ERROR BOUNDED APPROXIMATE REPARAMETRIZATION OF NON-UNIFORM RATIONAL B-SPLINE CURVES

by

Mark D. Bloomenthal

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

The University of Utah

August 1999

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Mark D. Bloomenthal

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

Chair: Elaine Cohen

Frank Stenger

Peter Shirley

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Mark D. Bloomenthal in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Elaine Cohen
Chair, Supervisory Committee

Approved for the Major Department

Robert R. Kessler
Chair/Dean

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

In designing free-form curves and surfaces, the user of a geometric modeler generally wishes to concentrate on shape. Geometric constructions using parametrically defined representations, however, often lead to unwanted or surprising effects due to curve and surface parametrizations. Reparametrizing curve or surface representations is an important approach to minimizing these effects. Reparametrization also can be viewed as a means for transforming curves and surfaces into representations that are better suited to establishing important properties or exploring the design space.

This research explores the use of curve reparametrization in modeling tools for NURBS (Non-Uniform Rational B-Spline) based CAD systems. There are many practical applications of such tools including establishing and exploring correspondence in geometry, creating related speed profiles along motion curves for animation, specifying speeds along tool paths, and identifying geometrically equivalent, or nearly equivalent, curve mappings.

Central to this work is the presentation of new error bounded algorithms for important cases of NURBS curve reparametrization. These algorithms are developed from within a general framework and address:

- approximate arc length parametrizations of curves,
- approximate inverses of NURBS functions, and
- reparametrizations that establish user specified tolerances as bounds on the Frechet distance between parametric curves.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Elaine Cohen, Frank Stenger, and Peter Shirley, for their invaluable help. Special thanks to the chair of my committee, Elaine Cohen, for her insight and continued insistence on the high level view. Thanks to Rich Riesenfeld and Elaine Cohen for giving me the time to finish this work; to the staff members of the Alpha1 research group, Russ Fish, David Johnson, and Tom Thompson, for their ideas and for maintaining the software environment; to staff members and graduate students, past and present, who have been part of the Alpha1 research group. Thanks also to the graduate studies committee for keeping the pressure on; to Bruce Gooch and Richard Coffey for their help with my defense; to Colleen Hoopes for her kind and continued help with the paper work. Thanks to Jules Bloomenthal for proofreading and words of wisdom.

CHAPTER 1

INTRODUCTION

1.1 Context

Many geometric modeling systems use parametric functions to represent free-form curves and surfaces. Formally these functions are maps from a portion of the real line, or real plane, into $\boldsymbol{R}^2$ or $\boldsymbol{R}^3$.

In designing and modifying curves and surfaces, users of geometric modeling systems generally are concerned with shape, not function maps. Unfortunately, geometric constructions based on parametric representations often lead to unwanted or surprising artifacts due to the particular mappings used in representing given curves or surfaces.

The parametrization of a particular curve or surface is often hidden from the user, but nonetheless affects the specification and outcome of operations the user applies. A correspondence between points on different curves used in a geometric construction, for example, may seem intuitive or obvious to the user, but that correspondence might not be reflected in the underlying representations nor in the end result of the construction.

Operations internal to the modeling system may suffer parametrization artifacts as well. It may be important for a modeling system to recognize that two curve or surface mappings actually represent the same locus of points. Such a matching, though obvious to human eyes, might not be simple to establish automatically. Grid and sample generation are other examples. Modeling systems are often required to sample data on curve and surface maps. Such sampling is usually affected by curve and surface parametrization.

There are a number of possible techniques for avoiding or minimizing parametrization artifacts including:

- Indirectly affecting the parametrization of a curve or surface by using a different modeling technique for its construction.

- The use of intrinsic properties of curves and surfaces such as curve arc length or curve and surface curvatures. Problems can then be formulated in terms of these intrinsic properties rather than in terms of parameter values.

- Inversion of the curve or surface mappings in an attempt to formulate problems directly in terms of spatial properties rather than parametric values.

- The use of nonparametric forms such as implicit or algebraic surfaces.

- Reparametrizing curves or surfaces by applying change of parameter functions to the parametric maps used to represent them.

The last approach is a particularly important one for treating parametrization related problems in modelers using parametric representations. This thesis focuses on NURBS curve reparametrization and its integration into high level tools for design from within the context of NURBS based CAD systems.

1.2 Reparametrization in CAD Design

Within the context of a CAD system, parametrization artifacts can be viewed as shortcomings of the representation, giving rise to problems needing solution. Figure 1.1 depicts the generation of a ruled surface from two curves. The result is degenerate due to an improper parametric correspondence between the curves. The nonuniqueness of representations gives rise to other examples. It may be difficult for a CAD system to recognize that two different curve representations give rise to the same, or nearly the same, geometry. This may present difficulties when constructing valid boundary representations for solids in a CAD system. Such cases arise when different surfaces agree **geometrically** along their edges but have unlike parametrization (see Figure 1.2).

Reparametrization operations can be viewed from other perspectives, however. These operations can transform curve and surface representations from one space into another space better suited for establishing or determining properties of interest. The typical situation in a CAD environment occurs when a specific representation is well suited for the expression of some aspect of shape, but not well suited for establishing some other property of importance to the user, such as correspondence or uniform spacing of features (see Figure 1.3).

Reparametrization operators may also be thought of as opportunities to explore the design space. Different correspondences between cross sections of an extruded or swept surface, for example, give rise to different surface shapes generated from the same curve

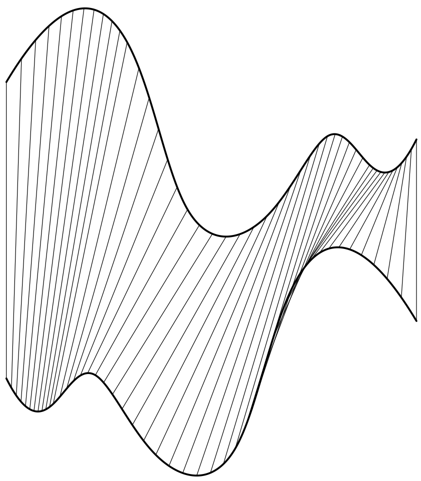


Figure 1.1. Ruled surface construction using curves with poor parametric correspondence. The resulting planar surface has a degenerate region.

geometry (see Figure 1.4). The interplay between curve shape, correspondence of curves, and resulting surface shape may be quite subtle in examples of this nature.

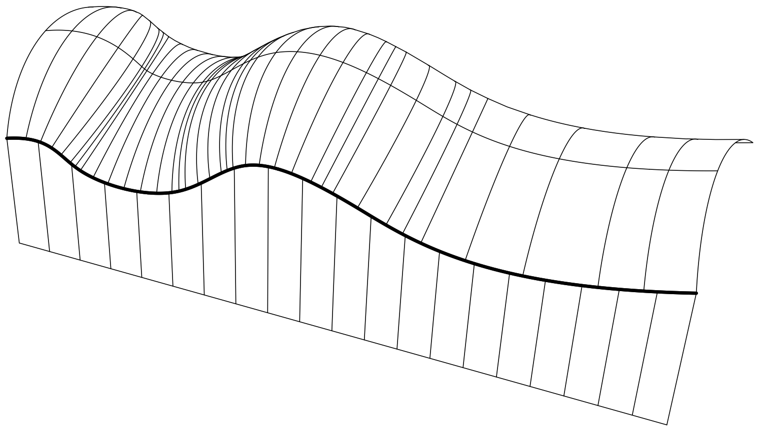
1.3 Problem Statement

Operations for exploring the space of reparametrizations are often lacking or limited in CAD systems, even though there are many important applications that can benefit from such operations. In order to be useful these operations must be integrated into the CAD environment as practical design tools.

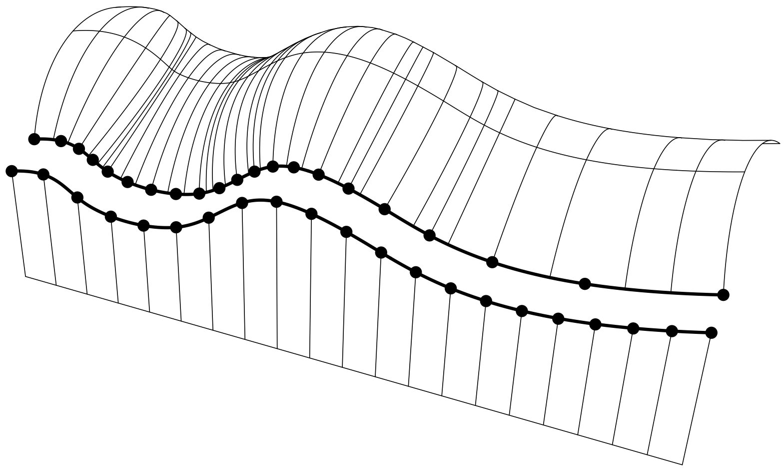
This research will focus on the reparametrization of NURBS curves from within the context of a NURBS based CAD system. Specifically this thesis will focus on reparametrizations of the form $c(r(t))$ where c is a NURBS curve and r may or may not be a NURBS curve.

There are many practical applications of this general problem including: establishing and exploring correspondences in geometry, creating related speed profiles along motion curves for animation, specifying speeds along tool paths, and identifying geometrically equivalent, or nearly equivalent, curve mappings.

In these applications practical issues arise regarding the integration of reparametrization operations into the CAD system as usable design tools. Operations need to be specified in ways that make sense to the user and not as the specification of obtuse formulae. Results must be predictable, take a usable form, and be capable of control by the user.



(a)



(b)

Figure 1.2. It may be difficult for a CAD system to recognize that different surfaces agree geometrically along their edges. (a) Two surfaces meeting at edges with nearly identical geometry but very different parametrizations. (b) Dots indicate equal steps in the parametric domains of the edge curves.

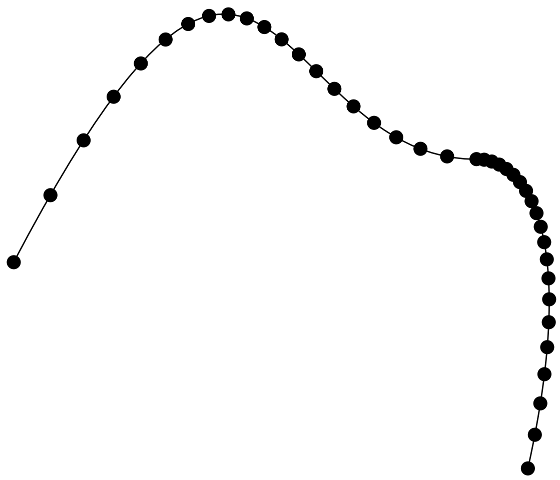


Figure 1.3. Free form curve representations, though well suited to express shape, are often not well suited for establishing other properties such as uniform geometric spacing. Here equal steps in the parametric domain of a curve result in the **not** very equally spaced points shown. For most representations, equal spacing along curves must be approximated numerically.

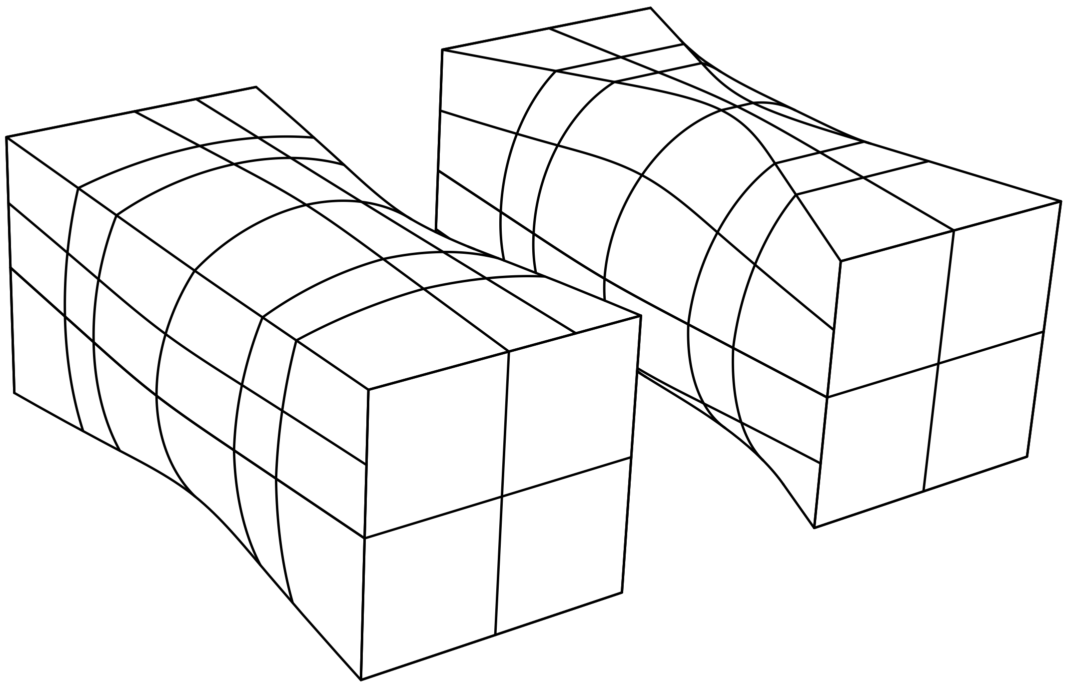


Figure 1.4. Different sweep surfaces generated by blending square and circular cross section curves. Left: sweep surface generated from the original curves. Right: sweep surface generated after the cross section curves have been reparametrized in order to create a different correspondence between them.

Problems also arise in the representation of the results of such reparametrizations. These problems include:

closure - It is important to represent the results of operations in NURBS form in a NURBS based modeling system so that operations in the system may be composed. Reparametrizations of NURBS curves, however, are not in general closed under the NURBS representation (c.f. section 4.2). This gives rise to the need to approximate results in NURBS spaces. These spaces of approximation are initially unknown.

error bounds - Given that approximation may be required, the user should be able to control the accuracy of the approximation. These controls should ideally take the form of meaningful tolerances that are maintained by the operations.

data complexity - Reparametrization involves (the approximation of) function composition. Many NURBS schemes to do this raise the degree of the polynomials used in the approximations. Increasing the accuracy of these approximations may also increase the number of polynomial pieces in the result. It is important to consider ways to manage this increased data complexity.

1.4 Research Objectives

This research addresses issues of the usefulness of NURBS curve reparametrization in tools for CAD and of the incorporation of such tools into a CAD environment. Rather than attempting an exhaustive survey, this work presents representative reparametrization operations and applications for CAD. These operations are incorporated into a set of CAD modeling tools that help solve interesting problems in design. Approaches that deal with approximation and data complexity issues in results are also developed.

Central to this work is the presentation of new error bounded algorithms for important cases of NURBS curve reparametrization. These algorithms are developed from within a general framework and address:

- approximate arc length parametrizations of curves,
- approximate inverses of NURBS functions, and
- reparametrizations that establish user specified tolerances as bounds on the Frechet distance between parametric curves.

Reparametrizations of this nature usually can only be approximated by NURBS curves. The algorithms developed here yield results as NURBS approximations either to the composed curve $c(r)$ or to the reparametrization function r. Users can specify tolerances for these approximations which the algorithms automatically maintain.

1.5 Document Summary

Chapter 2 presents some of the mathematical background necessary for the development of this thesis. This chapter concentrates on: distance metrics between parametric mappings, general definitions of parametric curves, and the NURBS representation for parametric curves. Chapter 3 is a summary of relevant work in the literature related to this thesis. Chapter 4 develops new error bounded algorithms for NURBS curve reparametrizations. These algorithms are developed from within a general framework. The algorithms are extended in Chapter 5, while Chapter 6 gives examples of their use. Chapter 7 gives examples of higher level applications and tools incorporating the algorithms of Chapters 4 and 5.

CHAPTER 2

MATHEMATICAL BACKGROUND

This chapter presents mathematical preliminaries that will be used throughout the rest of this thesis. Section 2.1 gives formal definitions for the notion of parametric curve. Section 2.2 gives the definition of the NURBS representation for parametric curves along with important properties of this representation.

2.1 What is a Parametric Curve?

The notion of a curve is defined differently for different purposes and by different authors. Individual functions that map from $\boldsymbol{R}$ into $\boldsymbol{R}^n$ are often referred to as parametric curves. Parametric curves also are often defined as *equivalence classes* of such mappings. In this view, individual mappings of an equivalence class are referred to as different *parametrizations* of the same curve. This latter view is adopted here.

Two equivalence class definitions for parametric curves are given below. The definition of *Frechet curve* is based on the distance between parametric mappings. The concepts of distance used in this definition are important in their own right and will be used for the development of CAD applications for reparametrization (c.f. section 4.4). The *regular curves* of section 2.1.3 are a specialization of the Frechet curves in that all regular curves are also Frechet curves. The definition of regular curve places restrictions on the types of mapping functions under consideration. These restrictions have practical implications for the development of algorithms. Section 2.1.4 discusses a useful extension to regular curves.

2.1.1 Distance Between Parametric Maps

In this section we give two useful formulations for the distance between parametric mappings. The equal-parameter distance is a commonly used metric which depends on the parameterization of the curve mappings. The notion of Frechet distance between curve mappings is **independent** of the mappings' parameterizations.

Given two mappings, $\alpha : I \to \boldsymbol{R}^n$ and $\beta : I \to \boldsymbol{R}^n$, the *equal-parameter distance* between these mappings is defined as:

$$d(\alpha, \beta) = \sup_t \|\alpha(t) - \beta(t)\|$$

for $t \in I$ and $\| \ \|$ Euclidean distance in $\boldsymbol{R}^n$. Section 2.2.3 discusses methods for bounding $d(\alpha, \beta)$ by a NURBS function when both α and β are NURBS curves.

Given two mappings, $\alpha : I_\alpha \to \boldsymbol{R}^n$ and $\beta : I_\beta \to \boldsymbol{R}^n$, the *Frechet distance* between these mappings is defined as:

$$\mathcal{F}(\alpha, \beta) = \inf_h \sup_t \|\alpha(t) - \beta(h(t))\|$$

where,

- $t \in I_\alpha$ and
- $h : I_\alpha \to I_\beta$ is a homeomorphism.

The Frechet distance is essentially the minimum equal-parameter distance between α and possible reparametrizations of β.

