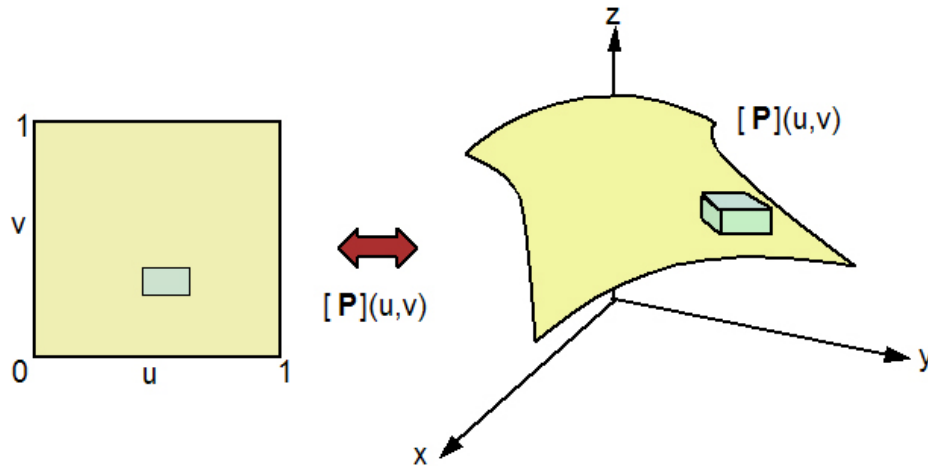


Figure 7-2: Depicts how we want to obtain tolerance in parametric space from tolerance in model space.

- (a) A strictly conservative method in the true spirit of interval arithmetic and,
 - (b) An approximate method, which is adapted from a previous work on interval solids by Shen *et al.* [38, 37].
2. A control mechanism to control the *a priori enclosure* size obtained using a validated ODE solver within the above tolerance as depicted in Figure 7.3.

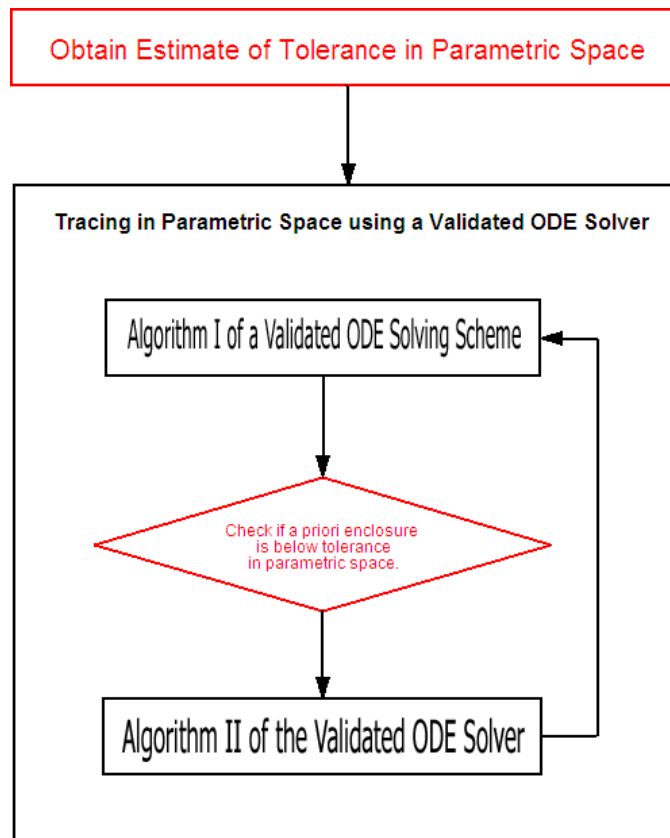


Figure 7-3: The flow chart representing control mechanism.

7.3.1 Conservative Relation

For our analysis we consider a bi-cubic RPP surface namely a bicubic Bézier surface. Similar expressions can be evaluated for higher order surfaces. Consider an interval bi-cubic Bézier surface $[P](\sigma, t)$ such that $0 \leq \sigma, t \leq 1$. Suppose we find the uncertainty of any one component (without loss of generality we can assume the x -component be $P(\sigma, t)$). We are to get a bound for $w([\sigma])$ or $w([t])$ such that, $w([P]([\sigma], [t])) \leq g(w([\sigma]), w([t]))$.

For a bi-cubic Bézier patch we can write:

$$P(\sigma, t) = \begin{bmatrix} (1-\sigma)^3 & 3\sigma(1-\sigma)^2 & 3\sigma^2(1-\sigma) & \sigma^3 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix},$$

where b_{ij} 's are the control points of the x -component. For the simplicity of analysis we assume that the control points are degenerate intervals (real numbers). We expand the above relation as,

$$P(\sigma, t) = (\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & \beta_4 \end{bmatrix} \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix},$$

$$\text{where, } \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} (1-t)^3\beta_1 \\ 3t(1-t)^2\beta_2 \\ 3t^2(1-t)\beta_3 \\ t^3\beta_4 \end{bmatrix} \text{ and,}$$

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} = \begin{bmatrix} b_{11}(1-\sigma)^3 + 3b_{21}\sigma(1-\sigma)^2 + 3b_{31}\sigma^2(1-\sigma) + b_{41}\sigma^3 \\ b_{12}(1-\sigma)^3 + 3b_{22}\sigma(1-\sigma)^2 + 3b_{32}\sigma^2(1-\sigma) + b_{42}\sigma^3 \\ b_{13}(1-\sigma)^3 + 3b_{23}\sigma(1-\sigma)^2 + 3b_{33}\sigma^2(1-\sigma) + b_{43}\sigma^3 \\ b_{14}(1-\sigma)^3 + 3b_{24}\sigma(1-\sigma)^2 + 3b_{34}\sigma^2(1-\sigma) + b_{44}\sigma^3 \end{bmatrix}.$$

Thus $P(\sigma, t) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$.

We are interested in finding the maximum width $w([P(\sigma, t)])$ within which the evaluated width would lie. From interval arithmetic we have the formula,

$$w(a + b) = w(a) + w(b),$$

where a and b are any two intervals. Applying this formula we obtain,

$$w([P(\sigma, t)]) = w(\alpha_1) + w(\alpha_2) + w(\alpha_3) + w(\alpha_4). \quad (7.1)$$

The RHS represents the thickness of the interval surface. Further using the interval

inequalities,

$$\begin{aligned} w(a.b) &\leq \|a\| w(b) + \|b\| w(a), & \text{and} \\ \|abc\| &= \|a\| \|b\| \|c\|. \end{aligned}$$

We can evaluate the widths of α_i s as:

$$\begin{aligned} w(\alpha_1) &= w((1-t)^3\beta_1) \leq w((1-t)^3) \|\beta_1\| + w(\beta_1) \|(1-t)^3\|, \\ w(\alpha_2) &= w(3t(1-t)^2\beta_2) \leq w(3t(1-t)^2) \|\beta_2\| + w(\beta_2) \|(3t(1-t)^2)\|, \\ w(\alpha_3) &= w(3t^2(1-t)\beta_3) \leq w(3t^2(1-t)) \|\beta_3\| + w(\beta_3) \|3t^2(1-t)\|, \\ w(\alpha_4) &= w(t^3\beta_4) \leq w(t^3) \|\beta_4\| + w(\beta_4) \|t^3\|. \end{aligned}$$

We can also assume that $0 \leq \sigma, t \leq 1$. After further simplification we can obtain the inequalities ¹:

$$\begin{aligned} \|(1-t)\| &\leq 1, \\ \|(1-t)^3\| &\leq 1, \\ \|t^3\| &\leq 1, \\ \|3t(1-t)^2\| &\leq \frac{4}{9}, \\ \|(1-t)^3\| &\leq \frac{4}{9}, \\ w(1-t) = w(1) + w(t) &= w(t), \\ w((1-t)^3) &\leq 12w(t), \\ w(t^3) &\leq 3w(t), \\ w(3t(1-t)^2) &\leq 24w(t), \\ w(3t^2(1-t)) &\leq 15w(t). \end{aligned}$$

Based on the assumption that the surfaces are represented by degenerate intervals i.e. $w(b_{ij}) = 0$, we can write

$$\begin{aligned} w(\beta_1) &\leq w(\sigma) (12 \|b_{11}\| + 24 \|b_{21}\| + 15 \|b_{31}\| + 3 \|b_{41}\|), \\ w(\beta_2) &\leq w(\sigma) (12 \|b_{12}\| + 24 \|b_{22}\| + 15 \|b_{32}\| + 3 \|b_{42}\|), \\ w(\beta_3) &\leq w(\sigma) (12 \|b_{13}\| + 24 \|b_{23}\| + 15 \|b_{33}\| + 3 \|b_{43}\|), \\ w(\beta_4) &\leq w(\sigma) (12 \|b_{14}\| + 24 \|b_{24}\| + 15 \|b_{34}\| + 3 \|b_{44}\|). \end{aligned} \tag{7.2}$$

¹The maximum of the functions over an interval is obtained from differential calculus.

To obtain $\|\beta_i\|$ s we use the triangular inequality,

$$\|a + b\| \leq \|a\| + \|b\|.$$

We obtain $\|\beta_i\|$ s as:

$$\begin{aligned} \|\beta_1\| &\leq \|b_{11}\| + \frac{4}{9}\|b_{21}\| + \frac{4}{9}\|b_{31}\| + \|b_{41}\|, \\ \|\beta_2\| &\leq \|b_{12}\| + \frac{4}{9}\|b_{22}\| + \frac{4}{9}\|b_{32}\| + \|b_{42}\|, \\ \|\beta_3\| &\leq \|b_{13}\| + \frac{4}{9}\|b_{23}\| + \frac{4}{9}\|b_{33}\| + \|b_{43}\|, \\ \|\beta_4\| &\leq \|b_{14}\| + \frac{4}{9}\|b_{24}\| + \frac{4}{9}\|b_{34}\| + \|b_{44}\|. \end{aligned}$$

If we assume that $\|b_{ij}\| \leq R_{max}$, i.e. the size of the boxes in which the surfaces are enclosed or the maximum dimension of the control points(convex hull), we can write,

$$\|\beta_1\| = \|\beta_2\| = \|\beta_3\| = \|\beta_4\| \leq \frac{26}{9}R_{max}.$$

Substituting into equation (7.2) we obtain,

$$\begin{aligned} w(\beta_1) &\leq w(\sigma).54R_{max} \\ w(\beta_2) &\leq w(\sigma).54R_{max} \\ w(\beta_3) &\leq w(\sigma).54R_{max} \\ w(\beta_4) &\leq w(\sigma).54R_{max} \end{aligned}$$

Applying the above relations to equation (7.1) we finally obtain the relationship,

$$w([P(\sigma, t)]) \leq 156.R_{max}.(w(\sigma) + w(t)). \quad (7.3)$$

The above equation (7.3) means that if we are given a tolerance in model space i.e. tol_{MS} , we force that:

$$w([P(\sigma, t)]) \leq 156.R_{max}.(w(\sigma) + w(t)) \leq tol_{MS}.$$

Thus if we assume that we require same tolerance ($\epsilon_{PS_{cons}}$) in the σ and t domain we can write,

$$\epsilon_{PS_{cons}} \leq \frac{tol_{MS}}{312R_{max}}. \quad (7.4)$$

We note the following points:

1. Evaluation of width depends on the form in which the equation is evaluated.
2. This formula might overestimate the width, but it is a conservative estimate in the sense that we obtain a tolerance in the parameter space, which will result

in a conservative tolerance in the model space..

3. The tolerance in parametric space depends on the convex hull of the surface.

7.3.2 Approximate Relation

This is yet another method for estimating a parameter space bound which would result in a model space bound close to a given tolerance. It is based on a relation obtained by Shen *et al.* [38, 37]. Let us assume an interval Bézier surface of degrees d_σ and d_t , with each control point having a constant width h_0 for each of its components. Just as in the conservative relation in Section 7.3.1 we obtain only one component, say R_x is the average of the absolute value of the x -coordinates of the R_i s. Similar formula for y and z components can be obtained by replacing R_x with R_y and R_z respectively. We obtain the following estimate:

$$\epsilon_{PS_{approx}} \approx \frac{tol_{MS} - h_0}{4(d_\sigma + d_t)(h_0 + R_{max})}, \quad (7.5)$$

where,

$$\begin{aligned} R_{max} &= \text{norm}(R_x, R_y, R_z), \\ tol_{MS} &= \text{allowed tolerance for error in the model space.} \end{aligned}$$

We further notice the following properties.