It is easily shown that $\mathcal{F}(\alpha, \beta)$ is a pseudo-metric satisfying:

1. $\mathcal{F}(a, b) = \mathcal{F}(b, a)$
2. $\mathcal{F}(a, b) \geq 0$
3. $\mathcal{F}(a, c) \leq \mathcal{F}(a, b) + \mathcal{F}(b, c)$
4. $a = b \Rightarrow \mathcal{F}(a, b) = 0$

but failing to satisfy: $\mathcal{F}(a, b) = 0 \Rightarrow a = b$.

2.1.2 Frechet Curves

The definition of Frechet curves is based on the Frechet distance between mappings.

For Γ, a class of curve mappings, and some $c, d \in \Gamma$ the relation defined by:

$$c \overset{\mathcal{F}}{\mapsto} d \equiv \mathcal{F}(c, d) = 0$$

is an equivalence relation.

In accordance with [26] we define a *Frechet curve* as follows:

Given $c \in \Gamma$, the curve C specified by c is the equivalence class:

$$C = \{f \in \Gamma : \mathcal{F}(f, c) = 0\}$$

A mapping contained in the equivalence class of a curve is said to be a *parametrization* of that curve.

The Frechet distance between curves is defined as follows: let C and D be two curves and let $c \in C$ and $d \in D$. Then

$$\mathcal{F}(C, D) = \mathcal{F}(c, d).$$

Note that the Frechet distance between curves is a true metric satisfying $\mathcal{F}(C, D) = 0 \Leftrightarrow C = D$

2.1.3 Regular Curves

Another, perhaps more familiar, definition of curve follows from the notion of a regular parametric mapping [47].

A mapping, $\alpha(u) : I_u \to \boldsymbol{R}^n$, is said to be *regular* if:

- $\alpha(u)$ is C^1 on I_u, and
- $\frac{d\alpha(u)}{du} \neq 0, \ \forall_{u \in I_u}$

A real valued function $\theta(t) : I_t \to I_u$ is called an *allowable change of parameter* if:

- $\theta(t)$ is C^1 on I_t, and
- $\frac{d\theta}{dt} \neq 0, \ \forall_{t \in I_t}$

It is straightforward to show that $\alpha(\theta(t))$ is regular for α regular and θ an allowable change of parameter.

Given two regular curve mappings $\alpha(u) : I_u \to \boldsymbol{R}^n$ and $\beta(t) : I_t \to \boldsymbol{R}^n$, define the relation: $\alpha \overset{R}{\mapsto} \beta$ to mean that there exists an allowable change of parameter $\theta(t) : I_t \to I_u$ such that $\theta(I_t) = I_u$ and $\beta(t) = \alpha(\theta(t)) \ \forall_{t \in I_t}$.

The relation $\overset{R}{\mapsto}$ is an equivalence relation and a *regular curve* is defined to be an equivalence class of regular curve mappings under $\overset{R}{\mapsto}$. As with Frechet curves, a mapping contained in the equivalence class of a regular curve is said to be a parametrization of that curve. Note that the definition of regular curve is a specialization of the definition of Frechet curve.

Some important properties of regular curves are [47, 54]:

- A regular curve mapping remains regular under an allowable change of parameter.
- All parametrizations of a regular curve are locally invertible.
- Regular curves are rectifiable over compact intervals of a given parametrization and hence have an arc length parametrization.
- Regular curves have well-defined tangent lines at all points.

2.1.4 Piecewise Regular Curves

Limiting consideration only to regular curves is too restrictive for CAD applications where intentional unit tangent and parametric derivative discontinuities are sometimes desired.

A mapping, $\alpha(u) : I_u \rightarrow \boldsymbol{R}^n$, is *piecewise regular* if:

- $\alpha(u)$ is C^1 on I_u except at a finite number of points on I_u where it is constrained to be only C^0, and
- $\frac{d\alpha(u)}{du} \neq 0,\ \forall_{u \in I_u}$. (At points on I_u where $\alpha(u)$ is only C^0, left and right derivatives must exist and be different from zero.)

A real valued function $\theta(t) : I_t \rightarrow I_u$ is a *piecewise allowable change of parameter* if:

- $\theta(t)$ is C^1 on I_t except at a finite number of points on I_t where it is constrained to be only C^0, and
- $\theta(t)$ is strictly monotonic on I_t

Piecewise regular curves are defined in a manner analogous to regular curves and have properties similar to properties of regular curves. There may be, however, a finite number of points in the domain of a piecewise regular curve where tangent lines are **not** well defined.

Throughout this work it is assumed that all curves are piecewise regular and that all change of parameter functions are piecewise allowable as defined above.

2.2 NURBS Curves

NURBS curve and surface formulations have become fairly standard representation schemes in current CAD systems. This section presents the NURBS curve definition along with some of its important properties.

A univariate B-spline is a series of parametrically defined polynomial pieces joined end to end, with certain smoothness (continuity) constraints at the places where the polynomial pieces are joined. Rational pieces must be used to allow exact representations for circles and circular arcs [66]. The term **NURBS** (Non-Uniform Rational B-Spline) is used to encompass both the polynomial and rational cases.

NURBS are expressed using a particular set of basis functions for the piecewise polynomials known as the B-spline basis. There are many advantages to the use of this basis, including computational stability and geometric locality (see [35] and [19]).

A NURBS curve mapping in three dimensional space, of order m over the knot vector $\boldsymbol{\tau}$, is denoted as:

$$\gamma(t) = \frac{\sum_i w_i \mathbf{E}_i B_{i,m,\boldsymbol{\tau}}(t)}{\sum_i w_i B_{i,m,\boldsymbol{\tau}}(t)} \tag{2.1}$$

where the knot vector $\boldsymbol{\tau}$ is a nondecreasing sequence of at least $2m$ real values, the $\mathbf{E}_i \in \boldsymbol{R}^3$, the w_i are positive real numbers, and $\{B_{i,m,\boldsymbol{\tau}}(t)\}_i$ are the B-spline basis functions of order m over knot vector $\boldsymbol{\tau}$.

We can consider this rational form as the projection to $\boldsymbol{R}^3$ of a polynomial B-spline in $\boldsymbol{R}^4$ with control points: $\{w_i(\mathbf{E}_i, 1)\}_i$.

It is common practice to refer to the NURBS form above as a NURBS *curve*. We will use this term to mean a NURBS representation for a *curve mapping*.

The curve mapping above is a convex combination of the points $\mathbf{E}_i$ for all values of t. Hence, for a given interval the curve lies in the convex hull of the points actually blended (i.e., that have associated nonzero basis functions) on that interval. This *convex hull property* for NURBS curves is often used to localize a NURBS curve quickly or for a fast check for possible intersections of NURBS curves. This property can also be used to establish both upper and lower bounds for scalar valued NURBS curves.

An order m and knot vector $\boldsymbol{\tau}$ together define a linear space of B-splines denoted here by $\mathcal{S}_{m,\tau}$ (see [19]).

2.2.1 NURBS Curve Refinement

Given any NURBS curve defined with a particular knot vector, $\boldsymbol{\tau}$, the same mapping can be represented using a different knot vector, $\boldsymbol{\tau}'$, if $\boldsymbol{\tau}'$ is a refined partition of $\boldsymbol{\tau}$. The process by which a NURBS curve is represented over a refined knot vector is known as *B-spline refinement*.

B-spline refinement has the very important property of convergence of the curve's control polygon to the curve. As more values are added to a knot vector and a curve refined, the control polygon for the curve approaches the actual curve in the parametric area where the new knot vector values are added. By refining a curve over its entire parametric domain, the control polygon can be made arbitrarily close to the curve everywhere [14].

There exist very efficient and elegant algorithms for the refinement of NURBS curves [12]. Points on NURBS curves can be evaluated by using special cases of these algorithms.

2.2.2 Algebraic Operations on NURBS

The univariate NURBS representation forms an algebra in the following sense. If $\mathcal{N}$ is the class of all univariate NURBS functions defined on an interval I_t, and if $f(t), g(t) \in \mathcal{N}$ and $\lambda \in \boldsymbol{R}$ then:

$$f(t) + g(t) \in \mathcal{N}, \;\; \lambda f(t) \in \mathcal{N}, \;\; \text{and } f(t)g(t) \in \mathcal{N}.$$

References [23] and [55] discuss the NURBS representations for addition and multiplication of NURBS functions. Here we provide a brief summary of important characteristics of these representations.

For $f(t) = \sum_i P_i B_i(t)$ we have $\lambda f(t) = \sum_i \lambda P_i B_i(t)$.

For addition of polynomial splines, we assume that $f(t)$ and $g(t)$ are NURBS functions defined over a common interval I_t. Without loss of generality, we assume also that $f(t)$ and $g(t)$ have the same order and are defined over the same set of B-spline basis functions. This can always be accomplished by using B-spline degree raising [13], positive affine transformations of knot vectors, and B-spline refinement [12].

If $f(t) = \sum_i P_i B_i(t)$ and $g = \sum_i Q_i B_i(t)$ then

$$f(t) + g(t) = \sum_i (P_i + Q_i) B_i(t).$$

The addition of rational functions requires the use of NURBS function multiplication since for $f = \frac{p_1(t)}{q_1(t)}$ and $g = \frac{p_2(t)}{q_2(t)}$

$$f(t) + g(t) = \frac{p_1(t)q_2(t) + p_2(t)q_1(t)}{q_1(t)q_2(t)}.$$

Determining a NURBS representation for the product $f(t)g(t)$ can be accomplished in several ways; e.g., using spline or Bezier function multiplication, or by solving a NURBS interpolation problem (see [29, 55, 23]).

The NURBS representation is also closed under the derivative operator. The derivative of a polynomial NURBS curve of order m is itself a polynomial NURBS curve of order $m-1$. For derivatives of rational NURBS curves we must use the quotient rule which tends to raise the order of the result. The derivative of a rational NURBS curve $f(t)/g(t)$, where f is order m and g is order n, is a rational NURBS curve $q(t)/r(t)$ where q is order $m+n-2$ and r is order $2n-1$.

2.2.3 An Example: Bounding Equal-Parameter Distance

In this section we illustrate the use of NURBS curve properties by developing bounds on the equal-parameter distance between NURBS curves (see section 2.1.1). The techniques developed here are used by the reparametrization algorithms of Chapters 4 and 5.

Given two NURBS curves with a common parametric domain $c_1(t), c_2(t) : I_t \to \boldsymbol{R}^m$ we seek to bound:

$$\|c_1(t) - c_2(t)\|$$

over subintervals of I_t where $\|\ \|$ denotes Euclidean distance.

Without loss of generality assume that c_1 and c_2 are of the same polynomial order and defined over the same knot vector (see section 2.2.2). The difference $v(t) = c_1(t) - c_2(t)$ can then be formed as a NURBS curve reducing the problem to bounding $\|v(t)\|$ for v a NURBS curve. Note that $\|v(t)\|$ is not in general a NURBS function given $v(t)$ a NURBS.

Let:

- $\{t_j\}_{j=0,\ldots,n}$ be the **distinct** values in the knot vector for $v(t)$,
- $\text{index}(t) = j : t \in I_j = [t_j, t_{j+1}),\ \text{ for } t \in I_t$,
- $\{v_{j_k}\}_k$ be the B-spline (vector) coefficients of $v(t)$ that are blended on I_j (i.e., for which the corresponding basis functions have positive support on I_j) for $j = 0, \ldots, n-1$, and

- $\mathcal{C}_j$ be the convex hull of the $\{v_{j_k}\}_k$.

Since $v(t)$ must lie in convex hull $\mathcal{C}_j$ on interval I_j, the function $\|v(t)\|$ can be no greater on I_j than the distance from the origin to the furthest point in $\mathcal{C}_j$. This maximum distance will occur at a vertex of $\mathcal{C}_j$ and hence for $U_j = {}^{\max}_{k} \|v_{j_k}\|$ we have $\|v(t)\| \leq U_j, \ \forall_{t \in I_j}$ for $j = 0, ..., n-1$.

Similarly $\|v(t)\|$ can be no less on I_j than the distance from the origin to the **nearest** point in convex hull $\mathcal{C}_j$. If the origin is contained in $\mathcal{C}_j$ then we can state only the obvious bound of zero. Otherwise the minimum distance from the origin to a point in $\mathcal{C}_j$ occurs either at a vertex, edge, or side of $\mathcal{C}_j$. This minimum distance can be computed efficiently using methods discussed in [33], [46], and [9]. Bounding box techniques also can be used to bound this minimum from below (see [39] pages 514-525).

Denote by L_j the minimum distance to $\mathcal{C}_j$ for $j = 0, ..., n-1$. Then,

$$U(t) = U_{\text{index}(t)}$$
$$L(t) = L_{\text{index}(t)}$$

are piecewise constant functions on I_t which bound $\|v(t)\|$ from above and below respectively.

To improve these bounds $v(t)$ can be refined and $\|v(t)\|$ bound on the finer partition of I_t represented by the distinct knots of the refined knot vector for $v(t)$.

The above development also holds for the case $c_1(t), c_2(t) : I_t \to \boldsymbol{R}$.

It was noted above that $\|v(t)\|$ is not, in general, a NURBS function given $v(t)$ a NURBS. However $\|v(t)\|^2 = < v(t), \ v(t) >$ **is** a NURBS whenever $v(t)$ is a NURBS since it is the product of two NURBS functions. Another approach to bounding $\|v(t)\|$ over intervals of its domain, therefore, is to establish $L(t) \leq \|v(t)\|^2 \leq U(t)$ for positive piecewise constant $L(t)$ and $U(t)$, from which $\sqrt{L(t)} \leq \|v(t)\| \leq \sqrt{U(t)}$ follows. The advantage of this technique is that bounds are always established over scalar valued functions. This obviates the need to compute, or bound, the minimum distance to convex hulls for vector valued $v(t)$ when lower bounds are sought. The disadvantage of this technique is that the dot product function in NURBS form can be relatively expensive to compute.

2.2.4 Composition of NURBS Curves

Consider mapping $c(u) : I_u \to \boldsymbol{R}^n$ and piecewise allowable change of parameter $r(t) : I_t \to I_u$ with $r(I_t) = I_u$. If both c and r are polynomial on their respective domains, then it is obvious that $c(r(t))$ is polynomial on I_t. Similarly, if both c and r are NURBS, it can be shown that $c(r(t))$ is a NURBS.

Consider the case where r is a polynomial spline, and c is either a polynomial or rational spline. If r is order l and c is order m, then $c(r(t))$ will be order $(l-1)(m-1)+1$. If r has a knot at t_i of multiplicity p, then $c(r(t))$ is **constrained** to be only C^{l-p-1} at t_i (it actually may have higher order continuity there however). If c has a knot at u_i of multiplicity q, then $c(r(t))$ is constrained to be only C^{m-q-1} at $r^{-1}(u_i)$.

From these considerations, we see that $c(r(t))$ is a polynomial or rational spline in the spline space $\mathcal{S}_{o=(l-1)(m-1)+1,\sigma}$ where knot vector σ is given by the algorithm of Appendix A.

Composition with linear rational functions merits special mention because of the low order and smoothness of such functions [32] (see also Appendix B). Lee and Lucian [43] give an explicit formulation for the composition $c(r(t))$ where c is NURBS and r is a linear rational function.

For further information see the introduction to NURBS function composition in [58].

CHAPTER 3

RELATED WORK

This thesis is concerned with the appropriateness of curve parametrizations to establish desirable properties. The need to change the parametrization or sampling of NURBS curves to establish specific properties has certainly been recognized in the literature. Of particular importance in this work are arc length parametrizations, to establish speed or equal spacing properties along a curve, and geometric correspondences established by the relative parametrizations of curves. There is related work in the literature to both of these general problems.

This chapter summarizes the previous work most directly related to the reparametrization algorithms presented in this thesis.

3.1 Arc Length Parametrized Curves

Because of the importance of the problem there has been significant work related to approximating arc length functions for parametric curves and sampling parametric curves at equal spacings in arc length.

Farouki and Sakkalis show in [30] that it is impossible to parametrize any real curve, other than a piecewise straight line, by rational functions of its arc length. A sketch of a proof of this fact is given also in [37] for quadratic and cubic B-spline curves. There are a number of methods, however, to extract arc length approximations for parametric curves at discrete points in the curve domain. It is easily shown that lengths of the sides of the control polygon of a NURBS curve can be used to form an upper bound on the arc length of the curve at discrete points where the polygon interpolates the curve. This can be used in combination with NURBS curve refinement and **inscribed** polygons (from NURBS curve point evaluations) to form both upper and lower bounds to the arc length function at discrete points on the curve.

Fritsch and Nielson [31] form a piecewise linear approximation, $r(t)$, to the inverse arc length function generated from a specified number of evaluations equally spaced in the domain of a parametric curve $c(u)$. They then perform discrete evaluations of $c(r(t))$ (i.e., a curve form for $c(r(t))$ is **not** computed) to produce approximately equally spaced points on the curve. The authors use this technique in the context of the computation of a metric for curve/curve comparison. In [18] and [7] similar techniques are used to approximate the arc length and inverse arc length functions for curves in the context of sweep surface construction.

Numerical integration techniques can be used to estimate arc lengths at discrete values in the curve domain. In [62] Newton-Raphson iteration in combination with Romberg quadrature is used to approximate discrete evaluation of the inverse arc length function for parametric curves. No error bounds are given though the authors state that a relative error test is done when applying Romberg integration (although the authors do not state what this test is). In [36] adaptive Guassian quadrature is used to create a lookup table of approximate arc length function values over the domain of a parametric curve. Newton-Raphson iteration in combination with nonadaptive Guassian quadrature is then used to approximate discrete evaluation of the inverse arc length function for the curve. No true error bounds are given for their technique though the user may control relative error by the specification of a tolerance value used to control the adaptive quadrature.

Some approaches try to make the parametrization of interpolants closer to an arc length parametrization by enforcing unit derivative lengths at discrete points in the domain. In [70] a global G^2 parametric interpolation scheme is used which enforces unit derivative length at data interpolation points. This technique uses an iterative algorithm to minimize an energy functional subject to the interpolation and unit derivative constraints. The context for this technique is the interpolation of discrete data points rather than the reparametrization of curves.

Wang and Yang [67] present a technique to interpolate discrete data points using quintic splines. Their application is the conversion of parametric curves to a more nearly arc length form. Data from the original curve are sampled. A piecewise scheme is then used to form a C^2 cubic curve that interpolates sampled points at the ends of each parametric segment. Finally a quintic spline is created that interpolates position, first derivative, and second derivative values of the cubic curve at segment end points with the first derivatives set to unit length. The length of each segment's parametric interval is adjusted so that a unit length derivative also results at the midpoint of the interval. An iterative scheme is used to solve this last, nonlinear, problem. This interpolation scheme typically alters the shape of the original curve; however Wang and Yang suggest that careful sampling of the original curve can minimize alterations to its shape (they assume that total curvature between samples is small and of comparable size). No attempt is made to establish error bounds either in terms of arc length parametrization or deviation from the shape of the original sampled curve. More recently Srinivasan and Ge [65] have extended these results to rational curves representing both translational and rotational motions.