- The coordinates of control points affect the obtained parametric space tolerance.
- The size of control points also affects the tolerance needed.
- This relation can account for surfaces of any order.

Thus we are in a position to obtain a tolerance in the parameter space $\epsilon_{PS_{approx}}$ which corresponds to a given tolerance in the model space.

7.3.3 Controlling A Priori Enclosure in a Validated ODE Solver

The control mechanism is implemented in a validated ODE solver. The first step is to obtain the parametric space tolerance ϵ_{PS} from the control points of the surfaces using either of equation (7.4) or (7.5). This tolerance in parametric space for each surface is compared to the width of the *a priori enclosure* obtained from Algorithm I of the validated ODE solver.

$$w([\tilde{\mathbf{y}}_j]) = \left[w([\tilde{\sigma}_j]) \quad w([\tilde{t}_j]) \quad w([\tilde{u}_j]) \quad w([\tilde{v}_j]) \right]^T \leq \left[\epsilon_{PS_{\mathbf{P}}} \quad \epsilon_{PS_{\mathbf{P}}} \quad \epsilon_{PS_{\mathbf{Q}}} \quad \epsilon_{PS_{\mathbf{Q}}} \right]^T,$$

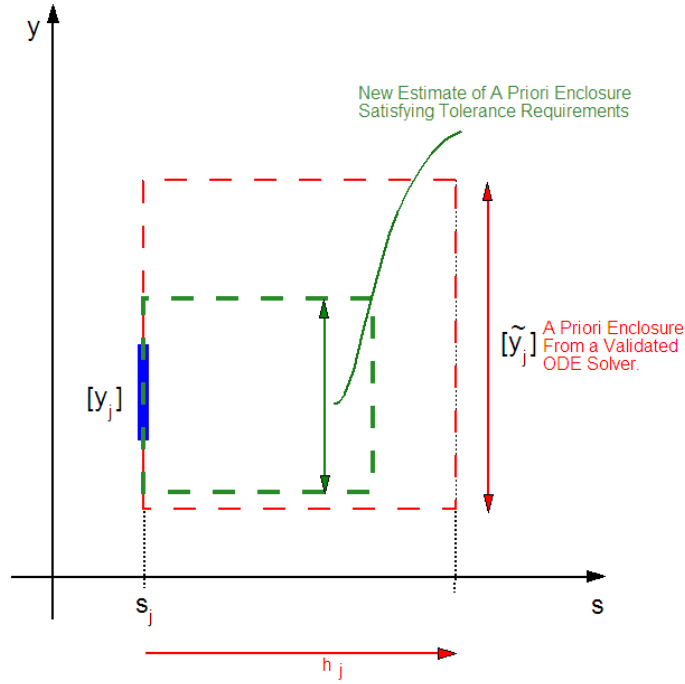


Figure 7-4: Mechanism to control the size of the *a priori enclosure* in a validated ODE solver.

where ϵ_{PS}^P and ϵ_{PS}^Q are the tolerance in the parametric space of surfaces \mathbf{P} and \mathbf{Q} , respectively.

If the width of $[\tilde{y}_j]$ is larger than the tolerance, we use this tolerance as the width of the new *a priori enclosure* and find the corresponding new step size h_j^* for which the validity criterion is satisfied. This condition is depicted in the Figure 7-4. If the width of $[\tilde{y}_j]$ is smaller than the tolerance, we have no problems at all. For further verification we might map the obtained enclosure right away to model space and check if the width of the boxes in the model space is less than the given tolerance.

7.3.4 Comparison of Two Methods

- The conservative approach complements the spirit of interval methods used throughout our work.
- There is an over estimation in the conservative method due to problems inherent with interval arithmetic.

Examples

Equations (7.4) and (7.5) are used to obtain tolerances in the parametric space. We tabulate the results in this section. We can observe that the parametric space bounds

obtained by the conservative relation are smaller than the parametric space bounds obtained by the approximate relation by almost an order of magnitude.

Example 1: $[\mathbf{Q}_3](u, v)$ of Chapter 8

| Test No. | Tolerance in Model Space | $\epsilon_{PS_{cons}}$ | $\epsilon_{PS_{approx}}$ |
|----------|--------------------------|------------------------|--------------------------|
| 1 | 10^{-2} | 1.42×10^{-6} | 1.85×10^{-5} |
| 2 | 10^{-4} | 1.42×10^{-8} | 1.85×10^{-7} |
| 3 | 10^{-6} | 1.42×10^{-10} | 1.85×10^{-9} |
| 4 | 10^{-8} | 1.42×10^{-12} | 1.85×10^{-11} |

Table 7.1: The tolerance in parametric space of surface $[\mathbf{Q}_3](u, v)$ obtained from the given tolerance in model space.

Example 2: $[\mathbf{P}_1](\sigma, t)$ of Chapter 8

| Test No. | Tolerance in Model Space | $\epsilon_{PS_{cons}}$ | $\epsilon_{PS_{approx}}$ |
|----------|--------------------------|------------------------|--------------------------|
| 1 | 10^{-2} | 2×10^{-6} | 2×10^{-5} |
| 2 | 10^{-4} | 2×10^{-8} | 2×10^{-7} |
| 3 | 10^{-6} | 2×10^{-10} | 2×10^{-9} |
| 4 | 10^{-8} | 2×10^{-12} | 2×10^{-11} |

Table 7.2: The tolerance in parametric space of surface $[\mathbf{P}_1](\sigma, t)$ obtained from the given tolerance in model space.

Example 3: $[\mathbf{P}_3](\sigma, t)$ of Chapter 8

| Test No. | Tolerance in Model Space | $\epsilon_{PS_{cons}}$ | $\epsilon_{PS_{approx}}$ |
|----------|--------------------------|------------------------|--------------------------|
| 1 | 10^{-2} | 1×10^{-6} | 1.5×10^{-5} |
| 2 | 10^{-4} | 1×10^{-8} | 1.5×10^{-7} |
| 3 | 10^{-6} | 1×10^{-10} | 1.5×10^{-9} |
| 4 | 10^{-8} | 1×10^{-12} | 1.5×10^{-11} |

Table 7.3: The tolerance in parametric space of surface $[\mathbf{P}_3](\sigma, t)$ obtained from the given tolerance in model space.

7.4 Complexity Analysis

If n_a is the number of steps we need in tracing, then the number of *a priori enclosure* boxes in the parameter space is also n_a .

Mapping The mapping involves 3 equations in each of the two surfaces, and we have an order m for the surfaces. Then the complexity is $6m$ per box. If we have n_a boxes to be mapped, the order of complexity is,

$$O(m \cdot n_a).$$

Intersection If the number of boxes that are to be mapped is n_a , for each box in a given surface we have to check if it intersects with every other box in the other surface. Thus,

$$O(n_a^2).$$

Chapter 8

Examples

An experimental implementation was developed using standard libraries, component packages [33, 6, 47] and Interval Library developed at Design Laboratory, MIT. Various examples depicting the different cases of intersections were performed and the results were tabulated. All calculations were performed on a single PC running at 1.4 GHz. with 512MB. of RAM under a Linux environment.

The quality of the results obtained (strict bound on error) is comparable only to the IPP solver developed at MIT. Any other method can obtain only an approximation and further an estimate of the error. This estimate of the error is not a bound but rather only an estimate. Please refer to [30, 31] and [23] for further explanations.

8.1 Transversal Intersection of Surfaces

We tested the validated ODE solver for a variety of intersection cases involving transversal intersection. As shown by the examples we obtain validated error bounds in model space.

8.1.1 Intersection of two Bicubic Bézier Surfaces

Figure 8-1 illustrates the intersection of two interval bicubic Bézier surfaces $[\mathbf{P}_1](\sigma, t)$ and $[\mathbf{Q}_1](u, v)$. The interval control points of the surfaces are represented by degenerate interval points and are listed below. The solution is obtained by the validated ODE solver and mapped into model space. In Table 8.1 we tabulate the variation of the *maximum relative model space error* with the number of steps and the corresponding VNODE tolerance. The maximum relative model space error is defined as the maximum width of the model space error obtained as a box in model space per unit arc length of the intersection curve segment.

$$[\mathbf{P}_1](\sigma, t) = \begin{array}{c} \text{Control points for} \\ \left[\begin{array}{cccc} ([0], [0], [5]) & ([0], [4], [5]) & ([0], [6], [5]) & ([0], [10], [5]) \\ ([4], [0], [5]) & ([4], [4], [8]) & ([4], [6], [8]) & ([4], [10], [5]) \\ ([6], [0], [5]) & ([6], [4], [8]) & ([6], [6], [8]) & ([6], [10], [5]) \\ ([10], [0], [5]) & ([10], [4], [5]) & ([10], [6], [5]) & ([10], [10], [5]) \end{array} \right] \end{array}$$

$$[\mathbf{Q}_1](u, v) = \begin{matrix} & \text{Control points for} \\ \left[\begin{array}{cccc} ([0], [5], [10]) & ([0], [5], [6]) & ([0], [5], [4]) & ([0], [5], [0]) \\ ([4], [5], [10]) & ([4], [8], [6]) & ([4], [8], [4]) & ([4], [5], [0]) \\ ([6], [5], [10]) & ([6], [8], [6]) & ([6], [8], [4]) & ([6], [5], [0]) \\ ([10], [5], [10]) & ([10], [5], [6]) & ([10], [5], [4]) & ([10], [5], [0]) \end{array} \right] \end{matrix}$$

| Test No. | Number of steps | VNODE tolerance | Maximum relative model space error |
|----------|-----------------|-----------------|------------------------------------|
| 1 | 488 | 10^{-12} | 0.00350 |
| 2 | 646 | 10^{-15} | 0.00234 |
| 3 | 1030 | 10^{-20} | 0.00165 |
| 4 | 1736 | 10^{-25} | 0.00084 |
| 5 | 3009 | 10^{-30} | 0.00043 |
| 6 | 5302 | 10^{-35} | 0.00028 |

Table 8.1: Variation of the model space error with the number of steps for a bicubic Bézier intersection.

8.1.2 Intersection of two Biquadratic Bézier Surfaces

Transversal intersection of two tensor product Bézier patches $[\mathbf{P}_2](\sigma, t)$ and $[\mathbf{Q}_2](u, v)$ is depicted in Figure 8-2. Like the previous example we solve the IVP for ODEs using a validated ODE solver and subsequently obtain the model space error bounds. Figure 8-2 shows two cases where we use different VNODE tolerances. We can see that with the application of a smaller tolerance we can obtain smaller bounds for the error. This shows that the 3D model space error bound converges to the true intersection for small values of the error.

$$[\mathbf{P}_2](\sigma, t) = \begin{matrix} & \text{Control points for} \\ \left[\begin{array}{ccc} ([0], [-50], [5]) & ([0], [0], [100]) & ([0], [50], [0]) \\ ([100], [-25], [0]) & ([100], [0], [0]) & ([100], [25], [0]) \\ ([200], [-75], [0]) & ([200], [0], [100]) & ([200], [75], [0]) \end{array} \right] \end{matrix}$$

$$[\mathbf{Q}_2](u, v) = \begin{matrix} & \text{Control points for} \\ \left[\begin{array}{ccc} ([0], [-50], [5]) & ([0], [0], [50]) & ([0], [50], [0]) \\ ([100], [-100], [0]) & ([100], [0], [300]) & ([100], [100], [0]) \\ ([200], [-75], [0]) & ([200], [0], [50]) & ([200], [75], [0]) \end{array} \right] \end{matrix}$$

In Table 8.2, we find the effect of changing the width of the starting point for various VNODE tolerances, the number of steps required, the maximum relative model space error and the VNODE global error.