In [38] discrete data points for rational spline motion are interpolated using a local cubic C^1 scheme. The context for this interpolation is real time control of industrial robots. A procedural reparametrization is applied to the interpolant in order to achieve a desired speed profile; which is assumed constant on each segment between interpolation points, with linear changes of speed between such points. This reparametrization is accomplished using an algorithm that tracks the motion curve and updates the robot controls after constant time steps. The motion is either sped up or slowed down based on local estimates of the current speed. The parameter value on the curve interpolant, giving the position at the next time interval, is estimated from current position and desired speed using a small number of iterations of Regula falsi. This technique is concerned with real-time motion control and not with the actual expression of the reparametrized trajectory as a parametric curve.

Elber [24] presents error bounded algorithms to approximate both normalized univariate vector fields and arc length functions for NURBS curves. Given a NURBS vector field $V(t)$ the dot product $f(t) = < V(t), V(t) >$ is computed as a NURBS curve. An approximation $m(t) \approx 1/\sqrt{f(t)}$ is then iteratively computed from samples on $f(t)$. The dot product $< m(t)f(t), m(t)f(t) >$ is formed in each iteration as a NURBS curve and its coefficients are checked for variation from unity. After the algorithm terminates, the NURBS function $m(t)V(t)$ represents an approximate normalized vector field. A similar algorithm is used to compute an approximation for the arc length function of a NURBS curve. Given NURBS curve $c(t)$, the function $f(t) = < c'(t), c'(t) > = \|c'(t)\|^2$ is computed as a NURBS curve. The function $\|c'(t)\| = \sqrt{f(t)}$ is then approximated as a NURBS curve $m(t)$ from samples on $f(t)$ in a manner similar to the algorithm for normalized vector field approximation. The NURBS function $s(t) = \int m(t)$ is then computed in closed form producing a NURBS approximation for the arc length function on $c(t)$. The results of both of these algorithms are bounded by a user specified tolerance on the variation of speed from unity. A NURBS approximation to inverse arc length is not computed and no algorithm for reparametrization by arc length is given. Instead root finding techniques are used on $s(t)$ for discrete point approximation of the inverse arc length function.

Blanc and Schlick [6] use quadratic rationals as change of parameter functions to improve the parametrization of NURBS curves. In particular they consider the use of these functions to improve the parametrization of quadratic NURBS representations of circular arcs. By exploiting the symmetry of the standard Bezier or NURBS formulation for arcs, and by preserving the parametric domain $[0, 1]$ of this formulation, the quadratic rational change of parameter functions are reduced to a one parameter family. The composition of a quadratic NURBS arc with such a change of parameter is then a quartic NURBS curve. They give a definition for chordal deviation of the standard parametrization of circular arcs from a constant speed parametrization. A heuristic is used to reduce the maximum value of this deviation. The reparametrization the authors use for circular arcs depends on the particulars of the behavior of the parametrization of the standard representation for these arcs. The authors do not give a generalization of their technique to approximate arc length parametrizations for more general curves.

The thrust in [28] is the reparametrization of polynomial curve segments to create nearly constant speed curves. The author adopts a "measure of closeness" to constant speed, and then analytically minimizes this measure subject to the use of a linear rational (Mobius) reparametrization. This accomplishes an adjustment in the parametrization of the curve that does not increase its order nor change its domain. On a single parametric span, this technique is limited in the adjustment of speed that can be made; one part of the curve is sped up, and one part slowed down. The author states that extending the specific minimization technique used to the reparametrization of rational curves is difficult.

Casciola and Morigi [10] create approximate arc length parametrized NURBS curves by using linear rational and C^1 piecewise linear rational change of parameter functions. On each parametric interval of the NURBS curve, they approximate the arc length function by a linear rational function. An iterative optimization technique is used to produce an error from the actual arc length function that equi-oscillates. The inverse of the linear rational arc length function approximation is then composed with the NURBS curve to effect the reparametrization.

An adaptive piecewise scheme is used for the reparametrization of the curve. They state two evaluation criteria for this adaptive scheme: q the maximum deviation from unit speed over the curve, and p the ratio of the maximum to minimum arc lengths for curve segments in a regular partition of the domain of the approximate inverse arc length function. If on a parametric interval the criterion chosen is not met, the interval is split and the approximate inverse arc length function recomputed over the subdivided interval.

The optimization method they use to create an arc length approximation over a given parametric interval assumes detailed knowledge of the error function relative to the actual arc length function over that interval. The evaluation criterion p also requires accurate knowledge of the arc length function whereas evaluation criterion q requires accurate knowledge of the speed function on the curve. The authors do not state how the arc length or speed functions are approximated in these contexts and what properties these approximations have. True error bounds are not given for these algorithms. Error results are shown, however, presumably compiled from fine sampling of the curve to produce accurate approximate arc length and finely sampled speed functions.

3.2 Distance Between Parametric Curves

3.2.1 Frechet Distance

This thesis presents algorithms to bound the Frechet distance between curves as defined in sections 2.1.1 and 2.1.2. This definition is consistent with the reparametrization of one curve to match the other in equal-parameter distance.

Given two curve mappings $c_1(u) : I_u = [u_s, u_e] \to \boldsymbol{R}^n$ and $c_2(v) : I_v = [v_s, v_e] \to \boldsymbol{R}^n$ the surface $s : I_u \times I_v \to \boldsymbol{R}$ given by $s(u,v) = \|c_1(u) - c_2(v)\|$ is a height field giving the distance between points on the curves for each point in the space $I_u \times I_v$. For a given $\epsilon > 0$ the level set of the surface $s(u,v) = \epsilon$ encloses regions of $I_u \times I_v$ where the height field is less than ϵ. These regions have been termed the *free space* in [2]. The question of whether the Frechet distance between $c_1(u)$ and $c_2(v)$ is less than ϵ reduces to whether there exists a continuous and monotone curve in the free space of $I_u \times I_v$ from (u_s, v_s) to (u_e, v_e).

The Frechet distance between curves can be computed analytically in the case that $c_1(u)$ and $c_2(v)$ are both polygonal curves (i.e., piecewise linear). Alt and Godau [2, 34] give algorithms for determining the Frechet distance between polygonal curves assuming the two curves have linear parametrizations and unit length parametric intervals for each segment. Assuming P and Q are two such curves, each consisting of a **single** segment, they show that the free space for a given ϵ is convex and in fact consists of a (possibly degenerate) ellipse intersected with the unit square. For P and Q polygonal curves with **arbitrary** numbers of segments, and defined on $[0, p]$ and $[0, q]$ respectively, the free space for a given $\epsilon > 0$ is composed of unit square sized cells in $[0, p] \times [0, q]$, each intersected with an ellipse. To determine whether $\mathcal{F}(p, q) < \epsilon$ they use an algorithm of $\mathcal{O}(pq)$ that determines whether a continuous monotone curve from $(0, 0)$ to (p, q) can be contained in the free space. This algorithm depends on the convexity of the free space within individual unit area cells of $[0, p] \times [0, q]$.

To actually compute the Frechet distance, they observe that a minimal ϵ occurs only under certain conditions as the free space graph grows with increasing ϵ. They relate these conditions to critical values that can be determined readily from the geometry of the two curves. They then use the previous algorithm to determine which is the smallest critical value that bounds the Frechet distance.

Elber in [24] observes that for $c_1(u)$ and $c_2(v)$ NURBS curves, the surface $\delta(u, v) = \|c_1(u) - c_2(v)\|^2 = < c_1(u) - c_2(v),\ c_1(u) - c_2(v) >$ can be represented as a NURBS. He further observes that determining the free space for these curves given an $\epsilon > 0$ is then reduced to finding the zero set of $\delta(u, v) - \epsilon^2$. He also discusses using the surface $\delta(u, v)$ to find intersection, closest, and furthest points of the two curves.

Section 4.4 presents a method for bounding Frechet distance between curves. This method uses closest point matches on the two curves as a heuristic for finding a change of parameter function that demonstrates a bound on Frechet distance. Closest point matches have also been used in computer vision research. Zhang [71] gives an algorithm to match polygonal curves by using closest point matches on the curves. The curve matching is done in the context of motion estimates for autonomous vehicle navigation with the curves coming from successive frames of captured video or range sensor data. The algorithm iteratively matches points in the first frame with their closest points on the curves in the second frame. This is done by first applying a motion estimate to the points on the curves of the first frame. Points that can not be matched or matches with distances outside an adaptively established deviation are discarded. An objective function is minimized based on the point matches in order to determine a suitable rigid motion between the frames. This motion estimate is then applied to points in the first frame and the algorithm iterates. In [72] these techniques are extended to matching surfaces between successive frames. Similar work has also been done in [5]. The primary application of these algorithms is an understanding of rigid body motion in computer vision. The authors do not apply these techniques to the reparametrization of curves or surfaces.

3.2.2 Other Metrics

The Frechet distance between curves is stated in terms of a min/max constraint. Its advantage is that it gives a measure of similarity between two curves over the entire path of both curves. Other metrics between parametrically defined curves are possible of course. Emery in [25] details an algorithm for the computation of the Hausdorff distance [56] between planar polygonal curves. This algorithm is applied to the creation of piecewise linear approximations to more general curves with bounded curvature. The Hausdorff metric between curves, however, considers the curves as point sets without regard to their parametrization. Curves that have a small Hausdorff distance are guaranteed never to stray far from one another, but they may trace out their paths in very different manners. A small Frechet distance between curves, on the other hand, indicates that a minimum distance can be maintained while tracing the curves simultaneously; a notion that considers curves as parametric entities.

Regular parametric curves are guaranteed to have arc length parametrizations. This notion can be used to form a metric between curves. In [31] a distance metric is defined as:

$$d(a(u), b(v)) = \sup_t \|\alpha(t) - \beta(t)\|$$

for α and β approximate normalized arc length parametrizations of a and b respectively. The arc length parametrization is approximated as referenced in section 3.1 above. This metric and the Frechet distance can yield substantially different results.

3.3 Texture Mapping and Grid Generation

Texture mapping and grid generation both can be thought of as forms of reparametrization.

Texture mapping is used to enhance geometric models with detail that is too fine to specify explicitly. Concern in texture mapping schemes often centers on the minimization of metric distortion to the texture as it is mapped to the geometry of the model [4, 51, 45]. Other issues, such as the distribution of texture memory with respect to the importance of texture detail can be considered also [63]. Texture mapping requires functions that map from the parameter space of a surface into texture space; this allows acquisition of texture values for given locations in the parametric domain. These maps are often, though not always, one to one. When invertible, the inverse maps can be thought of as reparametrization functions for parametric surfaces. Texture mapping, however, is seldom concerned with the explicit representation of the composed surface form in the representational scheme of the modeler.

Grid generation involves imposing a coordinate grid on a curve or surface representation. Applications include simulations, as in the field of computational fluid dynamics (CFD), and the adequate display of curves and surfaces by polygons. The properties desired for these grids are very application dependent. Study of turbulent flow, for example, may require fine spacing of grid lines adjacent to boundary layers of the model. In [64], methods are given for forming and remapping grids on NURBS surfaces for CFD analysis. Remapping surface grids provides compatibility for different types of surface analysis techniques. In [42] the desired property is that a uniform grid in the parametric domain of a curve or surface yield a grid of points in $\boldsymbol{R}^3$ with close spacing in areas of high curvature and sparse spacing in ares of low curvature. A discretization of the problem is solved numerically. The author in [53] formulates a general statement of an initial value problem for grid generation on curves and surfaces given a user specified distribution function. This initial value problem is then solved numerically. As with texture mapping, the concern in grid generation is generally **not** with the explicit representation of a reparametrized curve or surface using the representational scheme of the modeler.

3.4 Correspondence and Reparametrization

Correspondence in geometry involves finding matches, or associations, between shapes. Establishing correspondences between geometry is an important subproblem in computer vision, surface reconstruction from medical imagery, and image or shape metamorphosis for computer graphics. Correspondence between shapes also plays an important role in much geometric design; both for esthetic and functional reasons.

Reparametrization could be an important tool for expressing correspondence in CAD systems that use parametric representations. A reparametrization establishes a correspondence between a shape and itself and can be used to establish correspondences between different, but related, shapes. Two examples, drawn from the shape metamorphosis literature, are given below to illustrate the relationship between correspondence and reparametrization problems.

Shape metamorphosis for computer graphics has received much attention in recent years. The technique involves the warping, or deformation, of one shape to match another. The effect produced depends greatly on the nature of the correspondence established between these shapes. This correspondence is specified explicitly using matching points or feature lines [3, 44], by automatic techniques that seek to minimize some functional established between images or shapes [61, 15], or by a combination of explicit and automatic techniques [16].

Sederberg et al. in [61] discuss physically based metamorphosis for polygonal shapes. The correspondence problem is solved by minimizing functions that simulate the work done in bending and stretching one shape to match the other. This correspondence, which is specified by inserting and reordering vertices subject to certain restrictions, is effectively a reparametrization of the polygonal chain that specifies one of the shapes.

In [15] correspondence between free form NURBS curves is determined by adapting minimization of continuous functionals to discrete data. Applications include shape metamorphosis and avoiding degeneracies in surface constructions. A change of parameter function is fit to a set of discrete point matches between two different curves and one of the curves is then reparametrized to effect the correspondence.

3.5 “Relabeling” of Discrete Interpolation Data

A methodology employed in curve interpolation involves assigning and/or reassigning parametric values to discrete geometric points of interpolation. This technique is often referred to as *curve reparametrization* in the literature; it will be referred to as *data relabeling* here. The notion of curve reparametrization as developed in this thesis involves the direct composition of curve functions, i.e., $c(r(t))$, or the approximation of these compositions. Thus an infinite number of geometric points become “relabeled” with new parameter values.

Any further relationship between the concepts of data relabeling in interpolation schemes and curve function composition depends on the context in which the interpolation schemes are developed. Typical uses for curve interpolation methods are: 1) ab initio design, 2) the maintenance of given geometric constraints, and 3) the approximation of a point sampled curve by another curve that has desired properties lacking in the original.

In cases 1) and 2), relabeling data points with new parametric values is typically done to effect changes in the curve shape (this is often combined with some optimization scheme to enforce a measure of "fairness" to the interpolant's shape).

Examples of case 3) are approximation for smoothness, data compaction, or curve reparametrization. This last use is obviously related to the work in this thesis. Here the desire is to **maintain** shape if possible. Examples of interpolation schemes of this nature have already been cited in section 3.1 above.

CHAPTER 4

REPARAMETRIZATION ALGORITHMS

This chapter develops new algorithms for the reparametrization of NURBS curves. The next section presents a general framework that gives a unified approach for developing these algorithms. Sections 4.2 through 4.4 specialize this framework to individual reparametrization algorithms used to:

- approximate arc length parametrizations of curves,
- approximate inverses of NURBS functions, and
- generate reparametrizations that establish user specified tolerances as bounds on the Frechet distance between parametric curves.

4.1 Framework

This section presents a framework for new algorithms that approximate important cases of NURBS curve reparametrizations.

The central problem for these algorithms is to approximate $f(t) = c(r(t))$ by a NURBS curve where c is a NURBS curve. Throughout this section it is assumed that r is **not** a NURBS. The techniques developed here can be used, however, if r **is** a NURBS and we wish to approximate r in the context of $c(r(t))$ (or wish to approximate $c(r(t))$ itself) by a NURBS curve of lower polynomial order.

The algorithms in this framework:

- exhibit error bounds,
- exploit known invariants about the composition $c(r(t))$ to determine when convergence to an accurate result has been achieved,
- have a means of iteratively acquiring more data on r, or a converging approximation to r, and

- can represent their results either as approximations to r or approximations to $c(r(t))$.

Error bounds should be expressed in meaningful metrics for each algorithm. An error bound for an arc length reparametrization, for example, can be expressed as a fractional variation in speed. The user may then specify an allowable tolerance for this variation. For establishing bounds on the distance between curve mappings the Frechet distance is used as the error metric. The user then specifies a distance in Euclidean space as an allowable tolerance.

Even when it may be difficult to bound error with an **intuitive** metric, it is usually possible to use a metric that allows the user to specify tolerance values that control convergence of the approximation in a predictable manner.

Given an error metric, the algorithms in this framework compute NURBS functions from which bounds on the error in an approximation are determined. These error bounds are formed over intervals of the domain of the approximation by using the properties of NURBS curves discussed in section 2.2.

4.1.1 Method 1 – A General Iterative Refinement Scheme

Directly approximating $c(r(t))$ from sampled data typically results in approximations that do not have the exact geometric shape of the original curve c. Algorithms using such approaches must check that their approximations converge to both the correct parametrization **and** the correct shape.

This presents a problem in that checks for errors in approximations to curves are most readily done using equal-parameter distance measures (see section 2.2.3) which are useful for measuring the distance between curves of **like** parametrization. The algorithms wanted here, however, may change **both** shape and parametrization.

An approach for resolving this issue is to use NURBS/NURBS function composition to converge to the correct parametrization without affecting geometry.

Denote by $\mathcal{S}^i = \mathcal{S}_{m,\tau^i}$ the ith approximating polynomial, or rational, spline space given by the ith knot vector τ^i and fixed order m. The ingredients for the general scheme are:

- An initial spline space, $\mathcal{S}^0$, for approximating r.
- A monotone approximation operator $\mathcal{M}[r, \mathcal{S}^i]$, which approximates its first argument by a NURBS in the space $\mathcal{S}^i$. Choices for this approximation operator are discussed

in section 4.1.2.

- A metric, or pseudo metric, $d(a^i, f)$, that measures the distance between the ith approximation a^i and $f = c(r(t))$. It is important that we be able to bound this distance by a function $b^i(t)$ that can be represented by a NURBS. Some general choices for d are discussed in section 4.1.3. Techniques for bounding d will be discussed in sections 4.2.1, 4.3.1, and 4.4.2.

- A refinement scheme $\mathbf{refine}(\tau^i, b^i(t))$. Given error bound $b^i(t)$ and fixed order m this scheme will produce the knot vector τ_{i+1} which determines the approximating spline space for the next iteration of the algorithm. Some general characteristics of these refinement schemes are discussed in section 4.1.4.