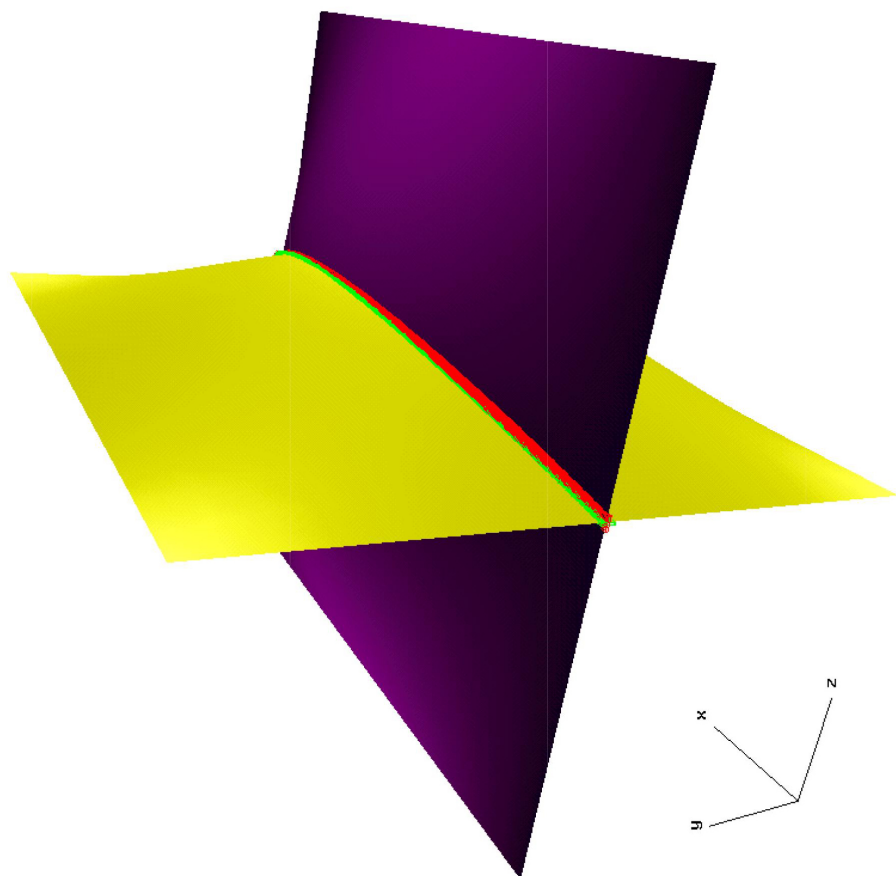


Figure 8-1: Transversal intersection of two bicubic Bézier surfaces corresponding to a maximum relative model space error of 0.00350.

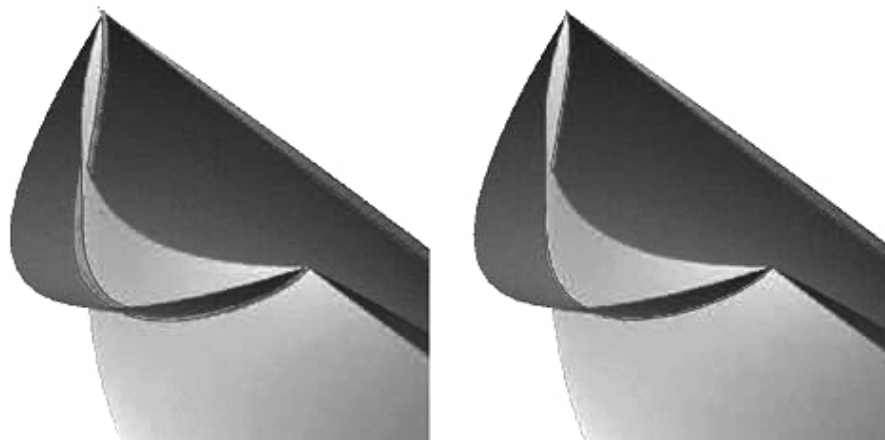


Figure 8-2: Transversal intersection of two tensor product Bézier surface patches. This figure depicts convergence of error bounds.

| Test No. | Time taken (s) | Number of steps | Width of starting point | Maximum relative model space error | VNODE Global Error |
|----------|----------------|-----------------|-------------------------|------------------------------------|-------------------------|
| 1 | 495 | 10116 | 10^{-6} | 0.29 | 1.0107×10^{-4} |
| 2 | 493 | 10045 | 10^{-7} | 0.29 | 1×10^{-6} |
| 3 | 493 | 10038 | 10^{-8} | 0.29 | 9.9989×10^{-8} |
| 4 | 494 | 10037 | 10^{-9} | 0.29 | 1.00×10^{-8} |
| 5 | 491 | 10037 | 10^{-10} | 0.29 | 1.0234×10^{-9} |

Table 8.2: The effect of changing the width of the starting point for the number of steps required, the maximum relative model space error and the VNODE global error for a VNODE tolerance of 10^{-50} .

8.1.3 Intersection with Singularity (Example: 1)

Figure 8-3 shows an example constructed in such a way that there is a singular point in the surface intersection curve segment. The surfaces are $[\mathbf{P}_3](\sigma, t)$ a cubic-quadratic surface and $[\mathbf{Q}_3](u, v)$ a bicubic surface. Tracing the surface intersection in this example would involve separately tracing the four intersection curve segments, given appropriate starting points.

$$\begin{array}{c}
 \text{Control points for} \\
 [\mathbf{P}_3](\sigma, t) = \begin{bmatrix}
 ([-10], [-30], [3]) & ([0], [-30], [-3]) & ([10], [-30], [3]) \\
 ([-10], [-10], [3]) & ([0], [-10], [-3]) & ([10], [-10], [3]) \\
 ([-10], [10], [3]) & ([0], [10], [-3]) & ([10], [10], [3]) \\
 ([-10], [30], [3]) & ([0], [30], [-3]) & ([10], [15], [3])
 \end{bmatrix} \\
 \\
 \text{Control points for } [\mathbf{Q}_3](u, v) = \\
 \begin{bmatrix}
 ([20], [-10], [3]) & ([10], [-10], [3]) & ([-10], [-10], [3]) & ([-20], [-10], [3]) \\
 ([20], [-10], [-1]) & ([10], [-10], [-1]) & ([-10], [-10], [-1]) & ([-20], [-10], [-1]) \\
 ([20], [10], [-1]) & ([10], [10], [-1]) & ([-10], [10], [-1]) & ([-20], [10], [-1]) \\
 ([20], [10], [3]) & ([10], [10], [3]) & ([-10], [10], [3]) & ([-20], [10], [3])
 \end{bmatrix}
 \end{array}$$

The four starting points are listed below.

$$\begin{aligned}
 \begin{bmatrix} [\sigma_0] & [t_0] & [u_0] & [v_0] \end{bmatrix}^T &= \begin{bmatrix} [0.666, 0.667] & [0.000, 0.000] & [1.000, 1.000] & [0.750, 0.750] \end{bmatrix}^T, \\
 \begin{bmatrix} [\sigma_0] & [t_0] & [u_0] & [v_0] \end{bmatrix}^T &= \begin{bmatrix} [0.333, 0.334] & [0.000, 0.000] & [1.000, 1.000] & [0.750, 0.750] \end{bmatrix}^T, \\
 \begin{bmatrix} [\sigma_0] & [t_0] & [u_0] & [v_0] \end{bmatrix}^T &= \begin{bmatrix} [0.333, 0.334] & [1.000, 1.000] & [0.000, 0.000] & [0.250, 0.250] \end{bmatrix}^T, \\
 \begin{bmatrix} [\sigma_0] & [t_0] & [u_0] & [v_0] \end{bmatrix}^T &= \begin{bmatrix} [0.666, 0.667] & [1.000, 1.000] & [1.000, 1.000] & [0.250, 0.250] \end{bmatrix}^T.
 \end{aligned}$$

A detailed description of the various cases which involve perturbation from this aligned position is described in detail in the Section 8.4.

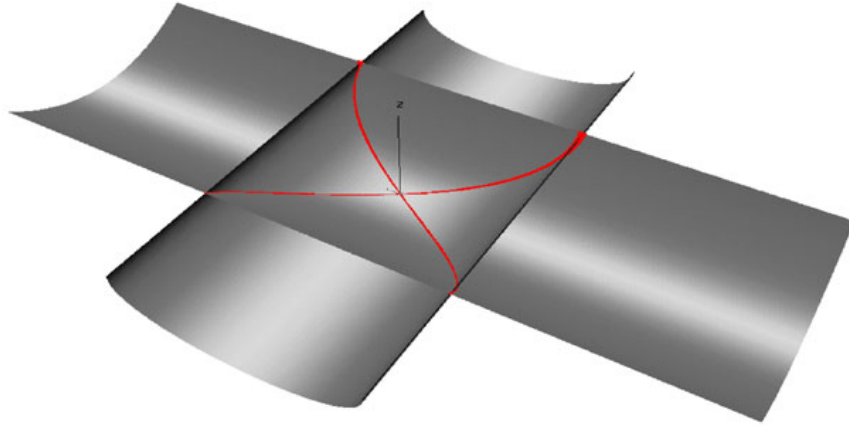


Figure 8-3: An example of transversal intersection with a singular point involving tracing four separate intersection curve segments.

8.1.4 Intersection with Singularity (Example: 2)

The surfaces involved are a hyperbolic surface $[\mathbf{P}_4](\sigma, t)$ and $[\mathbf{Q}_4](u, v)$ a plane. Refer Figure 8-4. The plane is arranged such that it touches the hyperbolic point of the surface $[\mathbf{P}_4](\sigma, t)$. That is when the normals to the surfaces are perfectly aligned there is a singular point in the surface intersection curve segment. Tracing the intersection now involves separately tracing the four intersection curve segments, given appropriate starting points. The control points of the hyperbolic surface $[\mathbf{P}_4](\sigma, t)$ are given below.

$$\begin{array}{c}
 \text{Control points for} \\
 [\mathbf{P}_4](\sigma, t) = \begin{bmatrix}
 ([-50], [0], [0]) & ([0], [0], [100]) & ([50], [0], [0]) \\
 ([-50], [100], [0]) & ([0], [100], [0]) & ([50], [100], [0]) \\
 ([-50], [200], [0]) & ([0], [200], [100]) & ([50], [200], [0])
 \end{bmatrix}
 \end{array}$$

The four starting points are listed below.

$$\begin{aligned}
 [\sigma_0 \quad t_0 \quad u_0 \quad v_0]^T &= [0.85355339, 0.85355340 \quad 0.000000, 0.000000 \quad 0.85355339, 0.85355340 \quad 0.000000, 0.000000]^T, \\
 [\sigma_0 \quad t_0 \quad u_0 \quad v_0]^T &= [0.146446609, 0.14644661 \quad 1.000000, 1.000000 \quad 0.146446609, 0.14644661 \quad 1.000000, 1.000000]^T, \\
 [\sigma_0 \quad t_0 \quad u_0 \quad v_0]^T &= [0.146446609, 0.14644661 \quad 0.000000, 0.000000 \quad 0.146446609, 0.14644661 \quad 0.000000, 0.000000]^T, \\
 [\sigma_0 \quad t_0 \quad u_0 \quad v_0]^T &= [0.85355339, 0.85355340 \quad 1.000000, 1.000000 \quad 0.85355339, 0.85355340 \quad 1.000000, 1.000000]^T.
 \end{aligned}$$

We tabulate (refer to table 8.3) the number of steps needed and the time taken to trace one of the four branches for varying VNODE tolerance.

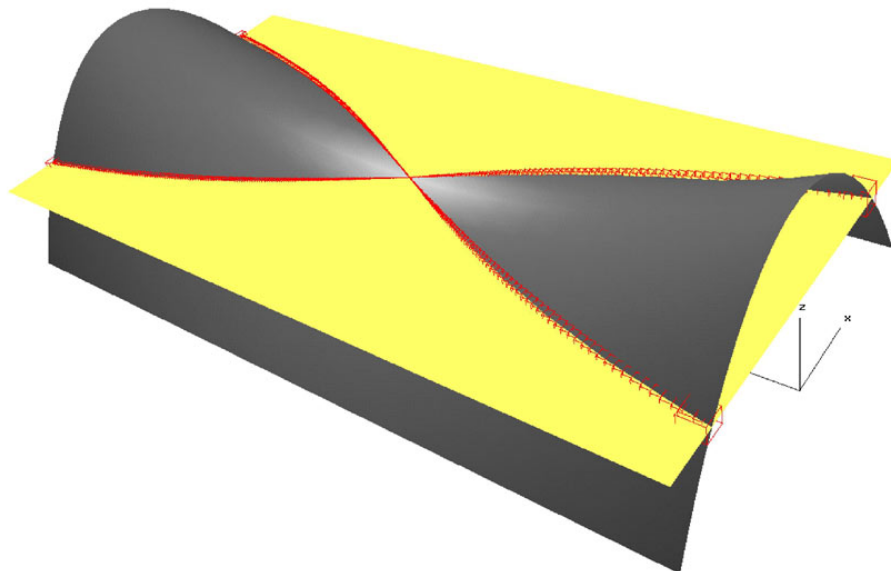


Figure 8-4: Transversal intersection of a hyperbolic surface and a plane involving tracing four separate intersection curve segments.

| Test No. | Number of steps | VNODE tolerance | Time taken (s) |
|----------|-----------------|-----------------|----------------|
| 1 | 224 | 10^{-15} | 9.89 |
| 2 | 355 | 10^{-20} | 15.6 |
| 3 | 582 | 10^{-25} | 25.6 |
| 4 | 974 | 10^{-30} | 42.7 |
| 5 | 1662 | 10^{-35} | 72.8 |
| 6 | 2882 | 10^{-40} | 126 |

Table 8.3: The number of steps needed, the time taken for various VNODE tolerances to trace one of the four branches of the intersection of a hyperbolic surface and a plane.

8.1.5 Torus-Cylinder Intersection (Trigonometric Functions)

The surfaces, $[\mathbf{P}_5]$ a torus and $[\mathbf{Q}_5]$ a cylinder are defined by trigonometric functions. We trace one of the four loops of the curves of intersection. We can apply our proposed algorithm. The continuity of the mapped boxes is ensured by rational continuous functions like sine or cosine. Figure 8-5 shows a torus and a cylinder intersecting. The maximum relative model space error is 0.0187.

$$[\mathbf{P}_5](\sigma, t) = \begin{bmatrix} \cos(2\pi\sigma) \cdot [10 + 5\cos(2\pi t)] \\ \sin(2\pi\sigma) \cdot [10 + 5\cos(2\pi t)] \\ 5 \cdot \sin(2\pi t) \end{bmatrix}$$

$$[\mathbf{Q}_5](u, v) = \begin{bmatrix} 2\cos(2\pi u) \\ 40v - 20 \\ 2\sin(2\pi u) \end{bmatrix}$$

We vary the error bound with respect to the number of steps by using various VNODE tolerances. The results are tabulated in Table 8.4. Another test was done where we study the effect of variation of the width of the starting point on the width of the error bound. The result is shown as a graph Figure 8-6, from which we notice that the VNODE global error bound is strongly dependent on the width of the initial point.

| Test No. | Time taken (s) | Number of steps | VNODE tolerance | Maximum relative model space error |
|----------|----------------|-----------------|-----------------|------------------------------------|
| 1 | 11.7 | 80 | 10^{-12} | 0.0187 |
| 2 | 15 | 108 | 10^{-15} | 0.0135 |
| 3 | 24.1 | 188 | 10^{-20} | 0.0076 |
| 4 | 41.1 | 338 | 10^{-25} | 0.0041 |
| 5 | 136 | 1166 | 10^{-35} | 0.00115 |
| 6 | 490 | 4254 | 10^{-45} | 0.00031 |
| 7 | 1843 | 16089 | 10^{-55} | 0.00008 |

Table 8.4: Table comparing maximum relative model space error bound with the time taken, number of steps required and the VNODE tolerance for an intersection involving a torus and a cylinder.

8.2 Tangential Intersection of Surfaces

8.2.1 Planar Intersection Curve

Figure 8-7 shows the tangential intersection of two interval Bézier surfaces $[\mathbf{P}_6](\sigma, t)$ and $[\mathbf{Q}_6](u, v)$. The surfaces are intersecting with each other tangentially. Degenerate intervals represent the control points of the surfaces. The pre-image of the curve of

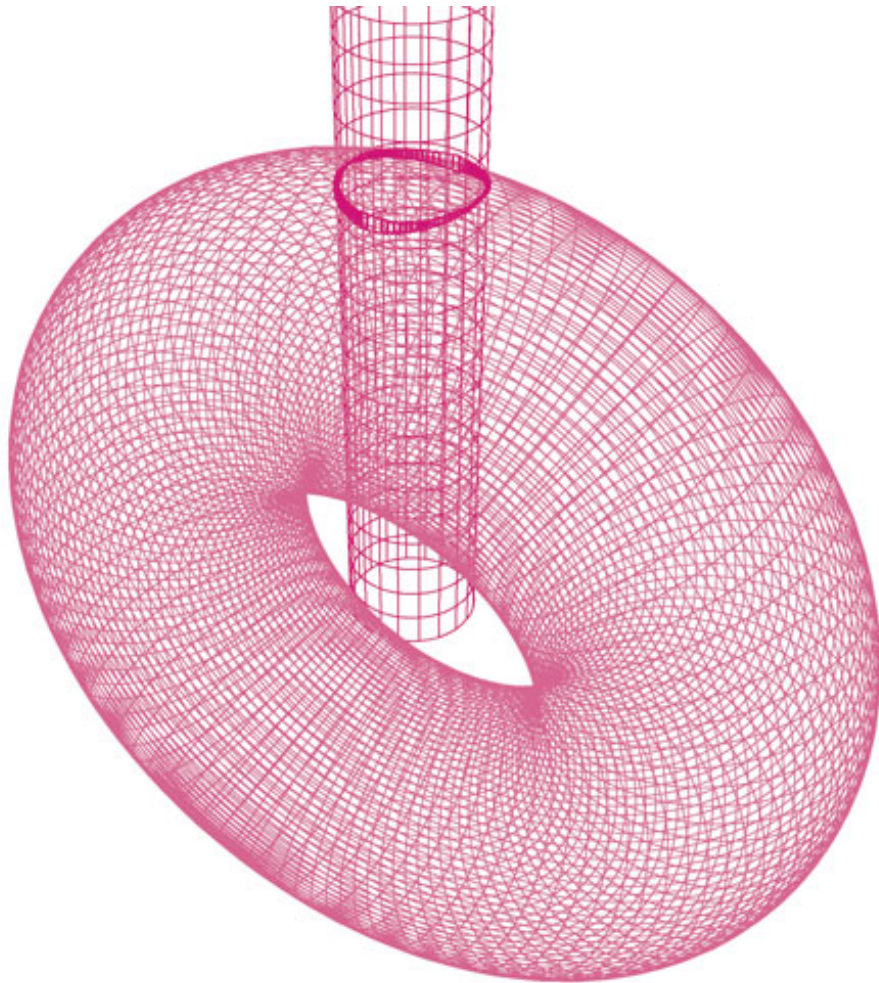


Figure 8-5: Transversal intersection of a Torus and a Cylinder.

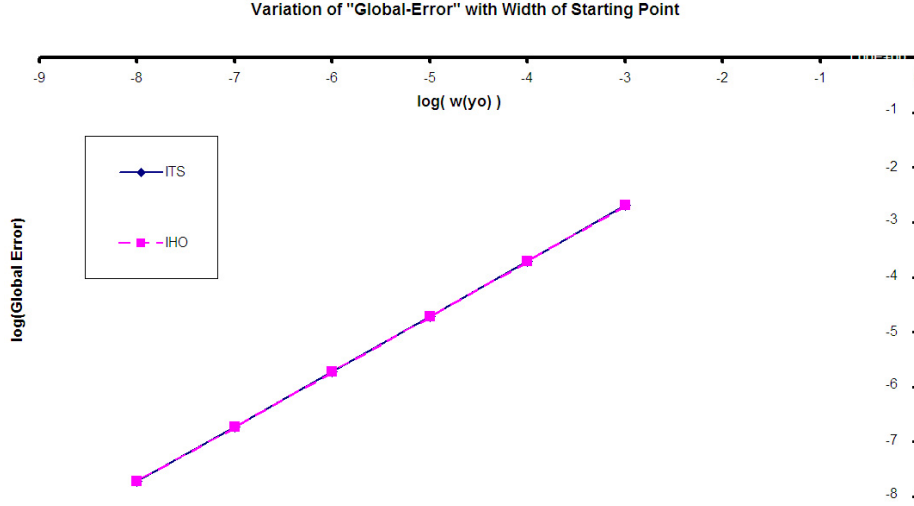


Figure 8-6: Dependence of error in parametric space on the width of starting point for a transversal intersection involving a Torus and a Cylinder.

intersection is obtained using the validated ODE solver and mapped into model space. In Table 8.5 we have tabulated the number of steps and the corresponding maximum relative model space error.

$$\text{Control points for } [\mathbf{P}_6](\sigma, t) = \begin{bmatrix} ([0], [0], [0]) & ([5], [0], [10]) & ([10], [0], [0]) \\ ([5], [5], [0]) & ([10], [5], [10]) & ([15], [5], [0]) \\ ([5], [10], [0]) & ([10], [10], [10]) & ([15], [10], [0]) \\ ([0], [15], [0]) & ([5], [15], [10]) & ([10], [15], [0]) \end{bmatrix}$$

$$\text{Control points for } [\mathbf{Q}_6](u, v) = \begin{bmatrix} ([0], [0], [10]) & ([5], [0], [0]) & ([10], [0], [10]) \\ ([5], [5], [10]) & ([10], [5], [0]) & ([15], [5], [10]) \\ ([5], [10], [10]) & ([10], [10], [0]) & ([15], [10], [10]) \\ ([0], [15], [10]) & ([5], [15], [0]) & ([10], [15], [10]) \end{bmatrix}$$

8.2.2 Non-Planar Intersection Curve

Figure 8-8 represents the intersection of two tensor product Bézier patches $[\mathbf{P}_7](\sigma, t)$ and $[\mathbf{Q}_7](u, v)$. The patches are positioned in such a way that they are tangential to each other and their curve of intersection is a 3D curve. The control points are represented as degenerate intervals and are provided as input to a validated ODE solver. The enclosure containing the curve of intersection is mapped from the parameter space to the 3D model space and we obtain rigorous bounds in the 3D model space, which guarantee to contain the true curve of intersection with a maximum relative model space error of 0.002.

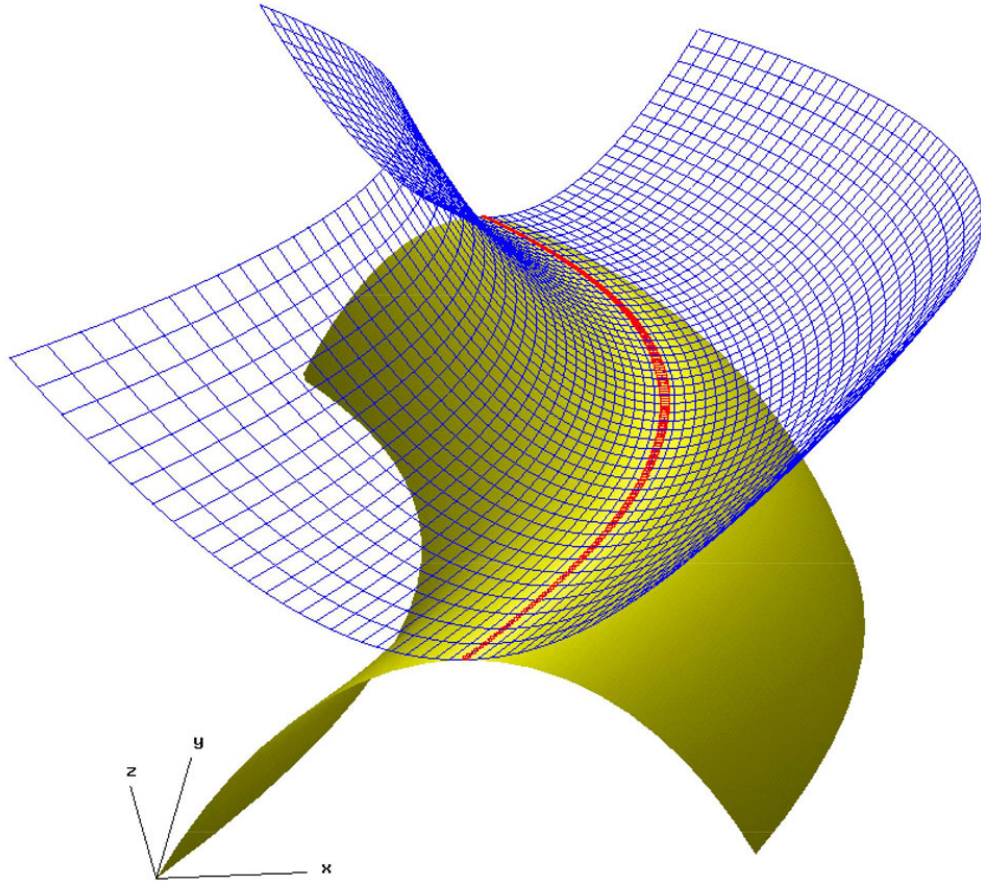


Figure 8-7: Tangential intersection of two cubic-quadratic Bézier patches for a maximum relative model space error = 0.0050. Note that the control points of the surfaces are chosen such that the curve of intersection lies on a plane.

| Test No. | Number of steps | Maximum relative model space error |
|----------|-----------------|------------------------------------|
| 1 | 530 | 0.005 |
| 2 | 806 | 0.003 |
| 3 | 1261 | 0.0015 |
| 4 | 2022 | 0.001 |
| 5 | 3305 | 0.0006 |
| 6 | 5496 | 0.0004 |

Table 8.5: Variation of the model space error with the number of steps for the tangential intersection of two surfaces. The intersection curve lies on a plane.