The general scheme is as follows:

Input:

$c(u) : I_u \to \boldsymbol{R}^n$ a piecewise regular NURBS curve map,

$r(t) : I_t \to I_u$ a piecewise allowable change of parameter with $r(I_t) = I_u$,

m the order of approximating functions for r,

τ^0 the knot vector for the initial approximation of r, and

$\epsilon > 0$ an error tolerance measure.

Output:

Approximations for $f(t) = c(r(t))$ or $r(t)$.

Algorithm:

1. $i = 0$
2. $r^i(t) = \mathcal{M}[r, \mathcal{S}^i]$
3. form $a^i = c(r^i(t))$ as a NURBS curve.
4. find a NURBS function $b^i(t)$ which bounds $d(a^i, c(r(t)))$.
5. if $\max_t b^i(t) > \epsilon$ then

   $\tau^{i+1} = \text{refine}(\tau^i, b^i(t)), \quad i = i + 1, \;$ go to 2.
6. else return $r^i(t)$ or $a^i(t)$.

Of key importance is the development of approximation operators, refinement schemes, and error metrics that cause the approximations formed in the algorithm to converge to the correct result. These issues are discussed in the next several sections.

4.1.2 Approximation Operators $\mathcal{M}$

This section discusses the monotone approximation operator $\mathcal{M}[r, \mathcal{S}^i]$ of Method 1. This operator is used to approximate reparametrization function r by a NURBS curve in the space $\mathcal{S}^i$.

Generally this operator will use discrete data resulting from point-wise evaluation, or point-wise approximation, of r. The data may include both function and derivative values. It is assumed the data reflect strictly monotonic functions (without loss of generality, assume these functions to be strictly increasing).

As an example consider approximations to the arc length parametrization. For NURBS curve $c(u) : I_u = [u_s, u_e] \to \boldsymbol{R}^3$ we have $r = s^{-1}$ where $s = s(u) = \int_{u_s}^{u \in I_u} \|\frac{d}{d\mu}c(\mu)\| d\mu$ is the arc length function. In general, function values for s, and hence s^{-1}, can only be approximated using numerical integration techniques (examples of such techniques are given in Appendix C). Derivative values for s^{-1}, however, can be computed directly from c.

Since we assume that all curves are piecewise regular and all change of parameter functions are piecewise allowable (see section 2.1.4), the data must be approximated or interpolated by a function that preserves monotonicity. Interpolation and approximation schemes that preserve convexity present in the data as well, may help to increase the rate of convergence of Method 1.

The Schoenberg variation diminishing spline approximation [52] is a simple monotonicity preserving scheme. This choice for $\mathcal{M}$ allows an approximation to r in any spline space $\mathcal{S}^i$, defined on a suitable parametric interval, and requires only function values. The continuity of this scheme depends on the knot vector τ_i chosen for the space. For polynomial order m and singleton knots in the interior of τ_i, this scheme yields C^{m-2} continuity. The major drawback of the variation diminishing spline approximation is its slow convergence.

The simplest shape preserving **interpolant** is a piecewise linear function. This scheme trivially preserves both monotonicity and convexity in the data. Linear schemes also have the virtue of preserving the order of polynomial functions under composition.

Shape preserving C^1 spline interpolation schemes are presented in [60] and [32], overviews of which are given in Appendix B. The schemes, as given in [60] and [32], produce C^1 interpolants; however this is not always appropriate if it is known a priori that the reparametrization function is **not** C^1 everywhere. The interpolation schemes can be augmented readily, however, to lower continuity at specified points in the interpolant's domain (see Appendix B). This lowering of continuity in the interpolant is critical to constructing C^1 approximate arc length parametrizations from G^1 (but not C^1) curves. (See section 4.2 for details.)

The interpolant of [60] may fail to maintain shape conditions; however such failures can be detected. This is especially important for detecting failures in monotonicity over intervals of interpolation. One method for "correcting" such failures is to lower the continuity at the endpoints of the errant interval (see Appendix B). Since this scheme is embedded in an iterative algorithm, this failure condition can be flagged, and another iteration of the outer algorithm initiated to collect more data on the interval in an attempt to further correct the problem and maintain C^1 continuity.

4.1.3 Metrics

The choice of metric, or pseudo metric, $d(f, g)$ for Method 1 depends on the application domain. Some choices for d are given below. Their use in reparametrization algorithms are discussed in sections 4.2 through 4.4.

1. $\sup_t \|f(t) - g(t)\|$, equal-parameter distance between mappings.

2. $\mathcal{F}(f, g)$, Frechet distance.

3. $\sup_t |\ \|f'(t)\| - \|g'(t)\|\ |$, maximum difference of first derivative length functions.

4.1.4 Refinement Schemes

Refinement schemes for Method 1 should be viewed in a general sense since τ^{i+1} may not be a simple refined partition of τ^i.

For algorithms that use interpolation of sample points for operator $\mathcal{M}$ it may be more natural to view the refinement scheme as generating a more refined or dense sampling of the function r to be approximated. In cases where the change of parameter function r is determined wholly, or in part, by c, the curve being reparametrized, this more refined sampling of r may actually be generated by embedding NURBS curve c in increasingly refined spline spaces. (Examples of this technique are given in sections 4.2 through 4.4.) The new knot vector τ^{i+1} may then be determined by assignment of parameter values to these sample points, by consideration of known discontinuities that should occur in a^i or r^i, and by the exact nature of operator $\mathcal{M}$.

The next several sections give specializations of Method 1 for performing particular reparametrizations.

4.2 Approximate Arc Length Reparametrization

Here Method 1 is specialized to reparametrize a NURBS curve $c(u)$ by approximate arc length. The resulting approximate arc length parametrization will be C^1 if $c(u)$ is G^1 and have a variation in speed bounded by a user specified tolerance. Given piecewise regular NURBS curve $c(u) : I_u = [u_s, u_e] \to \boldsymbol{R}^3$, the arc length function on $c(u)$ is defined as:

$$s(u) = \int_{u_s}^{u \in I_u} \left\| \frac{d}{d\mu} c(\mu) \right\| d\mu$$

The arc length parametrization is given by $\gamma(t) = c(s^{-1}(t))$ and is characterized by $\|\frac{d\gamma(t)}{dt}\| \equiv 1$; that is, an arc length parameterization has unit speed. This invariant will be exploited to measure convergence of the sequence of approximations produced by the algorithm.

It is known that arc length parametrizations of NURBS curves can themselves be expressed as NURBS curves only for very restricted cases [30]. Hence representing the arc length parametrization of a NURBS curve by a NURBS is, in general, an approximation problem. The user specified tolerance for this approximation can be given as an allowed variation in speed.

4.2.1 Bounding a Metric

Pseudo metric 3 of section 4.1.3 is used for $d(f, g)$ which then specializes to

$$d(a^i(t), \gamma(t)) = {}^{\sup}_{t} \left| \left\| \frac{d}{dt} a^i(t) \right\| - \left\| \frac{d}{dt} \gamma(t) \right\| \right| = {}^{\sup}_{t} \left| \left\| \frac{d}{dt} a^i(t) \right\| - 1 \right|$$

for $\gamma(t)$ the arc length parametrization.

Since $a^i(t)$ is a NURBS curve, $\frac{d}{dt} a^i(t)$ can be represented as a NURBS curve (see section 2.2.2). The techniques of section 2.2.3 can then be used to bound $\left\| \frac{d}{dt} a^i(t) \right\|$ from both above and below on intervals of the parametric domain. Hence $\left| \left\| \frac{d}{dt} a^i(t) \right\| - 1 \right|$ can be bounded over intervals of the domain to ensure that the speed of the approximations

$a^i(t)$ converge to within a user specified tolerance of unit speed everywhere in their domains.

4.2.2 Approximating Inverse Arc Length

Function $r^i(t)$ of Method 1 is computed from points $Q^i = \{(u^i_j, t^i_j)\}_j$, approximations of the arc length function for $c(u)$ at discrete points in the curve's domain. The points $R^i = \{(t^i_j, u^i_j)\}_j$ are then on an approximation to the inverse arc length function. The algorithms of Appendix C give methods for computing sets of sample points for approximate arc length functions. In one of these algorithms, lengths along the control polygon for refined representations of the NURBS curve $c(u)$ are used for arc length approximation. Iterations of this algorithm can be incorporated into the iterations of Method 1.

To avoid oversampling, data should be acquired for the arc length approximation only where needed. If the error bound for $c(r^i(t))$ on an interval $I = [t^i_j, t^i_{j+1}] \subseteq I_t$ exceeds the specified tolerance then sampling for the arc length function in the next iteration is increased in an interval $\hat{I} = [u^i_j, u^i_{j+1}] \subseteq I_u$. More generally, if the error bound for $c(r^i(t))$ on an interval $[a, b] \subseteq I_t$ exceeds the specified tolerance, then sampling for the arc length function is increased on the interval $[r^i(a), r^i(b)] \subseteq I_u$ in the next iteration.

4.2.3 The Approximations $\mathbf{r^i}$ and Continuity of $\mathbf{a^i(t)}$

There are a number of possibilities for approximating the inverse arc length function given sample points R^i. This section describes the use of the C^1 interpolants of [60] and [32] (see Appendix B). Use of such interpolants allows the construction of C^1 approximations $a^i(t) = c(r^i(t))$ from G^1 curves $c(u)$.

Data supplied to these schemes must be in the form $\{(t_j, u_j, D_j)\}_j$ where the u_j represent function values, the D_j represent first derivative values, and the t_j are the locations in the parametric domain where the given values are to be achieved. The $\{(t_j, u_j)\}_j$ correspond to the R^i above. The derivative values D_j can be computed from the speed function on the original curve $c(u)$ with $D_j = 1/\left\| \frac{dc(u)}{du}|_{u=u^i_j} \right\|$ in accordance with the inverse function theorem for derivatives. Note that because of the interpolation constraints, $\left\| \frac{dc(r^i(t))}{dt}|_{t=t^i_j} \right\| = \left\| \frac{dc(u)}{du}|_{u=u^i_j} \frac{dr^i(t)}{dt}|_{t=t^i_j} \right\| = \left\| \frac{dc(u)}{du}|_{u=u^i_j} \right\| / \left\| \frac{dc(u)}{du}|_{u=u^i_j} \right\| = 1$. Thus at each interpolation point t^i_j the speed of $c(r^i(t))$ becomes unity.

The interpolation schemes of [60] and [32] produce C^1 spline interpolants to the inverse arc length data. The arc length function, however, will be only C^0 at points where the speed function for $c(u)$ is discontinuous. Hence the inverse arc length function will be only C^0 at the associated points. The interpolation schemes can be augmented readily to insert knots of multiplicity two at corresponding points in the interpolant's domain, constraining the resulting interpolants to be only C^0 there (see Appendix B). The augmented algorithm requires both left and right hand derivative information at such points. These data can be supplied from left and right speed values on $c(u)$. Again, because of the interpolation conditions, the left and right speed values for the composed function $c(r^i(t))$ will be 1.0 at these points. Thus if $c(u)$ is a G^1 curve, $c(r^i(t))$ will be C^1 for all i provided that each set Q^i includes all points in the domain of $c(u)$ where the curve is only C^0.

4.3 Approximation of Inverse NURBS Functions

This section specializes Method 1 to an algorithm for approximating inverses of scalar valued NURBS functions. As mentioned before, it is important to note that this **is** an approximation problem since for $c(u) : I_u \rightarrow \boldsymbol{R}$ a NURBS function, $c^{-1}(t)$ is usually **not** a NURBS.

The series of functions r^i of Method 1 become approximations to c^{-1}, and the invariant used to determine the convergence of the algorithm is that $a^i(t) = c(r^i(t))$ should converge to the identity function. The user specified tolerance can be given as an allowable distance of $a^i(t)$ from the identity, or, alternatively, an allowable distance of $r^i(t)$ from $c^{-1}(t)$.

4.3.1 Bounding a Metric

Two metrics are proposed for measuring error for this algorithm. For $f(t) = c(c^{-1}(t)) = t$ the two metrics under consideration are:

$$d(a^i(t), f(t)) = \sup_t |c(r^i(t)) - t| = \sup_t |E(t)|, \tag{4.1}$$

and

$$d(r^i(t), c^{-1}(t)) = \sup_t |r^i(t) - c^{-1}(t)| = \sup_t |e(t)|. \tag{4.2}$$

For NURBS functions c and r^i, error function $E(t)$ can be represented as a NURBS (see section 2.2) and therefore can be bounded over intervals of its parametric domain. For a given interval I_0 of the parametric domain for the NURBS representation of $E(t)$ let $\{E_k\}_k$ be the B-spline coefficients for $E(t)$ for which the corresponding basis functions have positive support on I_0. Then $\max_k |E_k| \geq E(t)\ \forall_{t \in I_0}$.

Bounds on error function $e(t)$ can be related to bounds on $E(t)$ as follows: Assume $c(u)$ is a strictly monotonic increasing function. For a given interval $[u_a, u_b] \subset I_u$ let $t_a = c(u_a), t_b = c(u_b)$, and $\hat{I} = [u_a, u_b] \cup [r^i(t_a), r^i(t_b)]$. Let $c(u)$ be continuous on $\hat{I}$ and differentiable in the interior of $\hat{I}$. Then

$$|E(t)|/M \leq |e(t)| \leq |E(t)|/m \tag{4.3}$$

for $\forall_t \in [t_a, t_b]$ with $m = \min_{u \in \hat{I}} \frac{d}{du}c(u)$ and $M = \max_{u \in \hat{I}} \frac{d}{du}c(u)$. Since c is a NURBS function $\frac{d}{du}c(u)$ can be formed as a NURBS and its value bounded over intervals of the parametric domain.

To show relation (4.3) we have from the mean value theorem for derivatives:

$$\begin{aligned}
c(r^i(t)) - c(c^{-1}(t)) &= \left.\frac{d}{du}c(u)\right|_{u_0} (r^i(t) - c^{-1}(t)) \\
\Rightarrow \quad E(t) &= \left.\frac{d}{du}c(u)\right|_{u_0} e(t) \\
\Rightarrow \quad |E(t)| &= \left.\frac{d}{du}c(u)\right|_{u_0} |e(t)| \\
\Rightarrow \quad |e(t)| &= |E(t)| / \left.\frac{d}{du}c(u)\right|_{u_0} \quad \text{for some } u_0 \in \hat{I}
\end{aligned}$$

from which relation (4.3) follows.

4.3.2 The Approximations $\mathbf{r^i}$

The algorithm computes function r^i from sample points $Q^i = \{(u^i_j,\ t^i_j = c(u^i_j))\}_j$ on NURBS function $c(u)$. The points $R^i = \{(t^i_j, u^i_j)\}_j$ are then on the function $c^{-1}(t)$. Bounds on the error function $E(t)$ (or $e(t)$) over intervals of the domain, can be used to control where new sample points are evaluated in each iteration in a manner similar to that given in section 4.2.2. If the error bound for $E(t)$ on an interval $I = [t^i_j, t^i_{j+1}] \subseteq I_t$ exceeds the specified tolerance, sampling of $c(u)$ in the next iteration is increased in an interval $\hat{I} = [u^i_j, u^i_{j+1}] \subseteq I_u$. More generally, if the error bound for $E(t)$ on an interval $[a, b] \subseteq I_t$ exceeds the specified tolerance, then sampling of $c(u)$ is increased on the interval $[r^i(a), r^i(b)] \subseteq I_u$ in the next iteration.

In order to use the shape preserving quadratic or linear rational approximation schemes described in Appendix B, derivative information is required at each sample point. This information can be acquired easily by application of the inverse function theorem for derivatives. For sample point (t^i_j, u^i_j) the derivative value for r^i is set to $1/\frac{d}{du}c(u^i_j)$. Thus at each interpolation point t_j^i the derivative of the composed function $c(r^i(t))$ becomes unity.

Again, as in section 4.2.3, the continuity of the approximations r^i should be lowered at those points corresponding to lowered continuity of $c(u)$.

4.4 Bounding Frechet Distance

This section specializes Method 1 for the computation of bounds on the distance between parametric mappings. This algorithm will attempt to establish a user specified tolerance ϵ as an upper bound on the Frechet distance between two NURBS curves. The two curves, $c_1(t) : I_t = [t_s, t_e] \rightarrow \boldsymbol{R}^n$ and $c_2(u) : I_u = [u_s, u_e] \rightarrow \boldsymbol{R}^n$, are assumed to match at their end points; i.e., $\|c_1(t_s) - c_2(u_s)\| \leq \epsilon$ and $\|c_1(t_e) - c_2(u_e)\| \leq \epsilon$.

The algorithm reparametrizes c_2, using a sequence of piecewise allowable changes of parameter r^i, in an attempt to match c_1. From consideration of the definition of Frechet distance, one approach would be to construct a sequence of functions r^i such that

$$\lim_{i \to \infty} \sup_t \|c_1(t) - c_2(r^i(t))\| = \mathcal{F}(c_1, c_2). \tag{4.4}$$

The algorithm uses a heuristic to construct a sequence r^i that, although not guaranteed to have property (4.4), can be used to establish ϵ as a bound on $\mathcal{F}(c_1, c_2)$ in many cases where it **is** a bound. The heuristic is that c_2 should be reparametrized so that pairs of closest points on the two curves have the same parameter value. The rationale for this heuristic is that it holds for different parametrizations of the same Frechet curve and generalizes to many cases involving pairs of distinct Frechet curves. Note also that closest point matching guarantees finding a point in the free space for the Frechet problem (see section 3.2.1) at a given parameter value, if such points exist.

4.4.1 Closest Point Pairings

Our heuristic involves using pairs of closest points between two curves. Figure 4.1 indicates that such pairings are not necessarily unique. If $\mathbf{B} = c_1(t_0)$ is the closest point on c_1 to $\mathbf{A} = c_2(u_0)$, the closest point on c_2 to $\mathbf{B}$ is **not** guaranteed to be $\mathbf{A}$. The closest point pairings for this algorithm will be defined always to result from performing the minimization operation over nonempty intervals of c_1; i.e., curve c_1 is always searched for a closest point to an individual point on c_2.