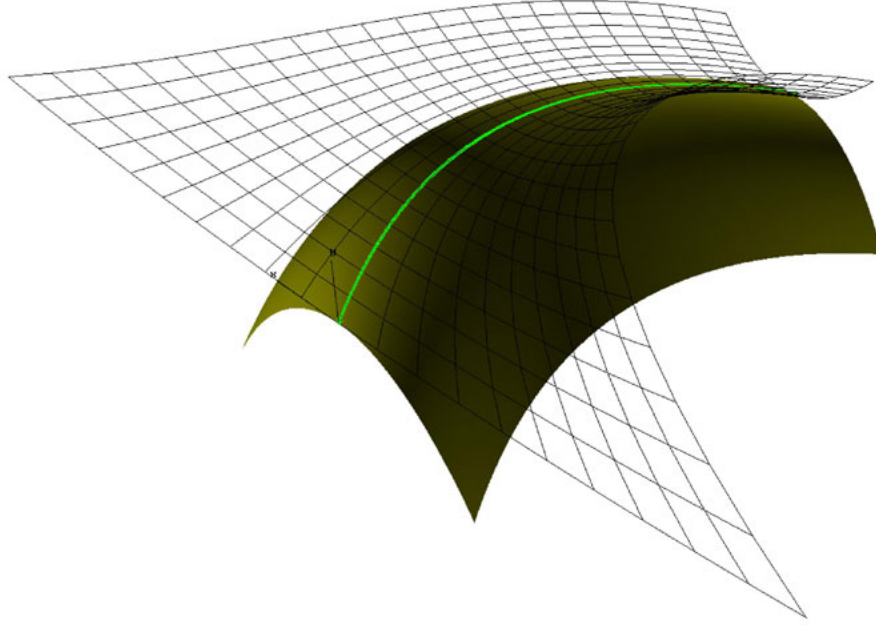


Figure 8-8: Tangential intersection of tensor product Bézier surface patches. Control points of surfaces are chosen such that the curves of intersection do not lie on a plane.

$$\begin{array}{c}
 \text{Control points for} \\
 [\mathbf{P}_7](\sigma, t) = \left[\begin{array}{ccc}
 ([-150], [10], [0]) & ([0], [-10], [0]) & ([150], [10], [0]) \\
 ([-10], [40], [50]) & ([20], [0], [50]) & ([50], [40], [50]) \\
 ([-10], [40], [110]) & ([20], [0], [110]) & ([50], [40], [110]) \\
 ([-55], [-5], [200]) & ([-25], [-45], [200]) & ([5], [-5], [200])
 \end{array} \right]
 \end{array}$$

$$\begin{array}{c}
 \text{Control points for} \\
 [\mathbf{Q}_7](u, v) = \left[\begin{array}{ccc}
 ([-50], [-50], [0]) & ([0], [50], [0]) & ([50], [-50], [0]) \\
 ([-30], [-30], [50]) & ([20], [70], [50]) & ([70], [-30], [50]) \\
 ([-30], [-30], [110]) & ([20], [70], [110]) & ([70], [-30], [110]) \\
 ([-75], [-75], [200]) & ([-25], [25], [200]) & ([25], [-75], [200])
 \end{array} \right]
 \end{array}$$

In Table 8.6 we have tabulated the number of steps, time taken, VNODE tolerances and the corresponding maximum relative model space error.

We however note that the strict distinction of a transversal and tangential intersection is blurred when we are talking about interval surfaces and interval methods for resolving the intersections. This is partly due to the inherent trouble with the floating point arithmetic scheme we use in computers. The input data is in some way or the other perturbed and hence the intersection which initially was tangential is no longer exactly tangential, but could be transversal or even result in no intersection.

| Test No. | Time taken (s) | Number of steps | VNODE tolerance | Maximum relative model space error |
|----------|----------------|-----------------|-----------------|------------------------------------|
| 1 | 249 | 858 | 10^{-18} | 0.66 |
| 2 | 293 | 1011 | 10^{-20} | 0.459 |
| 3 | 450 | 1556 | 10^{-25} | 0.249 |
| 4 | 712 | 2459 | 10^{-30} | 0.135 |

Table 8.6: Variation of the model space error with the number of steps for the tangential intersection of two surfaces. The intersection curve does not lie on a plane.

8.3 Self-Intersection of Surfaces

A validated ODE solver can tremendously increase the efficiency in solving self-intersection problems compared to the IPP solver. This is because a validated ODE solver does not suffer from the difficulty arising from the removal of the trivial solutions from the real solutions.

8.3.1 Self-Intersection of a Bicubic Bézier Patch

A test is performed with a bicubic Bézier surface $[\mathbf{P}_8](\sigma, t)$ which has self-intersections. The control points of the surface are degenerate intervals given below. Figure 8-9 depicts the surface with the bounds on the intersection curve. We also study the number of steps required for a given VNODE tolerance of 10^{-25} vs. the variation of the order of the Taylor's series used in the expansion in VNODE. From this we can conclude that it is always better to use a high order Taylor's series as this would result in lesser number of steps and hence the time taken. A previous correspondence from Prof. Nedialkov indicates that for nonlinear systems there is an improvement in the performance of the validated interval schemes till an optimal order for the Taylor series after which the performance again falls. Such a trend is expected here too.

$$[\mathbf{P}_8](\sigma, t) = \begin{array}{c} \text{Control points for} \\ \left[\begin{array}{cccc} ([0], [0], [20]) & ([0], [5], [20]) & ([0], [10], [20]) & ([0], [15], [20]) \\ ([20], [0], [0]) & ([25], [5], [5]) & ([25], [10], [5]) & ([25], [15], [5]) \\ ([-10], [0], [0]) & ([-10], [5], [0]) & ([-10], [10], [0]) & ([-10], [15], [0]) \\ ([10], [0], [20]) & ([25], [5], [20]) & ([25], [10], [20]) & ([25], [15], [20]) \end{array} \right] \end{array}$$

8.4 Resolving Straying and Looping

The validated ODE solver is applied in the context of resolving straying or looping during tracing the intersection curve segment. A detailed discussion of the theory is given in Section 5.5.

| Test No. | Order of Taylor Series | Number of Steps | VNODE tolerance | Time taken (s) |
|----------|------------------------|-----------------|-----------------|----------------|
| 1 | 10 | 3291 | 10^{-25} | 116 |
| 2 | 15 | 1469 | 10^{-25} | 74.4 |
| 3 | 17 | 1031 | 10^{-25} | 65 |
| 4 | 19 | 789 | 10^{-25} | 60.4 |

Table 8.7: Number of steps required for tracing the curve of intersection of the surface for different orders of the Taylor series. Note that the VNODE tolerance is kept constant at 1×10^{-25} .

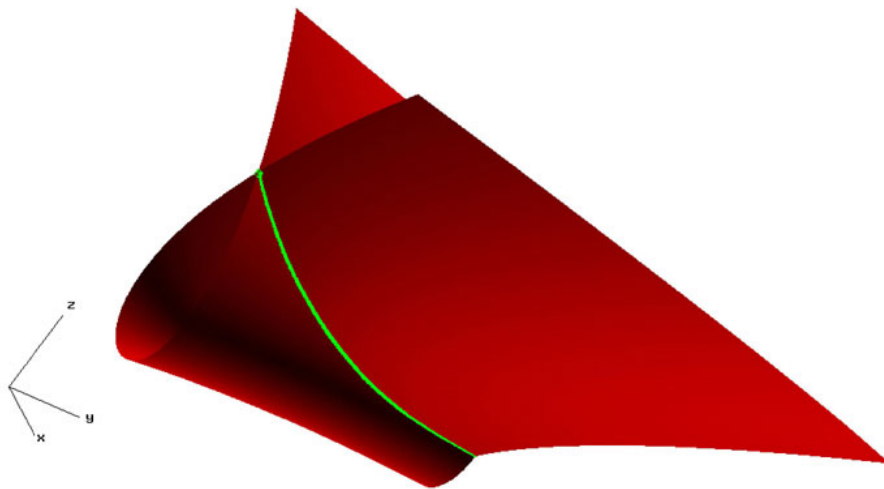


Figure 8-9: Self-intersection of a bicubic Bézier patch.

8.4.1 Example 1 Depicting Resolving Straying and Looping

Figure 8-16 shows an example constructed very similar to the one used in Hu *et al.* [13]. When the two interval Bézier surfaces of Section 8.1.3, $[\mathbf{P}_3](\sigma, t)$ and $[\mathbf{Q}_3](u, v)$ intersect, then there is a singularity in the intersection curve (curve is connected).

Application of a conventional ODE system solver, such as the Runge-Kutta or Adams-Bashforth methods, for solving the IVP corresponding to the intersection would involve the following pathologies:

1. Specifying a starting point which is approximate would mean that the curve traced would not have the singularity or bifurcation. The B-rep model generated would lose topological information and the result may further cause failure in CAD model processing.
2. Straying or looping near the singular region, which are essentially related to the uncertainty of the solver in taking a specific step.

Ideally given a starting point $[\sigma_0, t_0, u_0, v_0]^T = [\frac{1}{3}, 0, 0, \frac{3}{4}]^T$, we expect a solver to notify us as it approaches a region close to the singularity. The use of the recommended solvers in *Matlab* such as *ode45* (an implementation of Runge-Kutta method) and *ode113* (implementation of Adam's method) would result in an abnormal behavior as shown in Figures 8-10 and 8-11. We show in Figure 8-12 the behavior of a validated ODE solver which does not march across the singularity. Thus the intersection is traced by separately tracing all the four intersection curve segments.

Now consider the case when one of the surfaces in Figure 8-3 is perturbed by a small amount in z-direction such that the intersection curves have different topological configuration. The intersection is now just two separate curve segments, even though they lie very close to each other near the previously singular region.

Conventional methods show poor behavior near the region where two intersection curves are very close to each other. This is shown by Figures 8-13 and 8-14 obtained using the Adams-Bashforth and Runge-Kutta methods respectively. Note the inconsistency in topology of the intersection curves obtained from conventional methods. The validated ODE solver uses an adaptive step size strategy, easily resolves this case, and behaves well locally close to the near-singular region as shown by Figure 8-15.

Usual floating point representations can cause an intersection curve in examples 8.4.1 and 8.4.2 to become two separate curves which have conflicting topological structure from the real curve. These violations are manifested as gaps or as inappropriate intersections. This example illustrates that, a validated ODE solver should be able to resolve the singularity of the intersection curve and report to the user. We have performed experiments where we perturb one of the surfaces ($[\mathbf{Q}_3](u, v)$) in the z-direction (direction of the common normal at the singular point on the intersection) so that the curve of intersection is free of singularity. A positive perturbation in z-direction will lead to the branching of the curve in a sense different from a negative perturbation in the z-direction. The validated ODE solver traces the correct curve and provides us with error bounds even through regions where the conventional method fails. Table 8.8 compares the number of steps needed to resolve a possible

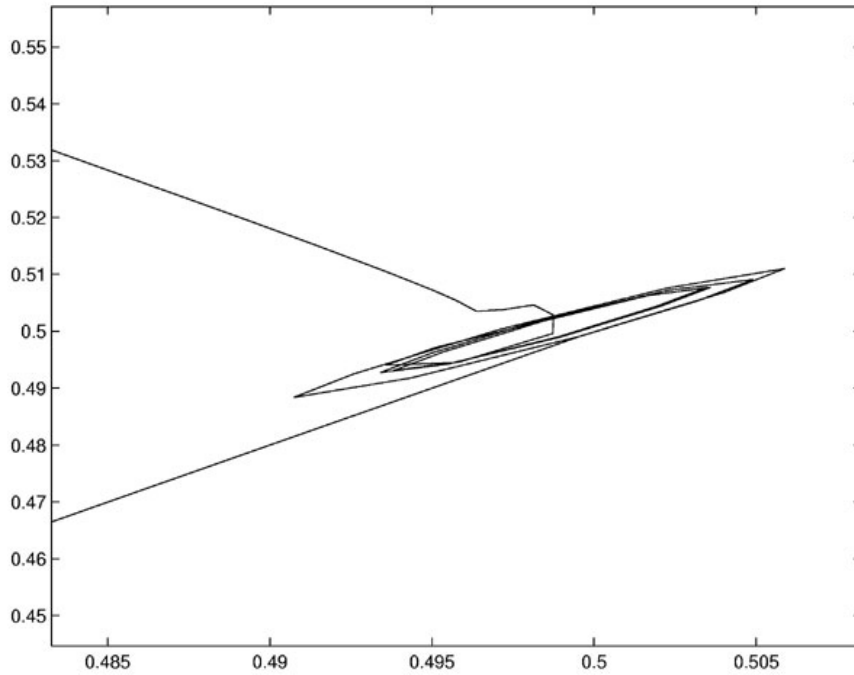


Figure 8-10: Integration using *ode45* in Matlab. Looping is seen at the region close to the singularity in the σ, t parameter space.