Figures 4.2(a) and (b) indicate that this restriction can give rise to problems also. Figure 4.2(a) indicates that closest point pairings are still not necessarily unique since

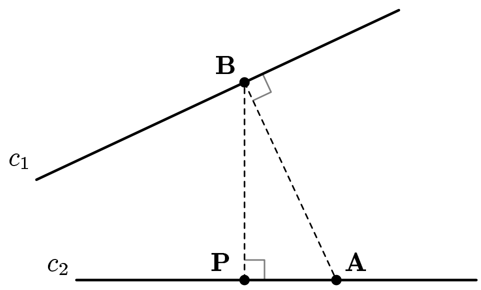


Figure 4.1. Closest point pairings are not necessarily unique since the closest point relation is not always symmetric. The closest point on c_1 to $\mathbf{A}$ is $\mathbf{B}$. However, the closest point on c_2 to $\mathbf{B}$ is $\mathbf{P}$.

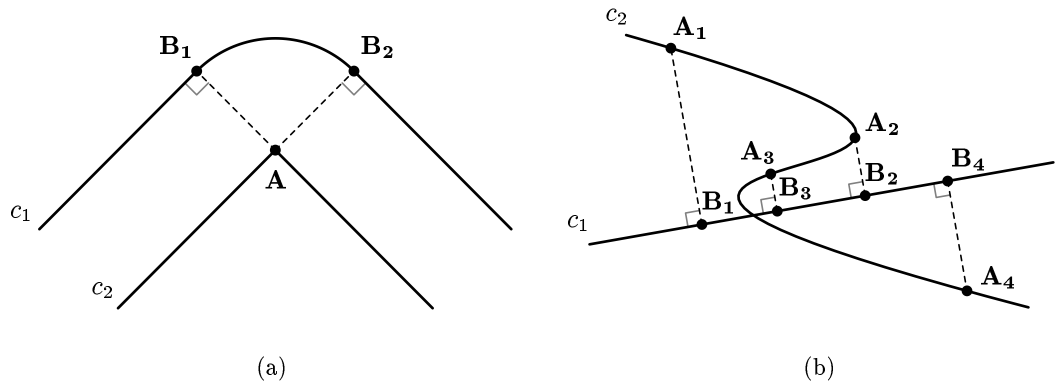


Figure 4.2. Closest point pairings. (a) More than one solution to the minimum distance problem. All points from $\mathbf{B_1}$ to $\mathbf{B_2}$ on c_1 are at a minimum distance from point $\mathbf{A}$ on c_2. (b) Point matches causing nonmonotonic parametric correspondence. The sequence of parameter values on c_2 associated with the ordered set $\{\mathbf{A_1}, \mathbf{A_2}, \mathbf{A_3}, \mathbf{A_4}\}$ is monotonic. The sequence of parameter values on c_1 associated with the ordered set of closest matching points $\{\mathbf{B_1}, \mathbf{B_3}, \mathbf{B_2}, \mathbf{B_4}\}$ is **not** monotonic however.

more than one point can be the solution for the minimization problem. Figure 4.2(b) indicates that closest point pairings can give rise to nonmonotonic change of parameter functions. Solving the minimization problem over the entire domain of c_1 each time may cause parameter matchings that "back up."

One approach to solving these difficulties in many cases is to use **local** solutions to the minimization problem. Hence searches on c_1 are performed over restricted intervals of its domain. If $c_1(t_j)$ is already paired with $c_2(u_j)$ and $c_1(t_{j+1})$ is already paired with $c_2(u_{j+1})$, then in finding a match for a point $c_2(u)$ with $u \in (u_j, u_{j+1})$, the search on c_1 is restricted to the interval (t_j, t_{j+1}). See section 4.4.3 below for more details. Note that restricting the search space for minimum point distance searches has the positive side effect of decreasing the computation necessary for these searches.

The algorithm makes use of the operation "find closest point on curve to point." See [41, 40] for recent work in this area.

4.4.2 Bounding a Metric

Metric 1 of section 4.1.3 is used for $d(f, g)$. Thus

$$d(c_1(t), c_2(r^i(t)) =^{\sup}_{t} \|c_1(t) - c_2(r^i(t))\| \geq \mathcal{F}(c_1(t), c_2(r^i(t)).$$

Establishing a bound for this choice of distance metric d is accomplished using the techniques of section 2.2.3.

4.4.3 Sample Points for $\mathbf{r^i}$

The algorithm computes function r^i using sample points resulting from pairs of closest points on the two curves. For each iteration i of the algorithm, a list of sample points $Q^i = \{(t^i_j, u^i_j)\}_j$ is maintained, sorted by increasing t^i_j, where $t^i_j \in I_t \ \forall_j$ and $u^i_j \in I_u \ \forall_j$. These tuples have the property that for each j, $c_1(t^i_j)$ is that point on c_1 which is found to be closest to $c_2(u^i_j)$ using a restricted closest point search on c_1 (as described below). We now discuss how to create the sets Q^i from which $r^i : I_t \to I_u$ is computed.

For $I_t = [t_s, t_e]$ and $I_u = [u_s, u_e]$, set Q^0 to $\{(t_s, u_s), (t_e, u_e)\}$. For $i > 0$, data should be acquired for r^i only where most needed; one approach is as follows: Let (t^i_j, u^i_j) and (t^i_{j+1}, u^i_{j+1}) be tuples in Q^i such that the error bound for $d(c_1(t), c_2(r^i(t))$ on $[t^i_j, t^i_{j+1}]$ exceeds the specified tolerance. Let $\hat{u} = (u^i_j + u^i_{j+1})/2$. In forming Q^{i+1} the tuple $(\hat{t}, \hat{u})$ is added to the set Q^i where $c_1(\hat{t}\,)$ is the closest point on c_1 to $c_2(\hat{u})$, with the closest point search on c_1 restricted to (t^i_j, t^i_{j+1}). If the resulting $\hat{t}$ is too close to either end of interval $[t^i_j, t^i_{j+1}]$, move $\hat{t}$ slightly towards the interior of the interval in order to maintain monotonicity.

This algorithm has two stopping criteria. If $\|c_1(\hat{t}\,) - c_2(\hat{u})\| > \epsilon$, the algorithm terminates and reports a failure to establish ϵ as a bound on the Frechet distance in one of two subconditions. Curve c_1 is searched over its entire domain for the point closest to $c_2(\hat{u})$ and c_2 is searched over its entire domain for the point closest to $c_1(\hat{t}\,)$. If either of these closest point distances are greater than ϵ then we know that $\mathcal{F}(c_1, c_2) > \epsilon$. Otherwise we can say only that the algorithm failed to establish ϵ as a bound. The algorithm terminates successfully if $d(c_1, c_2(r^i))$ is bounded by a value less than the user specified tolerance.

Note that $\{t^{i+1}_j\}_j$ is a refined partition of $\{t^i_j\}_j$ and similarly $\{u^{i+1}_j\}_j$ is a refined partition of $\{u^i_j\}_j$.

4.4.4 The Approximations $\mathbf{r^i}$

The algorithm can use sample points Q^i to yield piecewise linear approximations r^i. In order to use the C^1 schemes described in Appendix B, however, derivative information is required at each sample point. Two approaches for deriving this information are given below.

Assume c_1, c_2 piecewise regular, and $r : I_t \to I_u$ a piecewise allowable change of parameter with $\frac{d}{dt}r(t) > 0, \forall_t$, i.e., a sense preserving change of parameter. If $c_1(t) = c_2(r(t)),\ \forall_t \in I_t$ then $\frac{d}{dt}c_1(t) = \frac{d}{du}c_2(u)\Big|_{r(t)} \frac{d}{dt}r(t) \Rightarrow \frac{d}{dt}r(t) = \|\frac{d}{dt}c_1(t)\| / \|\frac{d}{du}c_2(u)\Big|_{r(t)}\|$.

This consideration leads to the following technique for approximating $\frac{d}{dt}r^i(t)$ at a point t^i_j. Given the sample point (t^i_j, u^i_j) for some j, evaluate $V_1 = \frac{d}{dt}c_1(t^i_j)$ and $V_2 = \frac{d}{du}c_2(u^i_j)$ and then approximate $\frac{d}{dt}r^i(t^i_j)$ by taking the ratio of the length of V_1 to the length of the projection of V_2 onto V_1. Thus:

$$\frac{d}{dt}r^i(t^i_j) \approx \frac{\|V_1\|}{< V_1, V_2 > / \|V_1\|} = \frac{< V_1, V_1 >}{< V_1, V_2 >}.$$

If the unit tangent functions for c_1 and c_2 are similar, then this is apt to yield a good approximation. However, it is possible for two curves to have a small Frechet distance between them and yet have very dissimilar unit tangent functions (see Figure 4.3). If the unit tangents $V_1/\|V_1\|$ and $V_2/\|V_2\|$ are dissimilar then a different method should be used for determining the derivative approximations.

One approach is to approximate derivative information based on the sample points Q^i alone. For example, a linear rational function can be fit to three successive points,

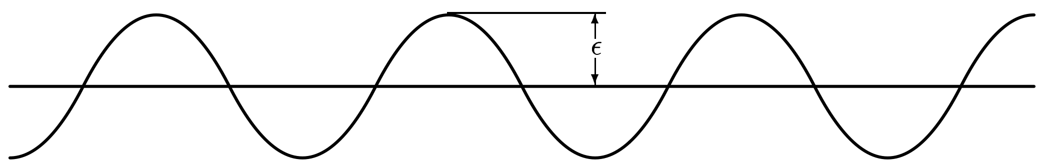


Figure 4.3. Curves with very dissimilar unit tangent functions but a small Frechet distance of ϵ between them.

$(t^i_{j-1}, u^i_{j-1}), (t^i_j, u^i_j), (t^i_{j+1}, u^i_{j+1})$, and the derivative of this function at t^i_j used for the derivative of r^i there. End points would be handled in a similar fashion. The use of a linear rational function for this purpose guarantees that derivative values will be consistent with monotonic functions given monotonic data (see [32]). For the case Q^0 either a third sample point is required interior to the interval, or a linear function is used for the first approximation r^0.

CHAPTER 5

ALGORITHMIC EXTENSIONS

This chapter develops extensions to the algorithms of Chapter 4. Section 5.1 presents an extension to the algorithm for approximating inverses of spline functions. Section 5.2 develops extensions to the Frechet distance algorithm of section 4.4. Section 5.3 gives an extension to Method 1, the general scheme developed in section 4.1.1. Section 5.4 gives an alternative to Method 1. Section 5.5 discusses a technique that can be used to tighten error bounds computed by the algorithms.

5.1 Reparametrization by Axis

This section extends the algorithm of section 4.3 to parametrizations of NURBS curves along arbitrary linear axes in space. By this is meant that equal steps in the parametric domain yield equal steps along the curve's perpendicular projection onto a specified linear axis. Using the affine invariance property of NURBS curves, it can be assumed, without loss of generality, that curve $c(u) = (x(u), y(u), z(u)) : I_u \to \boldsymbol{R}^3$ is to be parametrized along the X axis. It is further assumed that $x(u)$ is a strictly monotonic increasing function, although this algorithm can be generalized to piecewise monotonic coordinate functions.

The goal is to approximate the function $f(t) = (x(x^{-1}(t)), y(x^{-1}(t)), z(x^{-1}(t))) = (t, y(x^{-1}(t)), z(x^{-1}(t)))$ by a NURBS curve given $c(u)$ a NURBS. This approximation can be formed as $a^i(t) = (x(r^i(t)), y(r^i(t)), z(r^i(t)))$. The functions $r^i(t)$, approximations to $x^{-1}(t)$, are computed by the algorithm of section 4.3 using the metric of equation (4.1).

If the $r^i(t)$ are computed with the aid of a C^1 interpolation scheme, as suggested in section 4.3, then the resulting approximations $a^i(t)$ will be continuous, with X coordinate functions that are continuously differentiable, provided each set Q^i includes all points in the domain of $c(u)$ where the curve's X coordinate function is only C^0 (see section 4.3).

5.2 Extensions to Frechet Distance Algorithm

This section presents two extensions to the Frechet distance algorithm of section 4.4. The first addresses the problem of finding contact intervals between two curve mappings. The second modifies the Frechet algorithm of section 4.4 to yield an algorithm that produces relative radial reparametrizations of curve mappings.

5.2.1 Contact Intervals Between Curves

The algorithm of section 4.4 tries to construct a function that demonstrates a user specified tolerance as a bound on the Frechet distance between curve mappings. A useful and practical extension addresses the following problem: given two curve mappings c_1 and c_2, find and describe all contact intervals between the curves that the mappings represent.

As used here, the term *contact interval* means an interval over which the two curves have a Frechet distance less than a specified tolerance. Such an interval can be designated by:

- A parametric interval on curve 1.
- The corresponding interval on curve 2.
- A change of parameter function that reparametrizes one curve into the other over this interval, i.e., that establishes the specified tolerance as a bound on the Frechet distance over the interval.

To aid in the statement of this algorithm we define the *signal values* of a NURBS curve to be the set of parameter values corresponding to either the start or end of the curve, or to an internal knot with multiplicity $m-1$ for m the polynomial order of the curve. In short, the signal values of a NURBS curve are those parameter values where the curve is constrained to be only C^0. There are a finite, and generally small, number of such points on a NURBS curve.

This algorithm assumes that contact intervals between curves begin and end at parametric locations that are signal values on at least one (but not necessarily both) of the curves. Large classes of curves designed using a CAD system, and intended to be in contact along part of their domains, meet this criterion. Figure 5.1 indicates cases covered by this assumption. This assumption makes finding all **potential** contact intervals a

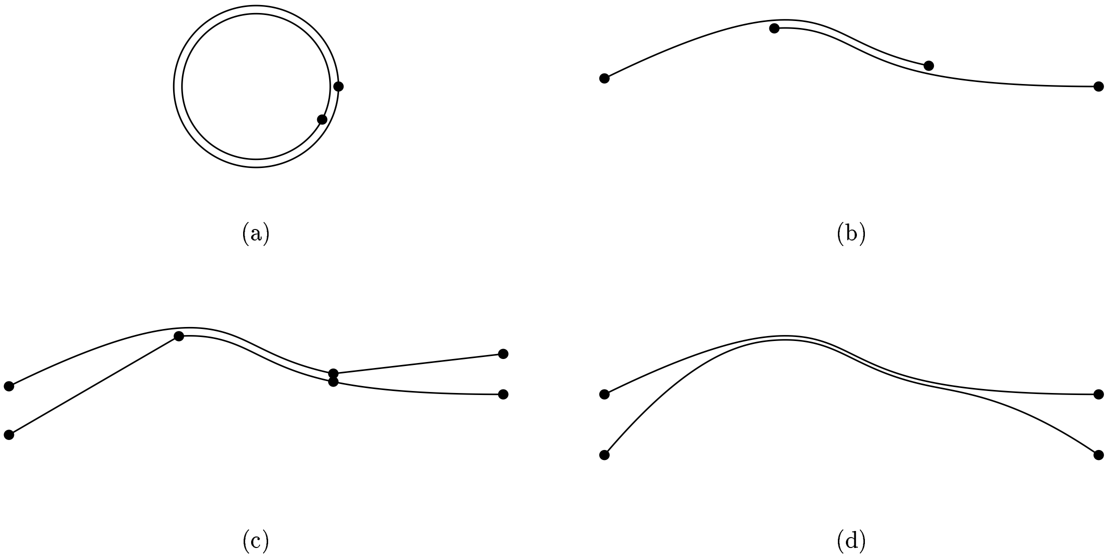


Figure 5.1. Cases covered and not covered by the basic assumption for contact intervals. Dots indicate the location of signal values on the curves. The cases illustrated in (a), (b), and (c) are covered by the assumption. Case (d) is **not** covered since the contact interval does not begin and end at parametric locations that are signal values on at least one of the curves.

discrete problem with a straightforward solution.

The algorithm uses the operator `find-closest-pt-tol( crv, pt, tol )`. The curve argument, `crv`, is searched for the closest point to the geometric point argument `pt`. If the closest point found is within distance `tol` of `pt` then the parametric location on `crv` corresponding to this closest point is returned. Otherwise an indication that no such matching point exists is returned.

5.2.1.1 General Change of Parameter

This section develops the algorithm for finding contact intervals between curve mappings when intervals may be related by general change of parameter functions. A special case for mappings related exactly by piecewise linear or piecewise linear rational change of parameter functions is given in the next section.

Input:

c_1 and c_2 piecewise regular NURBS curve maps.

ϵ a user specified tolerance given as a Euclidean space distance.

Output:

A list of contact interval entries. Each entry consists of a parametric interval on c_1, the corresponding interval on c_2, and the change of parameter function that reparameterizes c_2 into c_1 over this common interval.

Algorithm:

1. Create a sorted list of the signal values for c_1 and a sorted list of the signal values for c_2.
2. Partition the domain of c_1 using a sorted list of parametric locations. Each entry in this list contains a parameter value in the domain of c_1, a flag indicating whether this location has been matched by a point on c_2, and the matching parameter value on c_2 if a match was found. The list is sorted by increasing parameter value on c_1 and is formed as follows:
   (a) Add an entry for each signal value on c_1. For each of these entries use the `find-closest-pt-tol` operator to find a matching point on c_2, if one exists. If no match is found, mark the entry as unmatched. If a matching point **is** found, set the c_2 parameter value in the entry to the parametric location of the matching point. If the matching point is at a signal value on c_2, remove it from the list of c_2 signal values.
   (b) For each remaining unmatched signal on c_2 use the `find-closest-pt-tol` operator to find a matching point on c_1, if one exists. If a matching point is found, insert an entry into the list that partitions the domain of c_1 indicating the parametric location on c_1 and corresponding location on c_2. The new entry is inserted into the sorted list using the parameter value on c_1.

   At the end of this process the domain of c_1 has been partitioned into subintervals. The parametric locations forming this partition are marked as either matched or not matched on c_2. A subinterval that has both of its end points marked as **matched** denotes a *potential contact interval.*
3. Check each potential contact interval using the Frechet distance algorithm of section 4.4. If the intervals match, add an entry to the list of contact intervals to be returned.
4. Coalesce entries for matching intervals adjacent in the parameter space of c_1.

5.2.1.2 Piecewise Linear Correspondence

Many practical cases arise in CAD environments which involve curve mappings related **exactly** by piecewise linear or piecewise linear rational changes of parameter with relatively few pieces. A typical case occurs when surfaces share part of their boundary edges. This shared boundary often results from the same initial curve geometry that has been incorporated into the two surfaces in ways such that the parameterizations of the geometry no longer match but are related by linear or linear rational functions.

A simple modification to the algorithm of section 5.2.1.1 can be used to determine if one of these special case relationships exists between two curves. The algorithm follows the same steps as in section 5.2.1.1, except that instead of using the Frechet distance algorithm of section 4.4, each potential contact interval is tested to see if the curves differ only by a simple linear change of parameter over the interval. The linear function for an interval being tested is defined by the matching pair of parameter values (one from the domain of c_1 and one from the domain of c_2) at each end of the interval. The change of parameter function for adjacent matching intervals thus becomes a piecewise linear function.

For rational curve mappings, one can test for a linear **rational** change of parameter relationship over each potential contact interval. The linear rational function for an interval can be determined uniquely only if additional information is known on the interval, such as the values for an additional correspondence pair.

5.2.2 Radial Reparametrization

The Frechet distance algorithm of section 4.4 uses a distance metric and convergence criterion based on the Euclidean distance between points. Other metrics and convergence criteria could be employed instead. As an example we modify the Frechet distance algorithm by using a distance metric and convergence criterion based on the "radial distance" between points. This results in an algorithm that forms relative **radial** reparametrizations of curves.

Figure 5.2 illustrates the situation under consideration. Assume that planar, piecewise regular curves $c_1(t) : I_t \to \boldsymbol{R}^2$ and $c_2(u) : I_u \to \boldsymbol{R}^2$ are "star-shaped" with respect to a central point. Without loss of generality assume this point to be the origin $\mathbf{O}$. Curve c_2 will be reparametrized in order to establish a *radial correspondence* with c_1. This correspondence is characterized by a change of parameter function, $r(t) : I_t \to I_u$, such that the line through $c_1(t)$ and $c_2(r(t))$ also goes through $\mathbf{O}$ for all t, and such that both

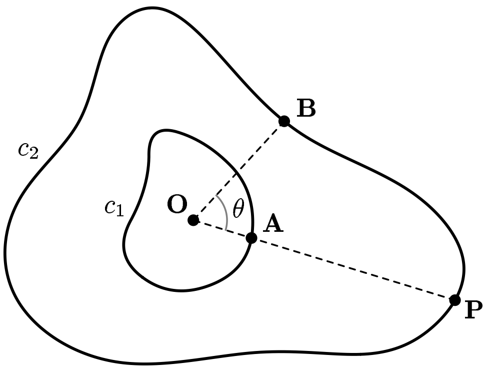


Figure 5.2. Radial distance and radial closest point operators between two curves. The radial distance between **A** on c_1 and **B** on c_2 is the angle θ made by the vectors $\overrightarrow{\mathbf{OA}}$ and $\overrightarrow{\mathbf{OB}}$ at the origin **O**. The radially closest point on c_2 to point **A** on c_1 is point **P** at the intersection of c_2 with the half line starting at **O** and going through **A**.

curves move from the points $c_1(t)$ and $c_2(r(t))$ into the same half space as defined by this line.

Given point **A** on c_1 and point **B** on c_2, the "radial distance" between these points is given by the angle between vectors $\overrightarrow{\mathbf{OA}}$ and $\overrightarrow{\mathbf{OB}}$, which we denote by $\angle_{\mathbf{O}}(\mathbf{A},\mathbf{B})$. Given point **A** on c_1, the radially closest point on c_2 is the point **P** at the intersection of c_2 with the half line starting at **O** and going through **A**. Thus the "find closest point on curve to point" operator of section 4.4.1 is a ray/curve intersection operation in this metric.

The algorithm of section 4.4 used the metric $\sup_t \|c_1(t) - c_2(r^i(t))\|$, for $\| \ \|$ Euclidean distance, to establish convergence and determine where more sample points for r^i, the approximation to r, should be generated. Here we use the pseudo metric

$$d(c_1(t), c_2(r^i(t))) = \sup_t \angle_{\mathbf{O}}(c_1(t), c_2(r^i(t))) \tag{5.1}$$

The user specifies a tolerance, θ, as the maximum allowable radial distance between $c_1(t)$ and $c_2(r^i(t))$. The pseudo metric of equation (5.1) can be bounded by:

$$\begin{aligned} & \sup_t \left| 1 - \frac{< c_1(t), c_2(r^i(t)) >^2}{< c_1(t), c_1(t) >< c_2(r^i(t)), c_2(r^i(t)) >} \right| \\ = \ & \sup_t \sin^2\left(\angle_{\mathbf{O}}(c_1(t), c_2(r^i(t)))\right) \\ \leq \ & \sin^2(\theta). \end{aligned} \tag{5.2}$$

For NURBS curves c_1, c_2, and r^i, the expression whose norm is taken in equation (5.2) is a NURBS function and hence its length can be bounded over intervals of its parametric domain (see section 2.2).

Substituting radial distance, the radial "find closest point on curve" operation, and the convergence criterion of equation (5.2) into the algorithm of section 4.4 yields an algorithm that produces a relative radial reparametrization of one curve to match another.

This algorithm can be embedded into the algorithm of section 5.2.1.1 yielding an algorithm that finds "radial contact intervals." For the star shaped curves assumed here, the signal values of section 5.2.1 need only include the start and end points of the curves (see Figure 5.3).

5.3 An Extension to Method 1

Method 1 of section 4.1.1 uses function composition to converge a sequence of approximations to a desired parametrization (see Figure 5.4). This general scheme has drawbacks which include:

- The function composition used may raise the polynomial degree of the result.
- For a given NURBS curve $c(u)$, choices for the polynomial degree of the result, $a^i(t)$, are restricted to multiples of the polynomial degree of $c(u)$.
- The smoothness of the result is tied directly to the smoothness of the approximations

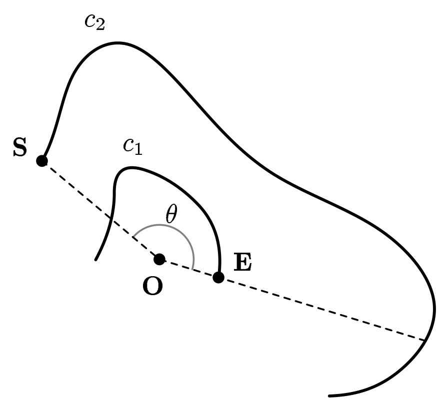


Figure 5.3. For star shaped curves, radial "contact intervals" begin and end at the start or end points of at least one of the curves. **S** and **E** indicate, respectively, the start and end points that define the contact interval in this example. The central angle θ here indicates the extent of the interval.

$$\begin{tikzcd} c(u) \arrow[r, "\mathcal{M}"] & r^i(t) \arrow[r] & c(r^i(t)) \arrow[r] & P? \arrow[lll, bend right, "i=i+1"'] \end{tikzcd}$$

Figure 5.4. Schematic for Method 1. The iteration counter for the algorithm is denoted by i. $\mathcal{M}$ is a monotone approximation operator to scalar valued data. "P?" checks for convergence to the correct parametrization.

$r^i(t)$. Smoother results may require a higher polynomial degree for $r^i(t)$.

A stage could be added to the algorithm which uses approximation techniques, in combination with equal-parameter distance measures, to allow the final approximation for $c(r(t))$ to be of any desired polynomial order.

Let $a(t) = c(r^{i^*}(t))$ be a NURBS representation for the function composition where i^* denotes the final iteration of Method 1. An interpolation or approximation operator could be used to approximate $a(t)$ from sampled data, yielding NURBS curve $g(t)$ in a spline space of any desired polynomial order. Equal-parameter distance $d(t) = {}^{\sup}_{t}$ $\|g(t) - a(t)\|$ can then be bounded over intervals of the domain using the techniques of section 2.2.3. Intervals of the domain where $d(t)$ is outside a user specified Euclidean space tolerance, ϵ_2, can then be resampled and the interpolation or approximation operator applied again. This new stage iterates until the approximation $g(t)$ is within the user defined tolerance of $a(t)$. Candidate interpolation and approximation techniques include those discussed below in section 5.4.1. Figure 5.5 diagrams Method 1 as augmented with this final approximation stage. Other data and degree reduction techniques could be used to accomplish this same goal [48, 49, 69, 68, 22, 57, 8].

One problem with this technique is that for general approximation or interpolation operator $\mathcal{A}$, the approximation that results at the end of the second stage may no longer be

$$\begin{tikzcd} c(u) \arrow[r, "\mathcal{M}"] & r^i(t) \arrow[r] & c(r^i(t)) \arrow[r] & P? \arrow[lll, bend right, "i=i+1"'] \arrow[r, "\mathcal{A}"] & g^k(t) \arrow[r] & D? \arrow[l, bend right, "k=k+1"'] \end{tikzcd}$$

Figure 5.5. Schematic for Method 1 augmented with a separate iterative approximation stage. The iteration counter for the second stage is denoted by k. $\mathcal{A}$ is an approximation operator to vector valued data. "D?" checks equal-parameter distance between $g^k(t)$ and $c(r^{i^*}(t))$ where i^* denotes the final iteration of the first stage of the algorithm.

sufficiently converged to the desired parametrization. Consider the arc length algorithm of section 4.2 as one example. The pseudo metric for this algorithm insures uniform convergence to unit speed within a specified tolerance. A curve $g(t)$, however, can be within a Euclidean distance ϵ_2 of $a(t)$ everywhere, but have a speed arbitrarily far removed from unity at points in its domain.

5.4 Method 2 – An Alternative General Scheme

This section develops an alternative to the general scheme of section 4.1.1. The new method directly approximates $c(r(t))$ in spline spaces of any desired polynomial order and forms an alternative framework for the algorithms of Chapter 4 and sections 5.1 and 5.2. This new algorithm has advantages over the schemes of sections 4.1.1 and 5.3 in a number of situations, which will be discussed below.

Two observations motivate the development of this new method:

1. Method 1 acquires data to approximate $r(t)$ iteratively. The algorithms presented for the arc length and inverse function reparametrization cases use point-wise evaluation, or point-wise approximation, of $r^{-1}(u)$ to create tables indexed by values in the domain of $c(u)$:

$$\begin{array}{ccc} \underline{u \text{ domain}} & & \underline{\text{derived from } r^{-i}} \\ u_j^i & \xrightarrow{r^{-i}} & t_j^i,\ \frac{d}{dt} r^i(t_j^i) \end{array}$$

with $t_j^i = r^{-i}(u_j^i)$ and $\frac{d}{dt} r^i(t_j^i) = 1/\frac{d}{du} r^{-i}(u_j^i)$. (A slight abuse of notation appears here: r^{-i} denotes the ith approximation to the inverse function denoted as r^{-1}.)

This information is then used to form NURBS approximation $r^i(t) \approx r(t)$ and subsequently NURBS curve $c(r^i(t)) \approx c(r(t))$.

Extended versions of these tables could be created instead as shown in Figure 5.6. Here $c(r^i(t_j^i)) = c(u_j^i)$ and $\frac{d}{dt} c(r^i(t_j^i)) = \frac{d}{du} c(u_j^i) / \frac{d}{du} r^{-i}(u_j^i)$.

The "t domain" and "composed function" columns could be used to approximate

$$\begin{array}{ccccc} \underline{\text{composed function}} & & \underline{u \text{ domain}} & & \underline{t \text{ domain}} \\ c(r^i(t_j^i)),\ \frac{d}{dt} c(r^i(t_j^i)) & \xleftarrow{c} & u_j^i & \xrightarrow{r^{-i}} & t_j^i \end{array}$$

Figure 5.6. Extended data table for use with Method 2.

$c(r(t))$ **directly**. The problem with such a technique (as noted in section 4.1.1) is that the geometry of the approximating curve, in general, will no longer be identical to $c(u)$.

2. Finding a solution to this last problem can be approached as follows: given NURBS curve $g(t) \approx c(r(t))$ we wish to bound the Frechet distance between $g(t)$ and $c(u)$ by a user specified tolerance. This could be established by:

$$\sup_t \|g(t) - c(r(t))\| \leq \epsilon, \tag{5.3}$$

or equivalently:

$$\sup_t \left\| g(r^{-1}(u)) - c(u) \right\| \leq \epsilon. \tag{5.4}$$

provided that $r(t) : I_t \to I_u$ is a piecewise allowable change of parameter (and hence a homeomorphism).

The problem now becomes how to establish one of these inequalities given our assumption that $r(t)$ is **not** a NURBS (and hence we may not be able to establish bounds for the left hand sides of (5.3) and (5.4) directly). In cases where $r^{-1}(u)$ **is** a NURBS, such as in the reparametrization along linear axis algorithm, the techniques of section 2.2.3 allow us to use (5.4) directly. Otherwise useful bounds on the Frechet distance can be established with the aid of a convergent sequence of NURBS functions that approximate either $r(t)$ or $r^{-1}(u)$ in (5.3) or (5.4).

These two observations lead to a new method for approximating the reparametrization of NURBS curves.

Let $\mathcal{S}^i = \mathcal{S}_{l,\sigma^i}$ denote the ith approximating polynomial spline space given by ith knot vector σ^i and fixed order l. These spaces are used to formulate the spline approximations $r^{-i}(u) \approx r^{-1}(u)$.

Let $\mathcal{T}^i = \mathcal{T}_{m,\tau^i}$ denote the ith approximating polynomial, or rational, spline space given by ith knot vector τ^i and fixed order m. These spaces are used to formulate the spline approximations $g^i(t) \approx f(t) = c(r(t))$.

The ingredients for this new scheme are:

- An initial spline space, $\mathcal{T}^0$, in which to approximate $c(r(t))$.

- An approximation operator $\mathcal{A}[f, \mathcal{T}^i]$, which approximates its first argument by a NURBS in the space $\mathcal{T}^i$. Note that this operator is used to approximate **parametric** functions. Choices for this operator are discussed in section 5.4.1.

- A means of deriving a spline space $\mathcal{S}^i$ from space $\mathcal{T}^i$. Section 5.4.3 discusses the relationship between these spaces.

- A monotone approximation operator $\mathcal{M}[r^{-1}, \mathcal{S}^i]$, which approximates its first argument by a NURBS in the space $\mathcal{S}^i$. Note that this operator is used to approximate **scalar** functions. (See section 4.1.2.)

- A metric, or pseudo metric, $d(g^i, f)$, that measures the distance between the ith approximation g^i and $f = c(r(t))$ as in section 4.1.3.

- Refinement schemes **refine1**$(\tau^i, b^i(t))$ and **refine2**$(\tau^i, \beta^i(u))$. The scheme refine1 is similar to the refinement schemes of section 4.1.4. The scheme refine2 takes a function $\beta(u)$ with the same domain as $c(u)$. These schemes will be discussed in section 5.4.2.

The new scheme is as follows:

Input:

$c(u) : I_u \to \boldsymbol{R}^n$ a piecewise regular NURBS curve map,

$r(t) : I_t \to I_u$ a piecewise allowable change of parameter with $r(I_t) = I_u$,

m the order of approximating functions for $c(r(t))$,

l the order of approximating functions for $r^{-1}(u)$,

τ^0 the knot vector for the initial approximation of $c(r(t))$, and

$\epsilon_1, \epsilon_2 > 0$ error tolerance measures.

Output:

Approximations for $f(t) = c(r(t))$ or $r^{-1}(u)$.

Algorithm:

1. $i = 0$

2. $g^i(t) = \mathcal{A}[f(t), \mathcal{T}^i]$
3. find a NURBS function $b^i(t)$ which bounds $d(g^i, f(t))$.
4. if $\max_t b^i(t) > \epsilon_1$ then

   $\tau^{i+1} = \text{refine1}(\tau^i, b^i(t)), \quad i = i+1, \;$ go to 2.
5. derive $\mathcal{S}^i$ from $\mathcal{T}^i$.
6. $r^{-i}(u) = \mathcal{M}[r^{-1}, \mathcal{S}^i]$
7. form $q^i(u) = g^i(r^{-i}(u))$ as a NURBS curve.
8. find a NURBS function $\beta^i(u)$ which bounds $\|q^i(u) - c(u)\|$.
9. if $\max_u \beta^i(u) > \epsilon_2$ then

   $\tau^{i+1} = \text{refine2}(\tau^i, \beta^i(u)), \quad i = i+1, \;$ go to 2.
10. return $g^i(t)$ or $r^{-i}(u)$.

Figure 5.7 compares Methods 1 and 2. Method 1 uses function composition to guarantee that the geometry of the reparametrized curve does not change in the algorithm's first stage. Method 2 uses parametric approximation to generate approximations for $c(r(t))$. In general **both** the parametrization and shape of the curve will change. A test

$$\begin{tikzcd}
c(u) \arrow[r, "\mathcal{M}"] & r^i(t) \arrow[r] & c(r^i(t)) \arrow[r] & P? \arrow[lll, bend right, "i = i+1"'] \arrow[r, "\mathcal{A}"] & g^k(t) \arrow[r] & D? \arrow[l, bend right, "k = k+1"']
\end{tikzcd}$$

(a) Method 1

$$\begin{tikzcd}
c(u) \arrow[r, "\mathcal{M}"] & r^{-i}(u) \arrow[r, "\mathcal{A}"] & g^i(t) \arrow[r] & P? \arrow[r] & g^i(r^{-i}(u)) \arrow[r] & D? \arrow[llll, bend right, "i = i+1"']
\end{tikzcd}$$

(b) Method 2

Figure 5.7. Schematics for Methods 1 and 2. Here Method 1 has been augmented with the extension of section 5.3. $\mathcal{M}$ is a monotone approximation operator to scalar valued data whereas $\mathcal{A}$ is an approximation operator to vector valued data. "P?" checks for convergence to the correct parametrization. "D?" checks equal-parameter distance between $g^k(t)$ and $c(r^i(t))$ in Method 1 and between $g^i(r^{-i}(u))$ and $c(u)$ in Method 2.

is first performed to make sure that the approximation is close enough to the correct parametrization. Since the metric used in this test may not take shape into account, a subsequent test is used to ensure that the shape of the result is within a specified Frechet distance of the original curve $c(u)$.

Method 2 has advantages over Method 1 which include (refer to Figure 5.7):

- Decoupling the data complexity (order and number of control points) and smoothness of the result $g^i(t)$, from the complexity and smoothness of the NURBS function composition form, is done early, rather than late, in the algorithm.

- The formation of $r^{-i}(u)$ may be more straightforward than the formation of $r^i(t)$. (See, for example, section 5.1.)

- The results of Method 2 satisfy both parametrization and shape tolerances simultaneously. This is not necessarily the case with the results of the extended version of Method 1.

- Method 2 can have an advantage over Method 1 in handling cases where curve $c(u)$ is outside the allowed tolerance (for the desired parametrization) on only a relatively small interval of the domain. Use of Method 1 can cause a penalty in data complexity which affects the curve representation over its entire domain. Method 2, on the other hand, need only affect the representation on the interval in error.