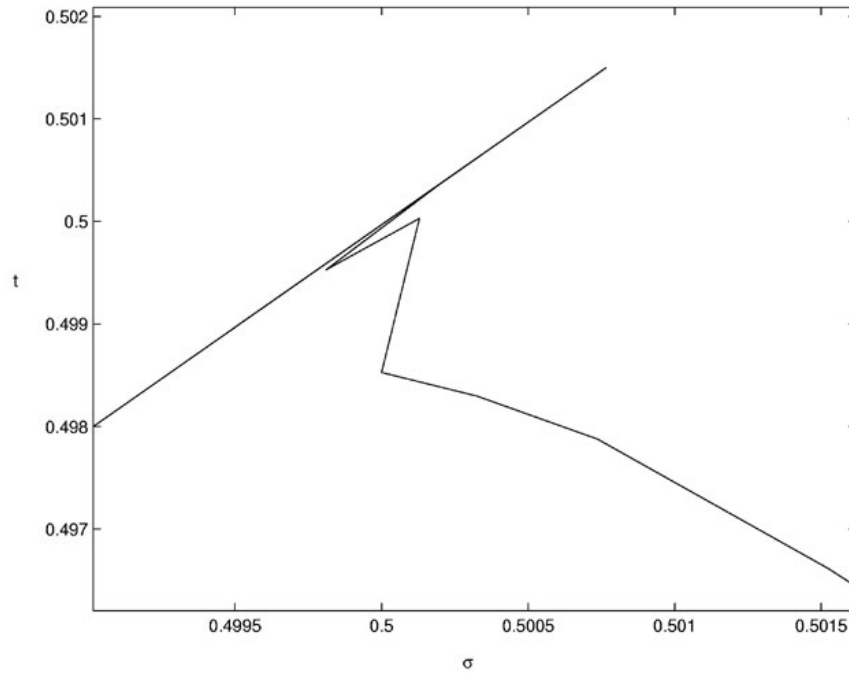


Figure 8-11: Integration using *ode113* in Matlab. Straying and looping is seen at the region close to the singularity in the σ, t parameter space.

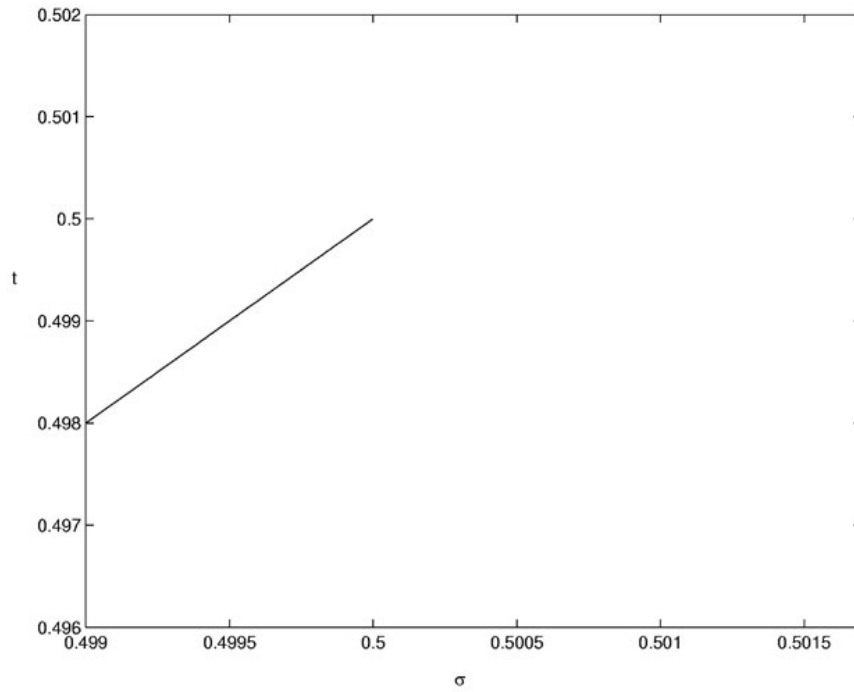


Figure 8-12: Integration using a validated ODE solver, not crossing the singular region in σ, t parameter space.

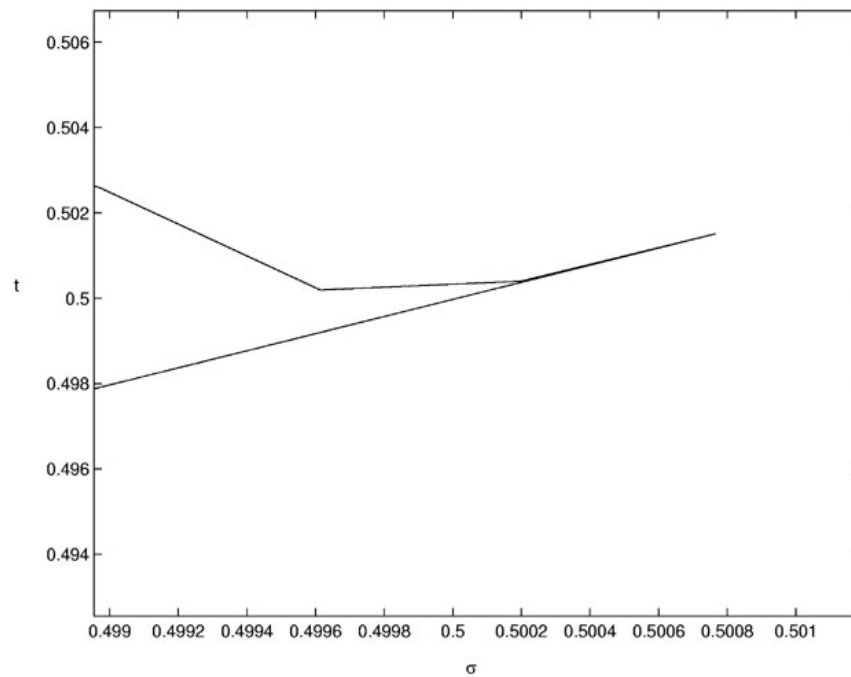


Figure 8-13: Result from *ode113* in Matlab in the σ, t parameter space. This experiment is done for the case of a small perturbation of one of the surface $[\mathbf{Q}_3](u, v)$.

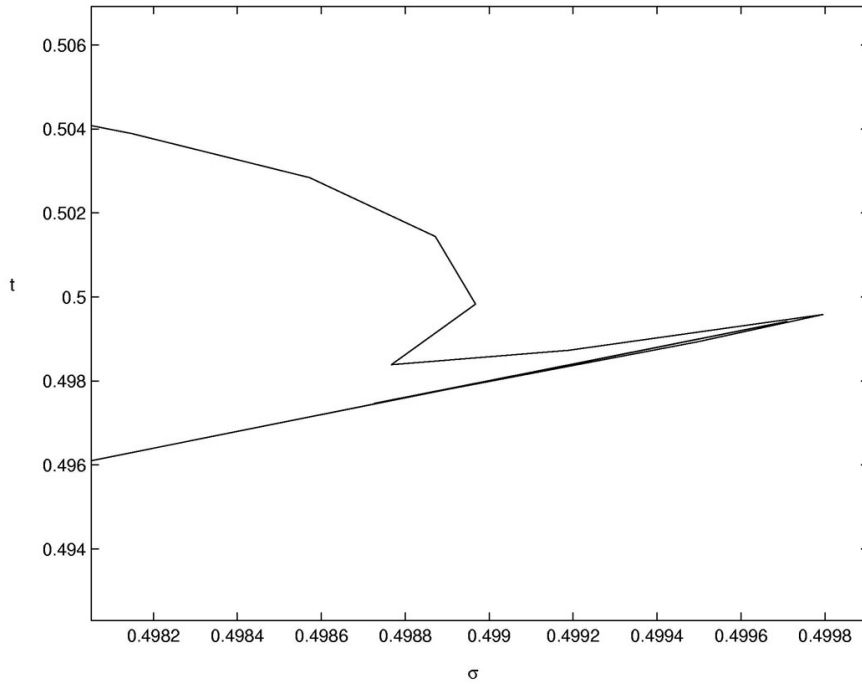


Figure 8-14: Result from *ode45* in Matlab in the σ, t parameter space. This experiment is done for the case of a small perturbation of one of the surface $[\mathbf{Q}_3](u, v)$.

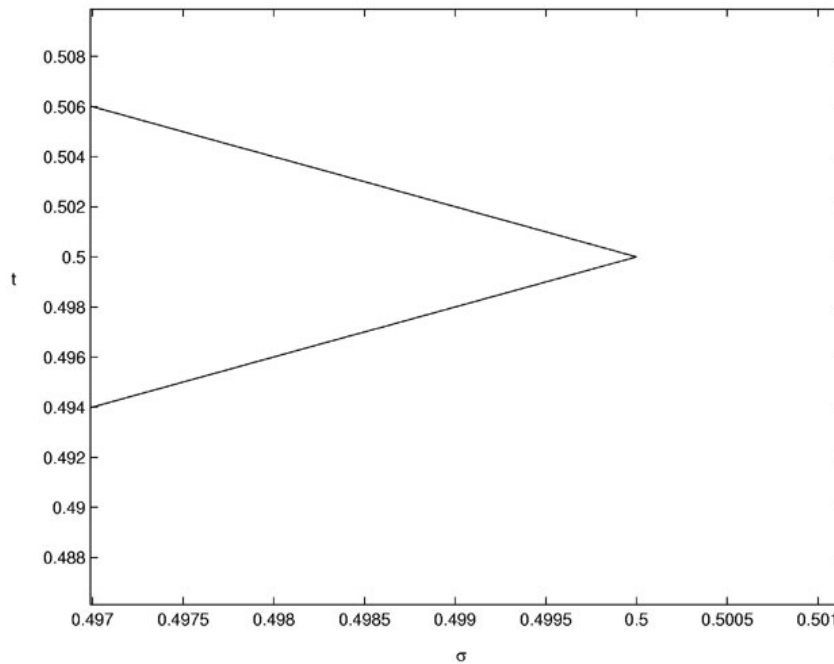


Figure 8-15: Result from a validated ODE solver in the σ, t parameter space for a case involving a small perturbation of one of the surface $[\mathbf{Q}_3](u, v)$.

candidate for looping or straying vs. perturbation of the Bézier patches. Note that when the perturbations are small we need more steps for resolving.

| Test No. | Perturbation of $[\mathbf{Q}_3]$ in z-direction in model space | Steps required by validated ODE solver |
|----------|--|--|
| 1 | $+0.03$ | 845 |
| 2 | $+0.0003$ | 989 |
| 3 | $+0.000003$ | 1139 |
| 4 | 0.0 | <i>singularity reported</i> |
| 5 | -0.000003 | 1303 |
| 6 | -0.0003 | 1153 |
| 7 | -0.03 | 1007 |

Table 8.8: Resolving singularities of the curve of intersection for the intersection of a bi-cubic surface and a cubic-quadratic surface. Table shows the perturbations along the common normal and the corresponding number of steps needed to trace the intersection.

8.4.2 Example 2 Depicting Resolving Straying and Looping

The intersection of two interval Bézier surfaces $[\mathbf{P}_4](\sigma, t)$ and $[\mathbf{Q}_4](u, v)$ in Section 8.1.4 is also tested for checking our claim of resolving straying and looping. We perturb one of the surfaces, the plane $[\mathbf{Q}_4](u, v)$ along the common normal near the hyperbolic point such that the intersection curve has a completely different behavior if the direction of the perturbation (z-direction) changes. A positive perturbation in z-direction will lead to the branching of the curve in a sense different from a negative perturbation in the z-direction. Table 8.9 compares the number of steps needed to resolve a possible candidate for looping or straying vs. perturbation of the Bézier patches. Note that when the perturbations are small we need more steps for resolving.

8.5 A Difficult Case

The example shown in Figure 8-18 involves the case of two surfaces $[\mathbf{P}_9](\sigma, t)$ and $[\mathbf{Q}_9](u, v)$ designed such that the intersection is tangential along an iso-parametric line. Also there is a transversal intersection of the two surfaces and hence the two intersection curves intersect. Thus for obtaining the intersection of the intersection curves the integration using the equations for tangential intersection fails even using a validated ODE solver.