- In Method 2, test "P?", for convergence to the correct parametrization, can be done on $g^i(t)$, typically of lower order than $c(r^i(t))$.

- The approximation $g^i(t)$ can be converted to the polynomial case, when $c(u)$ is rational, early in the algorithm.

- NURBS function composition and test "D?", for equal-parameter distance, may be performed less often in Method 2.

Method 1 may be more useful, however, if either an approximation to r is wanted, or if the polynomial order of the approximation for r is 2 (degree 1).

5.4.1 Approximation Operators $\mathcal{A}$

This section discusses the approximation operator $\mathcal{A}[f, \mathcal{T}^i]$ of Method 2. This operator is used to approximate parametric functions by NURBS curves in space $\mathcal{T}^i$.

As with the monotone approximation operators discussed in section 4.1.2, this operator will generally use discrete data resulting from point-wise evaluation, or point-wise approximation. Unlike the monotone operators of section 4.1.2 this operator will be used to approximate **parametric** functions.

The Schoenberg variation diminishing spline approximation, discussed in section 4.1.2 for use as a monotone approximation operator, can be used for approximation operator $\mathcal{A}$ as well.

The quasi-interpolant of deBoor and Fix [20] provides another choice for $\mathcal{A}$. It has optimal convergence properties but, for polynomial order m, requires function and derivative evaluation through order $m-1$. This scheme is useful, however, in cases where these derivatives are available. Other spline quasi-interpolant schemes trade-off more function evaluations for fewer, and lower order, derivative evaluations [50].

Approximation operator $\mathcal{A}$ could also employ interpolation to discrete parametric data. C^1 cubic Hermite spline interpolation, requiring both function and first derivative data, is an example of such a scheme. Many other interpolation schemes are possible. For introductions to techniques see [58] and [27].

As in the case of the monotone operators of section 4.1.2, it is important to lower the continuity of the approximation operator $\mathcal{A}$ when it is known a priori that function $c(r(t))$ is **not** C^1 everywhere.

5.4.2 Relationship Between Refine1 and Refine2

As with the refinement schemes of section 4.1.4 it may be more natural to view refine1 and refine2 as ways of generating a more refined or dense sampling of the function $f = c(r(t))$ to be approximated. This sampling can proceed from the selection of values in the domain of $c(u)$ as illustrated by Figure 5.6. Without loss of generality assume that $r(t)$ is a monotone increasing function. The table of Figure 5.6 may be sorted by increasing value of u_i resulting in entries that are also sorted by increasing value of t_i.

A very simple relationship exists between the two refinement schemes. Refine1 takes an error function, $b^i(t)$, in the domain of $c(r(t))$ whereas refine2 takes an error function, $\beta^i(u)$, in the domain of $c(u)$. For refine1 if $b(t)$ exceeds the specified tolerance for a data interval $[t_i, t_{i+1}]$ in the table of Figure 5.6, then the sampling in the next iteration should be increased in the interval $[u_i, u_{i+1}]$ (a new data point can be inserted at the midpoint of this interval for example). For refine2 if $\beta(u)$ exceeds the specified tolerance for a data interval $[u_i, u_{i+1}]$, then the sampling in the next iteration should be increased in this interval.

5.4.3 Relationship Between $\mathcal{S}^i$ and $\mathcal{T}^i$

Spline space $\mathcal{T}^i$ is used to approximate the parametric function $f = c(r(t))$ using approximation operator $\mathcal{A}$, whereas spline space $\mathcal{S}^i$ is used to approximate the scalar valued monotonic function $r^{-1}(u)$ using the monotone approximation operator $\mathcal{M}$.

In general there may be little relationship between these spline spaces. In practice, however, these spaces can be related by the data used for approximation and the particulars of the approximation schemes used for $\mathcal{M}$ and $\mathcal{A}$. Data from the same table can be used to approximate both r^{-1} and $c(r(t))$ for example. Referring to Figure 5.6, the data in the "u domain" and "t domain" columns are used to approximate r^{-1} and data from the "t domain" and "composed function" columns used to approximate $c(r(t))$. Spline space $\mathcal{T}^i$ can be generated from the set $\{t_i\}$, along with particulars of the approximation scheme $\mathcal{A}$, and consideration of known discontinuities of $c(r(t))$. Similarly, spline space $\mathcal{S}^i$ can be generated from the set $\{u_i\}$, particulars of the approximation scheme $\mathcal{M}$, and consideration of the known discontinuities of $r^{-1}(u)$.

5.5 Knot Refinement for Improved Error Bounds

The algorithms of this thesis make extensive use of the techniques of section 2.2.3 for bounding the length of NURBS functions; bounds on the length of a NURBS function, over an interval of its parametric domain, are based on the convex hull of the control points actually blended on that interval. Bounds on function length are then used as bounds on approximation errors in the algorithms. Functions used in this manner depend on the nature of the algorithm and include: derivatives of NURBS curves for the arc length algorithm of section 4.2, differences of scalar or vector valued NURBS functions for the inverse function and Frechet distance algorithms of sections 4.3 and 4.4, and the NURBS function bounded in equation 5.2 for the radial reparametrization algorithm of section 5.2.2.

Error overestimation can cause the algorithms to produce results with **true** errors smaller, and data complexity higher, than is actually necessary. Tighter error bounds can help alleviate this problem. These can be achieved by applying knot refinement (see section 2.2.1) to the NURBS function used to estimate error prior to bounding its length. This refinement can be accomplished as follows:

1. Insert knots to isolate error estimates to individual data intervals (intervals between parametric locations where data are collected). One way to do this is by inserting knots of multiplicity one less than the order $(o - 1)$ at the ends of these intervals in the NURBS function used for error estimation, where o is the polynomial order of this function. The multiple knots ensure that, for the NURBS function whose length is bounded, blending of control points over successive data intervals have only a single control point in common. Thus the effects of control points internal to the ends of a given interval can only affect that portion of the function **interior** to that interval.

2. Insert additional, evenly spaced, knots of multiplicity one into the **interior** of the data intervals to help tighten error bounds further. The current implementation allows the user to specify the number of additional knots per data interval. Using the minimum distance to convex hull techniques of [33] and [9] (see section 2.2.3), most of the examples run on the algorithms require only two or three additional knots in the interior of data intervals, with minimal, or no, further decrease in data complexity resulting from the use of more knots.

CHAPTER 6

RESULTS

This chapter gives example results from experiments with the algorithms of Chapters 4 and 5. Chapter 7 gives higher level applications of these algorithms.

6.1 Arc Length Examples

This section presents results from experiments with the arc length algorithm of section 4.2. Figure 6.1 shows a NURBS curve reparametrized by approximate arc length using this algorithm. Superimposed on the curve are points equally spaced in the curve's parametric domain. Figure 6.2 shows the quadratic change of parameter function used to reparametrize this curve. Note that the change of parameter function is only C^0 at a point corresponding to a speed discontinuity on the original G^1 curve. Figures 6.3 and 6.4 plot speed functions for this example.

Figure 6.5 shows a NURBS circular arc after application of the algorithm of section 4.2. Dots represent points equally spaced in the domain of the original parametrization, tick marks points equally spaced in the domain of the approximate arc length parametrization. Figures 6.6 and 6.7 plot speed functions for this example.

Tables 6.1 and 6.2 give detailed results for executions of the arc length algorithm for these examples. Headings for tables in this chapter are interpreted according to Figure 6.8.

6.2 Inverse Function Examples

This section presents examples using the inverse function algorithms of Chapters 4 and 5. Section 6.2.1 gives results for experiments with the approximation of inverse NURBS functions using the algorithm of section 4.3. Section 6.2.2 gives results for experiments with the reparametrization by axis algorithm of section 5.1.

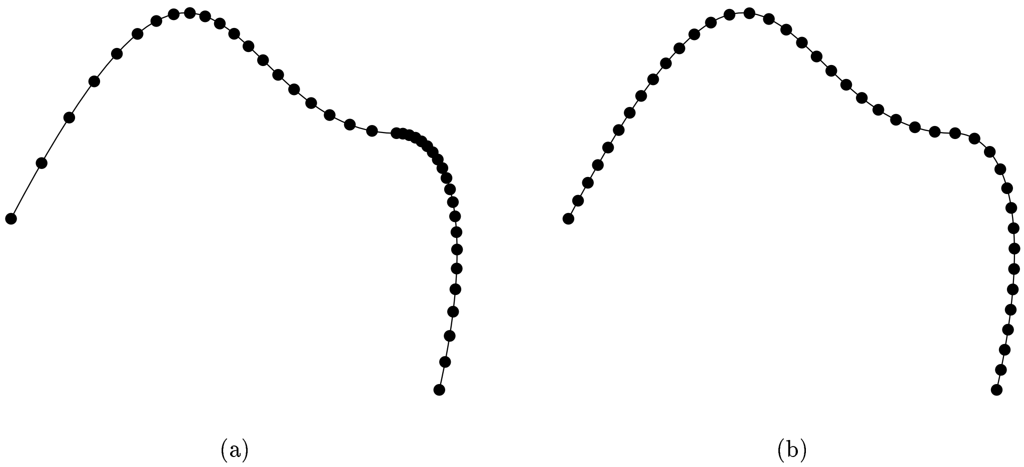


Figure 6.1. Reparametrization by arc length of a NURBS curve with a speed discontinuity: (a) original curve, (b) reparametrized curve. Dots represent equal spacing in the parametric domains.

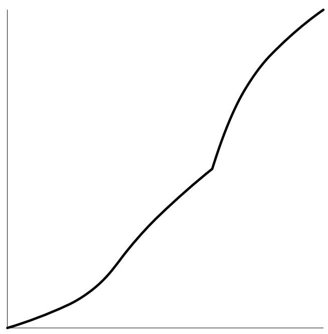
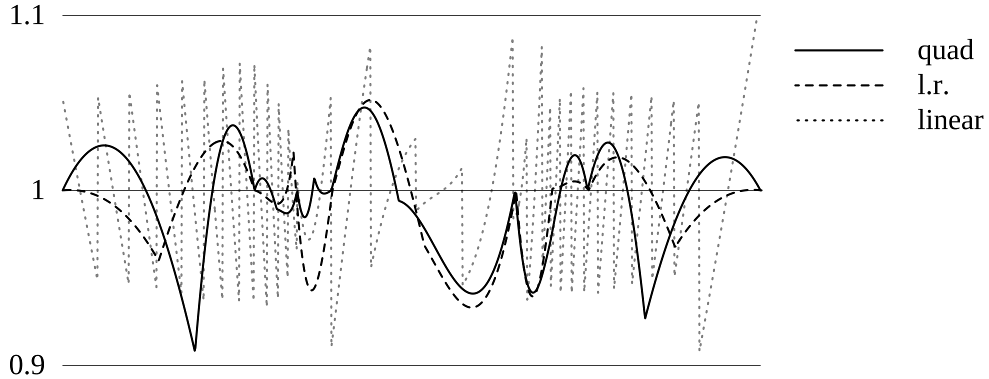


Figure 6.2. Quadratic spline change of parameter function used in the example of Figure 6.1.

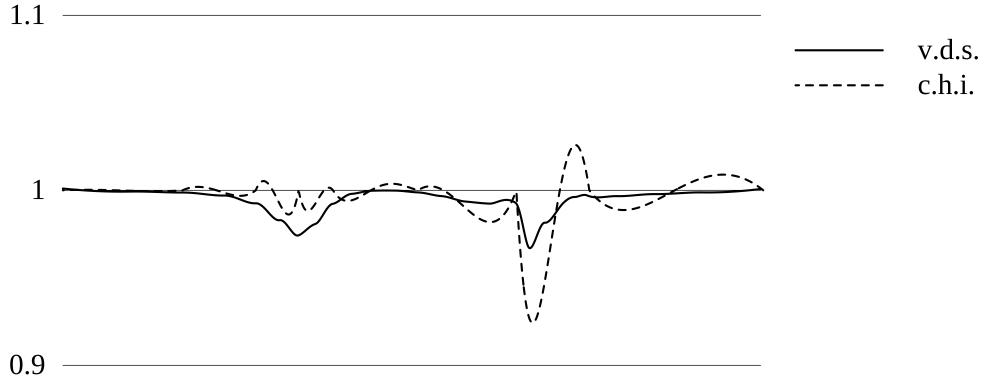
Table 6.1. Results for arc length example of Figure 6.1

method	ϵ_1	ϵ_2	type	iterations	entries	order[a]	size[a]	time
1	.1	-	linear	6	31	4	91	0.0183
1	.1	-	l.r.	3	6	4	31	0.0842
1	.1	-	quad	4	7	7	63	0.0254
2	.1	.1	v.d.s.	4,3	26	4	28	0.1051
2	.1	.1	c.h.i.	3,2	9	4	19	0.0270
2	.05	.05	v.d.s.	5,3	33	4	35	0.1349
2	.05	.05	c.h.i.	4,2	11	4	23	0.0339

[a]Order and size for reparametrized curve. Original curve order: 4, original curve size: 11.

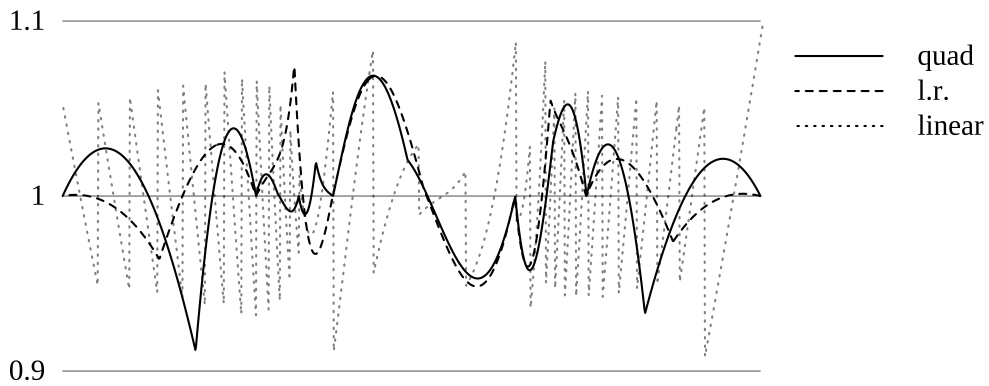


(a)

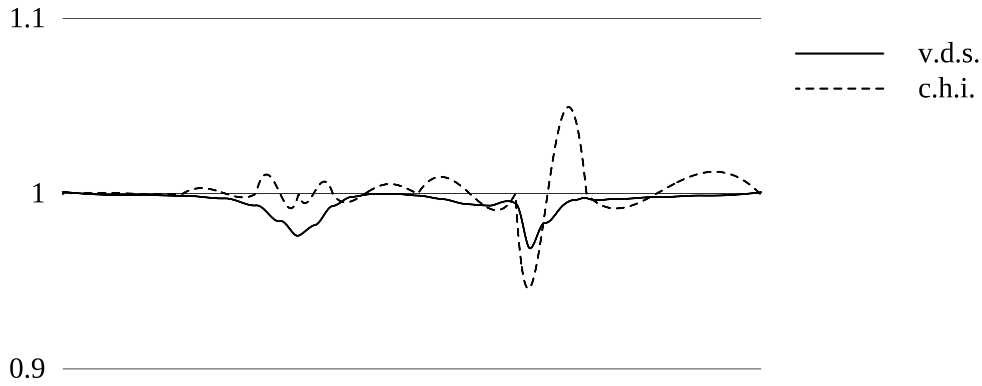


(b)

Figure 6.3. Graphs of speed functions for the example of Figure 6.1. Plots in (a) show errors for applications of Method 1 using piecewise linear, C^1 piecewise linear rational, and C^1 piecewise quadratic interpolation schemes. A tolerance of 0.1 was specified to the algorithm. Plots in (b) show errors for applications of Method 2 using variation diminishing spline approximation and cubic Hermite interpolation. Tolerance values ϵ_1 and ϵ_2 where specified as 0.1 to the algorithm. Plots in (a) and (b) result from estimation of arc length from **above** (see Appendix C). All graphs are for runs of the algorithms using additional knot refinement as discussed in section 5.5.



(a)



(b)

Figure 6.4. Graphs of speed functions for the example of Figure 6.1. Errors in (a) and (b) result from executions of the algorithms identical to those used in Figure 6.3 except that estimation of arc length is made from **below** (see Appendix C).

Table 6.2. Results for arc length example of Figure 6.5

method	ϵ_1	ϵ_2	type	iterations	entries	order[a]	size[a]	time
1	.1	-	linear	4	7	3	13	0.0412
1	.1	-	l.r.	2	3	3	9	0.0232
1	.1	-	quad	2	3	5	14	0.0467
2	.1	.1	v.d.s.	4,1	7	3	8	0.1177
2	.1	.1	c.h.i.	2,1	3	4	6	0.0383
2	.05	.05	v.d.s.	5,1	13	3	14	0.2259
2	.05	.05	c.h.i.	2,1	3	4	6	0.0375

[a]Order and size for reparametrized curve. Original curve order: 3, original curve size: 3.

6.2.1 Inverse Function Approximation

Figure 6.9 shows the graph of a scalar valued NURBS function. The algorithm of section 4.3 was used to approximate the inverse function, also shown in Figure 6.9. Error functions for this example are plotted in Figure 6.10 using the metric of equation (4.1). Table 6.3 gives detailed results for executions of the algorithm for this example.

6.2.2 Reparametrization by Axis

Figure 6.11(a) shows the result of a ruled surface construction. The curves' parametrizations lead to poor correspondences which result in surface degeneracies. Figure 6.11(b) shows the ruled surface that results after the curves are reparametrized along the X axis using the algorithm of section 5.1.

Error functions for this example are plotted in Figure 6.12, where error is measured as distance of the X coordinate function from the identity; i.e., error metric (4.1).

Table 6.4 gives data from several runs of the algorithm of section 5.1 for this example.

6.3 Frechet Distance Examples

This section gives examples using the Frechet distance algorithms of Chapters 4 and 5. Section 6.3.1 gives results from experiments with the algorithm of section 4.4 for bounding the Frechet distance between NURBS curves. Section 6.3.2 gives examples of the use of the radial reparametrization algorithm of section 5.2.2. Finally, section 6.3.3 gives results from experiments with the algorithm of section 5.2.1 for finding contact intervals between NURBS curves.

6.3.1 Frechet Distance Algorithm

Figure 6.13 shows two NURBS curves that result from using Method 2 to approximate reparametrizations of a third NURBS curve. Reparametrization by arc length was used for one curve, reparametrization along the X axis used for the other. The algorithm of section 4.4 was used to establish different tolerance values as bounds on the Frechet distance between these two curves.