The control points of the surfaces are given below.

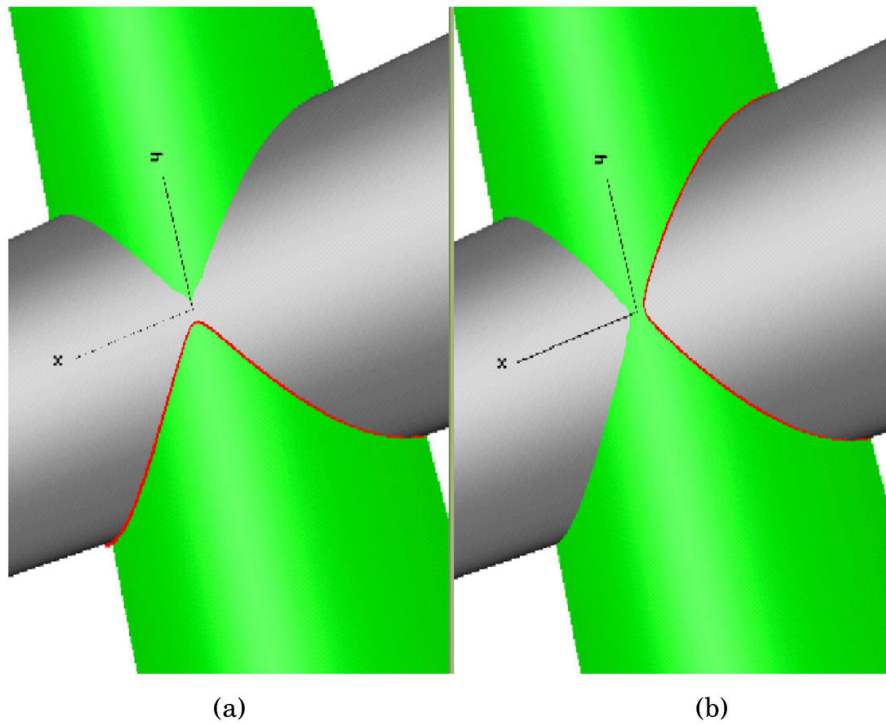


Figure 8-16: Example depicting how validated ODE solver prevents straying and looping. Figure (a) shows the surface $[\mathbf{Q}_3](u, v)$ perturbed along the positive z-direction, the intersection curve segment is correctly traced by the validated ODE solver. Figure (b) in a similar way illustrates how the validated ODE solver successfully trace the correct intersection curve segment when the perturbation is in the negative z-direction.

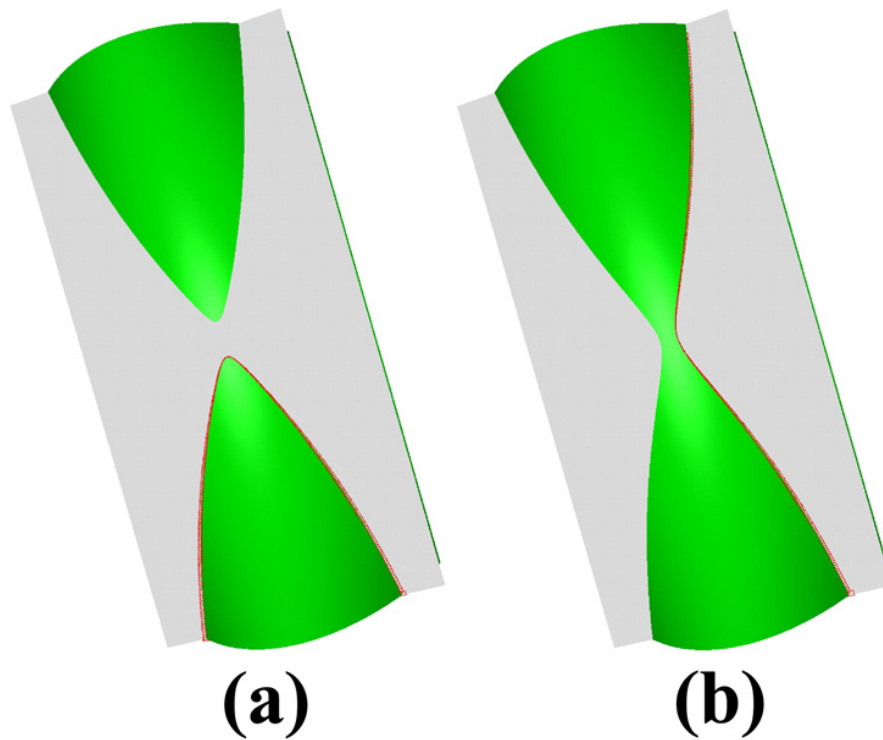


Figure 8-17: Resolving straying and looping of curve of intersection for the intersection of a hyperbolic surface $[\mathbf{P}_4](u, v)$ and a plane $[\mathbf{Q}_4](u, v)$. Figure (a) shows the plane $([\mathbf{Q}_4](u, v))$ perturbed along the positive z-direction, the intersection curve segment is correctly traced by the validated ODE solver. Figure (b) in a similar way illustrates how the validated ODE solver successfully trace the correct intersection curve segment when the perturbation is in the negative z-direction.

| Test No. | Perturbation of $[Q_4]$ in z-direction in model space | Steps required by validated ODE solver | Time taken(s) |
|----------|---|--|---------------|
| 1 | $+0.1$ | 426 | 14.1 |
| 2 | $+0.001$ | 585 | 19.3 |
| 3 | $+0.00001$ | 748 | 24.4 |
| 4 | $+0.0000001$ | 913 | 29.9 |
| 5 | 0.0 | <i>singularity reported</i> | - |
| 6 | -0.0000001 | 826 | 27.0 |
| 7 | -0.00001 | 660 | 21.9 |
| 8 | -0.001 | 495 | 16.2 |
| 9 | -0.1 | 331 | 11.1 |

Table 8.9: Resolving singularities of the curve of intersection for the intersection of a hyperbolic surface and a plane. We tabulate the perturbations along the common normal (z-axis) and the steps needed to trace the intersection. A constant VNODE tolerance of 1×10^{-20} was used.

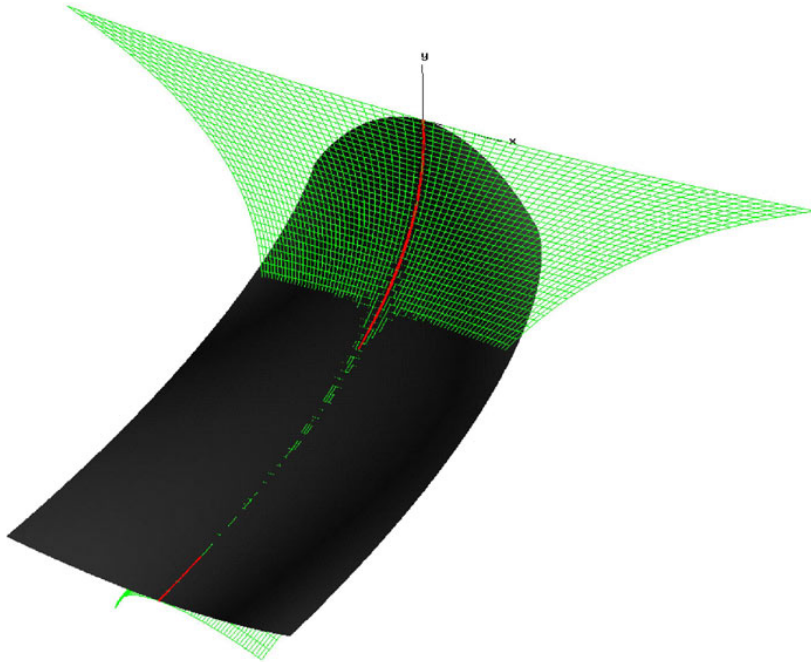


Figure 8-18: Figure shows an intersection where the intersection becomes difficult to solve as the governing differential equation fail.

$$\begin{array}{c}
\text{Control points for} \\
[\mathbf{P}_7](\sigma, t) = \left[\begin{array}{ccc}
([-150], [10], [0]) & ([0], [-10], [0]) & ([150], [10], [0]) \\
([-10], [5], [50]) & ([20], [0], [50]) & ([50], [5], [50]) \\
([-10], [-5], [110]) & ([20], [0], [110]) & ([50], [-5], [110]) \\
([-55], [-75], [200]) & ([-25], [25], [200]) & ([5], [-75], [200])
\end{array} \right]
\end{array}$$

$$\begin{array}{c}
\text{Control points for} \\
[\mathbf{Q}_7](u, v) = \left[\begin{array}{ccc}
([-50], [-50], [0]) & ([0], [50], [0]) & ([50], [-50], [0]) \\
([-30], [0], [50]) & ([20], [5], [50]) & ([70], [0], [50]) \\
([-30], [0], [110]) & ([20], [-5], [110]) & ([70], [0], [110]) \\
([-75], [-5], [200]) & ([-25], [-45], [200]) & ([25], [-5], [200])
\end{array} \right]
\end{array}$$

Chapter 9

Conclusions

9.1 Conclusions

Investigating the effects of floating point arithmetic on the implementation of intersection algorithms has been an important area for basic research during the last decade [29]. Methods to enhance the precision of intersection computation, to monitor numerical error contamination and alternate means of performing arithmetic, not relying on imprecise floating point computation alone, have been previously explored in some detail. A different direction of research involves the use of non-conventional interval methods like a validated ODE solver discussed in this thesis, which considers errors arising from computation as well as initial conditions. It provides a guaranteed bound which encloses the exact solution, and fits well with the concept of robust interval solid modeling [13, 34]. Parts of this thesis is also discussed in greater detail in the papers [23, 30].

Given two RPP surfaces, and a tolerance on the error in the model space, we are now in a position to obtain strict bounds on the entire curve of intersection. This problem of obtaining strict (validated) bounds on the intersection is subdivided into simply stated problems of:

1. Obtaining a corresponding tolerance in the parameter space.
2. Solving an IVP for ODEs using a validated interval ODE solving scheme to obtain a series of bounds in the parameter space of each of the surface which encloses the pre-image of the true solution within the given tolerance in the parameter space.
3. Mapping the enclosures in parameter space to the model space to obtain validated error bounds for the intersection in the 3D model space.

In this thesis we propose two algorithms for obtaining error thresholds in parameter space given the error thresholds in the model space. We apply validated interval ODE solving schemes (IHO and ITS) to obtain bounds on the entire intersection in parameter space. Further, theorems are proved which would enable us to obtain strict

error bounds in the 3D model space. Again we show the use of a validated interval scheme in resolving the phenomenon of *straying* and *looping*.

One feasible way for obtaining the validated representation for solids in the context of floating point arithmetic is to use an interval boundary representation developed by [13, 34]. A key requirement of the interval boundary representation scheme is to have a continuous gap-free bound on the intersection of two surfaces. We fulfill this condition of a continuous gap-free boundary with a numerically verified upper bound for the intersection curve error. The theorems we state in this thesis help us to map this error bound in parameter space to 3D model space bound conservatively. This definite upper bound for error is crucial in defining well formed boundary representation of complex 3D solids using interval boundary representation developed at MIT [13, 34].

With the application of a validated ODE solver, we are at a position to clearly distinguish between the questions of incidence and non-incidence in a point test, which would eventually lead to consistent representations with given bounds for the error. This enables us to realize a gap-free boundary. Using tools developed in Chapter 7, we can monotonically control the error bounds in model space. Further as a by-product of developing a gap-free bound for error, we are able to obtain very accurate error bounds at discrete points on the intersection curve.

As mentioned before we are able to resolve straying or looping in the parameter space by the application of a validated ODE solver which adapts its own step-size to verify the existence and uniqueness of the solution obtained. This comes from the fact that the uniqueness criterion is not satisfied at the point where there is a self-intersection or singularity in the pre-image of the intersection curve.

Next, we summarize the results from the various numerical experiments. We also note that increased accuracy will lead to smaller step sizes and hence more number of steps which in turn increases computation time. There is considerable reduction in model space error after we perform the intersection of the model space error bounds especially for the cases of transversal intersection. Various tests we have performed have shown that the accuracy of finding the starting points is of prime importance in the quality of the results. The use of higher order Taylor series is found to have a positive influence on the performance of the ITS method. Of the two validated interval schemes we have discussed, the IHO method is found to be faster than the ITS method.

It was noticed that by following Horner's rule while we input the expression for $[\mathbf{f}(\mathbf{y}(s))]$ we are not only able to reduce the computation time but also reduce the resulting intervals' widths. Further the large amount of data we have obtained could be reduced by the use of an approximation scheme proposed by Tuohy *et al.* [45].

Some limitations of the validated interval scheme are discussed below. This algorithm is costlier compared to a conventional scheme, but the quality of the solution (the guarantee of the true solution lying within the estimated bounds) far outweighs the cost factor. Also unfortunate is the inherent problem of very small but non-zero increase in the width of the interval solutions due to rounding.

9.2 Recommendations for Future Research

Future work on the topic of interval solid modeling could be on how to reduce the width of the *a priori enclosures*, in the parametric spaces by using higher order enclosure methods proposed by Nedialkov *et al* [24].

Another important direction for future research would be to obtain starting points on each branch of the intersection and further to obtain a strict bound on the starting point in each of the parametric spaces. The use of the IPP algorithm in this context is noteworthy, but such an algorithm tends to have problems in the presence of multiple roots. An algorithm which allows for the robust evaluation of multiple zeros of polynomials is another important requirement. This would be valuable for computing starting points for tracing intersection curves [15].

Extension of current intersection methods applied on rational B-spline surfaces to more general and complex surfaces requires further study. Such surfaces include offset, generalized cylinder, blending and medial surfaces, and surfaces arising from the solution of partial differential equations or via recursion techniques.

Application of this algorithm from an industrial perspective also requires more thought. Yet another interesting topic for future research is to apply the validated interval solution scheme to a variety of problems in computational geometry which reduces to solving IVP for ODEs.

Much research remains to be done in bringing such methods to the CAD practice, generalizing the arithmetic to go beyond rational and algebraic numbers (eg. involving transcendental numbers of trigonometric form), and to explore more efficient alternatives that are generally applicable in low and high degree problems alike.

Appendix A

Tables

| Notation | Meaning |
|--|---|
| $[\mathbf{y}(s_0)] = [\mathbf{y}_0]$ | The initial interval vector, or the starting interval vector. |
| $h_j = s_{j+1} - s_j$ | The j^{th} step size. |
| $[\mathbf{y}_j]$ | The interval vector containing the family of solutions to the IVP at s_j . |
| $[\mathbf{y}_{j+1}]$ | The interval containing the family of solutions to the IVP at s_{j+1} . |
| $\mathbf{y}(s; s_0, [\mathbf{y}_0])$ | The ideal family of curves passing through $[\mathbf{y}_0]$ satisfying the ODE system for $s \in [s_0, s_{end}]$. |
| $\mathbf{y}(s; s_j, [\mathbf{y}_j])$ | The ideal family of curves passing through $[\mathbf{y}_j]$ satisfying the ODE system for $s \in [s_j, s_{j+1}]$. |
| $[\tilde{\mathbf{y}}_j]$ | The <i>a priori enclosure</i> which bounds the family of solutions for the step h_j . |
| $\hat{\mathbf{y}}_j$ | A point vector such that $\hat{\mathbf{y}}_j \in [\mathbf{y}_j]$ at s_j . |
| $[\mathbf{y}_j]_i = \mathbf{f}^{[i]}([\mathbf{y}_j])$ | The i^{th} Taylor series coefficient of $\mathbf{f}([\mathbf{y}(s)])$ evaluated at s_j . |
| $\mathbf{f}^{[i]}(\hat{\mathbf{y}}_j)$ | The i^{th} Taylor series coefficient of \mathbf{f} evaluated at $\hat{\mathbf{y}}_j$. |
| $[\tilde{\mathbf{y}}_j]_k = \mathbf{f}^{[k]}([\tilde{\mathbf{y}}_j]) = \mathbf{f}^{[k]}(\mathbf{y}; s_j, s_{j+1})$ | The k^{th} Taylor series coefficient of \mathbf{f} evaluated for some $\mathbf{y}(s)$ at some s such that $s \in [s_j, s_{j+1}]$. |
| $\mathbf{J}(\mathbf{f}^{[i]}; [\mathbf{y}_j], \hat{\mathbf{y}}_j)$ | The Jacobian matrix of $\mathbf{f}^{[i]}$. |

Table A.1: Interval Notations.

Bibliography

- [1] S. L. Abrams, W. Cho, C.-Y. Hu, T. Maekawa, N. M. Patrikalakis, E. C. Sherbrooke, and X. Ye. Efficient and reliable methods for rounded-interval arithmetic. *Computer-Aided Design*, 30(8):657–665, July 1998.
- [2] C. L. Bajaj, C. M. Hoffmann, J. E. Hopcroft, and R. E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5(4):285–307, November 1988.
- [3] R. E. Barnhill and S. N. Kersey. A marching method for parametric surface / surface intersection. *Computer Aided Geometric Design*, 7(1-4):257–280, June 1990.
- [4] G. F. Corliss and R. Rihm. Validating an a priori enclosure using high-order Taylor series. In G. Alefeld, A. Frommer, and B. Lang, editors, *Scientific Computing and Validated Numerics: Proceedings of the International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics - SCAN '95*, pages 228–238. Akademie Verlag, Berlin, 1996.
- [5] P. Eijgenraam. *The Solution of Initial Value Problems Using Interval Arithmetic*. Mathematical Centre Tracts No. 144., Stichting Mathematisch Centrum, Amsterdam, 1981.
- [6] FADBAD/TADIFF, A C++ package for automatic differentiation. <http://www.imm.dtu.dk/fadbad.html/>.
- [7] T. A. Grandine and F. W. Klein. A new approach to the surface intersection problem. *Computer Aided Geometric Design*, 14(2):111–134, 1997.
- [8] E. Hansen, editor. *Topics in Interval Analysis*. Oxford University Press, 1969.
- [9] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1989.
- [10] C. M. Hoffmann. Robustness in geometric computations. *Journal of Computing and Information Science in Engineering*, 1(2):105–204, June 2001.
- [11] C. Y. Hu, T. Maekawa, N. M. Patrikalakis, and X. Ye. Robust interval algorithm for surface intersections. *Computer-Aided Design*, 29(9):617–627, September 1997.

- [12] C. Y. Hu, T. Maekawa, E. C. Sherbrooke, and N. M. Patrikalakis. Robust interval algorithm for curve intersections. *Computer-Aided Design*, 28(6/7):495–506, June/July 1996.
- [13] C. Y. Hu, N. M. Patrikalakis, and X. Ye. Robust interval solid modeling: Part I, Representations. *Computer-Aided Design*, 28(10):807–817, October 1996.
- [14] K. R. Jackson and N. S. Nedialkov. Some recent advances in validated methods for IVPs for ODEs. *Applied Numerical Mathematics*, 42(1-3):269–284, August 2002.
- [15] K. H. Ko, T. Sakkalis, and N. M. Patrikalakis. Nonlinear polynomial systems: Multiple roots and their multiplicities. In F. Giannini and A. Pasko, editors, *Shape Modeling International Conference, SMI 2004*, Genoa, Italy, June 2004. IEEE Computer Society Press, Los Alamitos, CA.
- [16] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley Eastern Edition, New Age International Limited, 1994.
- [17] G. A. Kriezis, N. M. Patrikalakis, and F.-E. Wolter. Topological and differential-equation methods for surface intersections. *Computer-Aided Design*, 24(1):41–55, January 1992.
- [18] F. Kruckeberg. Ordinary differential equations. *Topics in Interval Analysis*, pages 91–97, 1969.
- [19] U. Kulisch, R. J. Lohner, and A. Facius. *Perspectives on Enclosure Methods*. Springer, New York, 2001.
- [20] R. J. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In J.R. Cash and I. Gladwell, editors, *Computational Ordinary Differential Equations*, pages 425–435. Clarendon Press, Oxford, 1992.
- [21] R. J. Lohner. Step size and order control in the verified solution of IVP with ODEs. In *SciCADE'95 International Conference on Scientific Computation and Differential Equations*, Stanford, CA, 1995.
- [22] R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.
- [23] H. Mukundan, K. H. Ko, T. Maekawa, T. Sakkalis, and N. M. Patrikalakis. Tracing surface intersections with a validated ode system solver. In G. Elber and G. Taubin, editors, *Proceedings of the Ninth EG/ACM Symposium on Solid Modeling and Applications*, Genoa, Italy, 2004. Eurographics Press.
- [24] N. S. Nedialkov. *Computing the Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*. PhD thesis, University of Toronto, Toronto, Canada, 1999.

- [25] N. S. Nedialkov and K. R. Jackson. *An Interval Hermite-Obreschkoff Method for Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*. Kluwer, Dordrecht, The Netherlands, 1999.
- [26] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.
- [27] N. M. Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13(1):89–95, January 1993.
- [28] N. M. Patrikalakis and T. Maekawa. Intersection problems. In G. Farin, J. Hoschek, and M. S. Kim, editors, *Handbook of Computer Aided Geometric Design, Chapter 25*, pages 623–650. Elsevier, Amsterdam, July 2002.
- [29] N. M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer-Verlag, Heidelberg, 2002.
- [30] N. M. Patrikalakis, T. Maekawa, K. H. Ko, and H. Mukundan. Surface to surface intersection. In L. Piegl, editor, *International CAD Conference and Exhibition, CAD'04*, Thailand, May 2004.
- [31] N. M. Patrikalakis, T. Maekawa, K. H. Ko, and H. Mukundan. Surface to surface intersection. *Computer-Aided Design and Applications*, 1(1-4):449–458, 2004.
- [32] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1988.
- [33] PROFIL/BIAS, Interval Arithmetic Subroutines. <http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html>.
- [34] T. Sakkalis, G. Shen, and N. M. Patrikalakis. Topological and geometric properties of interval solid models. *Graphical Models*, 63(3):163–175, 2001.
- [35] T. W. Sederberg, H. N. Christiansen, and S. Katz. Improved test for closed loops in surface intersections. *Computer-Aided Design*, 21(8):505–508, October 1989.
- [36] G. Shen and N. M. Patrikalakis. Numerical and geometric properties of interval B-splines. *International Journal of Shape Modeling*, 4(1 and 2):35–62, March and June 1998.
- [37] G. Shen, T. Sakkalis, and N. M. Patrikalakis. Boundary representation model rectification. *Graphical Models*, 63(3):177–195. Also in: *Proceedings of the Sixth ACM Solid Modeling Symposium*. D. Anderson and K. Lee, editors. Ann Arbor, Michigan, June 2001. NY: ACM, 2001.
- [38] G. Shen, T. Sakkalis, and N. M. Patrikalakis. Interval methods for B-Rep model verification and rectification. In *Proceedings of the ASME 26th Design Automation Conference*, pages 140 and CD-ROM, Baltimore, MD, September 2000.

- [39] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of non-linear polynomial systems. *Computer Aided Geometric Design*, 10(5):379–405, October 1993.
- [40] H. Shou, H. Lin, R. Martin, and G. Wang. Modified affine arithmetic is more accurate than centered interval arithmetic or affine arithmetic. In Wilson M. Martin R., editor, *IMA Conference on the Mathematics of Surfaces 2003*, pages 355–365. Springer, September 2003.
- [41] P. Sinha, E. Klassen, and K. K. Wang. Exploiting topological and geometric properties for selective subdivision. In *SCG '85: Proceedings of the first annual symposium on Computational geometry*, pages 39–45. ACM Press, 1985.
- [42] M. R. Spiegel. *Theory and Problems of Advanced Mathematics for Engineers and Scientists*. McGraw-Hill Book Company, Singapore, 1983.
- [43] O. Stauning. *Automatic Validation of Numerical Solutions*. PhD thesis, Technical University of Denmark, Lyngby, Denmark, 1997.
- [44] N. F. Stewart. A heuristic to reduce the wrapping effect in the numerical solution of $x' = f(t, x)$. *BIT*, 11(1):329–337, 1971.
- [45] S. T. Tuohy, T. Maekawa, G. Shen, and N. M. Patrikalakis. Approximation of measured data with interval B-splines. *Computer-Aided Design*, 29(11):791–799, November 1997.
- [46] C. Ullrich. *Computer arithmetic and self-validating numerical methods*. Academic Press, Inc., 1990.
- [47] VNODE, Validated Nonlinear Ordinary Differential Equations solver. www.cas.mcmaster.ca/mediak/Software/VNODE/VNODE.shtml.
- [48] S.-T. Wu and L. N. Andrade. Marching along a regular surface/surface intersection with circular steps. *Computer Aided Geometric Design*, 16(4):249–268, May 1999.
- [49] X. Ye and T. Maekawa. Differential geometry of intersection curves of two surfaces. *Computer Aided Geometric Design*, 16(8):767–788, September 1999.