Figures 6.13(b) and (c) show closeups of the curves indicating their relative parametrizations before and after the bounds were established. Table 6.5 gives detailed results for running the algorithm on this example.

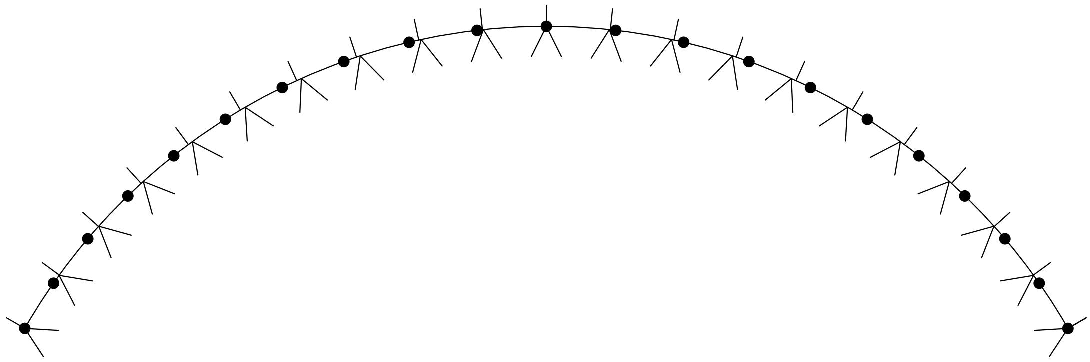


Figure 6.5. Reparametrization by arc length of a NURBS curve consisting of a 120 degree arc of unit radius. The curve was reparametrized by Method 1 using the piecewise quadratic interpolation scheme with a tolerance of 0.1. Dots indicate equal spacing in the parametric domain of the original curve. Tick marks ("|") indicate equal spacing in the parametric domain of the reparametrized curve. Arrows ("$\wedge$") indicate equal spacing in the true arc length parametrization.

Table 6.3. Results for the inverse function approximation example of Figure 6.9

method	ϵ_1	type	iterations	entries	order[a]	size[a]	time
1	.5	linear	1	2	2	2	0.00074
1	.5	l.r.	1	2	2	3	0.00673
1	.5	quad	1	2	3	4	0.00207
1	.1	linear	3	5	2	5	0.00301
1	.1	l.r.	3	4	2	7	0.03446
1	.1	quad	2	3	3	6	0.00548
1	.05	linear	4	7	2	7	0.00433
1	.05	l.r.	3	4	2	7	0.03416
1	.05	quad	2	3	3	6	0.00552
1	.01	linear	5	14	2	14	0.00682
1	.01	l.r.	3	5	2	9	0.03926
1	.01	quad	3	5	3	10	0.01159

[a]Order and size for inverse function approximation. Original curve order: 4, original curve size: 4.

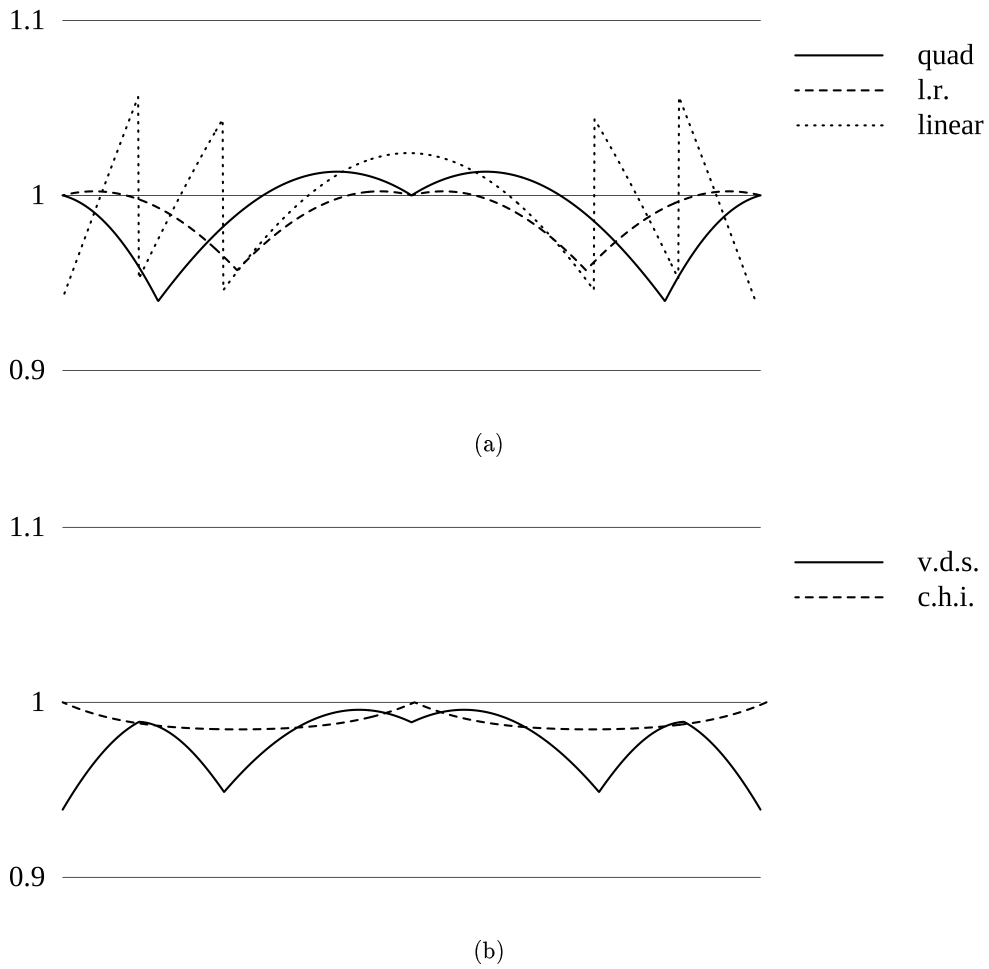


Figure 6.6. Graphs of speed functions for the example of Figure 6.5. Plots in (a) show errors for applications of Method 1 using piecewise linear, C^1 piecewise linear rational, and C^1 piecewise quadratic interpolation schemes. A tolerance of 0.1 was specified to the algorithm. Plots in (b) show errors for applications of Method 2 using variation diminishing spline approximation and cubic Hermite interpolation. Tolerance values ϵ_1 and ϵ_2 where specified as 0.1 to the algorithm. Plots in (a) and (b) result from estimation of arc length from **above** (see Appendix C). All graphs are for runs of the algorithms using additional knot refinement as discussed in section 5.5.

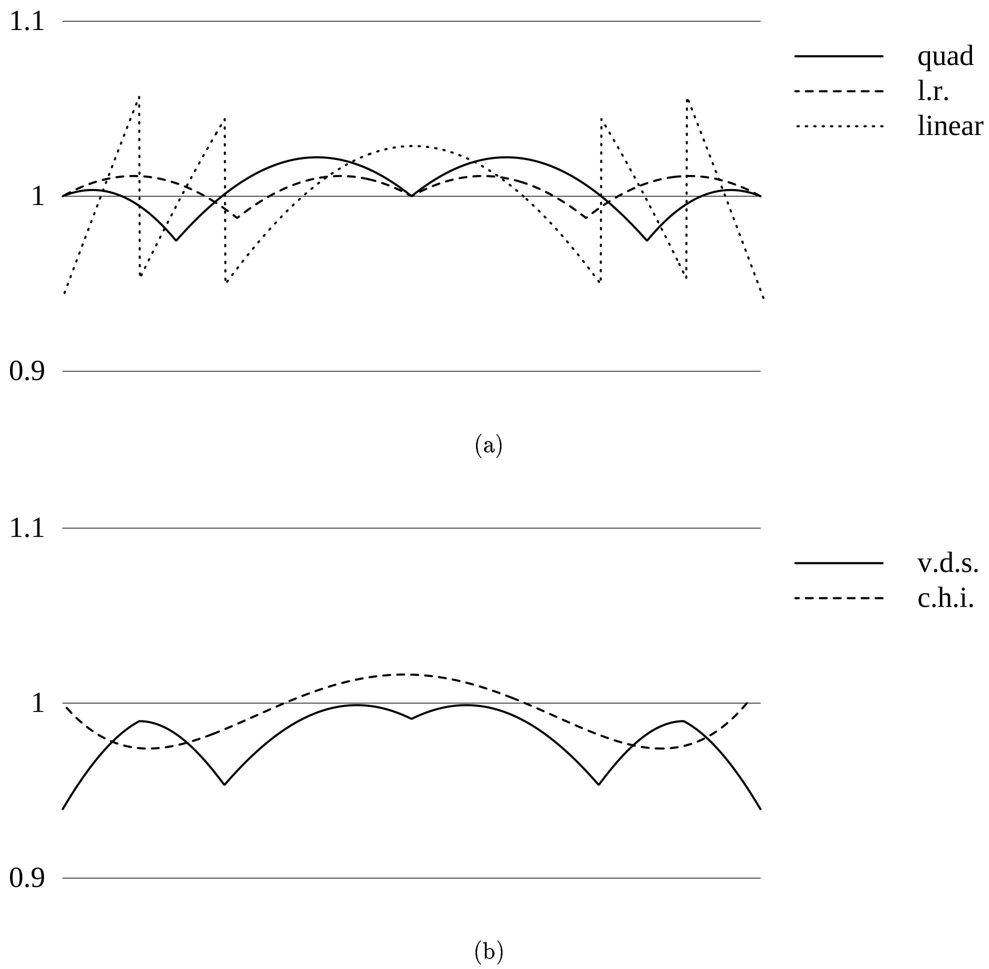


Figure 6.7. Graphs of speed functions for the example of Figure 6.5. Errors in (a) and (b) result from executions of the algorithms identical to those used in Figure 6.6 except that estimation of arc length is made from **below** (see Appendix C).

Table 6.4. Results for reparametrization by X axis example of Figure 6.11

method	ϵ_1[a]	ϵ_2	type	iterations	entries	order[b]	size[b]	time
1	.01	-	linear	3	14	4	40	0.0059
1	.01	-	l.r.	1	7	4	37	0.0327
1	.01	-	quad	1	7	7	62	0.0092
2	.01	.01	v.d.s.	1,4	36	4	38	0.1280
2	.01	.01	c.h.i.	1,2	11	4	22	0.0434
2	.005	.005	v.d.s.	1,5	53	4	55	0.2119
2	.005	.005	c.h.i.	1,2	13	4	26	0.0460

[a]Error metric given by equation (4.1).

[b]Order and size for reparametrized curve. Original curve order: 4, original curve size: 13.

method The general reparametrization scheme used:

- **1** Method 1 of section 4.1.1 (without the extension of section 5.3).
- **2** Method 2 of section 5.4.

ϵ_1 Tolerance for Method 1 or for the parametrization metric of Method 2.

ϵ_2 Euclidean space tolerance controlling the second stage of Method 2.

type Type of interpolation or approximation scheme used:

- **linear** Piecewise linear interpolation for $\mathcal{M}$ of Method 1.
- **l.r.** C^1 piecewise linear rational interpolation for $\mathcal{M}$ of Method 1.
- **quad** C^1 piecewise quadratic interpolation for $\mathcal{M}$ of Method 1.
- **v.d.s.** $C^{\text{order}-1}$ variation diminishing spline approximation for $\mathcal{A}$ of Method 2 and C^1 piecewise quadratic interpolation for $\mathcal{M}$ of Method 2.
- **c.h.i.** C^1 cubic Hermite spline interpolation for $\mathcal{A}$ of Method 2 and C^1 piecewise quadratic interpolation for $\mathcal{M}$ of Method 2.

deriv type Type of derivative estimation for the Frechet distance algorithm.

- **1** Projection method of section 4.4.4.
- **2** Linear rational fitting method of section 4.4.4.

iterations Number of iterations for the algorithm. Results for Method 2 give comma separated counts for both the first and second stages of the algorithm respectively.

entries Number of evaluation points used by the algorithm.

order Polynomial order of the result.

size Number of control points in the result.

time Execution time in seconds on an SGI Indigo 2, MIPS R10K, 195 MHZ, 64 kilobyte level one cache, 1 megabyte level two cache.

Figure 6.8. Key to table headings.

Table 6.5. Results for Frechet distance example of Figure 6.13

method	ϵ_1	type	deriv type[a]	iterations	entries	order[b]	size[b]	time
1	.023	linear	-	2	21	2	21	0.0593
1	.023	l.r.	1	2	21	2	41	0.2822
1	.023	quad	1	2	21	3	42	0.0935
1	.023	l.r.	2	2	21	2	41	0.2832
1	.023	quad	2	2	21	3	42	0.0911

[a] Type of derivative estimation: 1 projection, 2 fitting.

[b] Order and size for the change of parameter function used. The original curves of Figure 6.13 are order 4 and size 20.

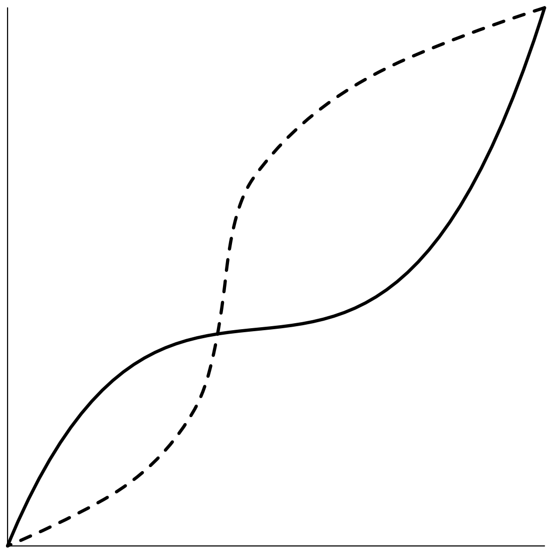


Figure 6.9. Graph of a scalar valued NURBS function and approximate inverse. The inverse function approximation (shown as a dotted line) was created by the algorithm of section 4.3 using the C^1 piecewise linear rational interpolation scheme with a tolerance of 0.01 measured according to the metric of equation (4.1).

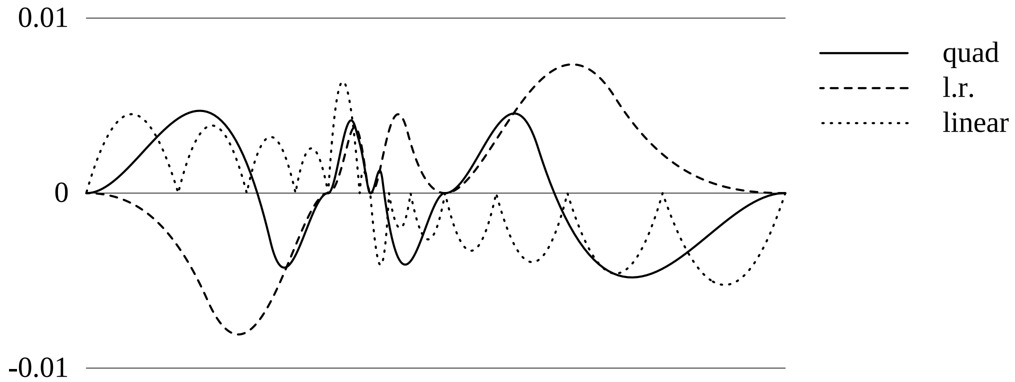


Figure 6.10. Error functions for approximations to the inverse of the function in Figure 6.9.

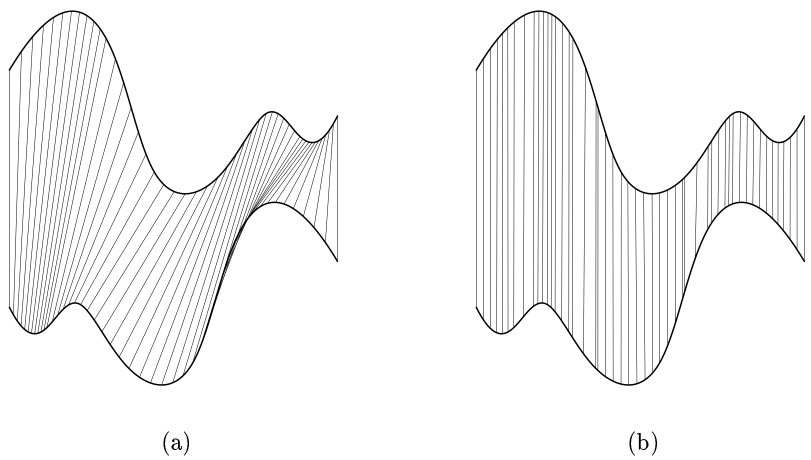


Figure 6.11. Example using the reparametrization by X axis algorithm of section 5.1. (a) Ruled surface construction using curves with poor parametric correspondence. (b) Surface generated after reparametrization of the curves along the X axis. The extent of these curves along the X axis is 2.15 units.

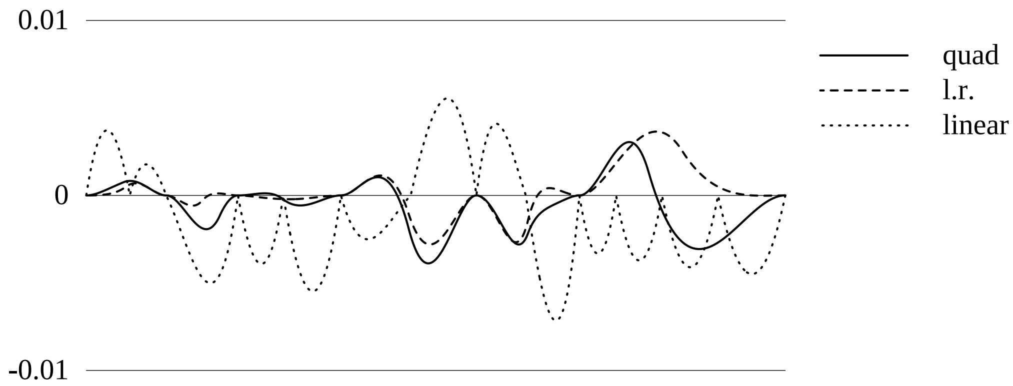


Figure 6.12. Error functions for the bottom curve of Figure 6.11 after reparametrization along the X axis using the algorithm of section 5.1. Errors are shown for the use of Method 1 with piecewise linear, C^1 piecewise linear rational, and C^1 piecewise quadratic interpolation schemes. An error tolerance of 0.01 was used for these approximations, measured according to the metric of equation (4.1).

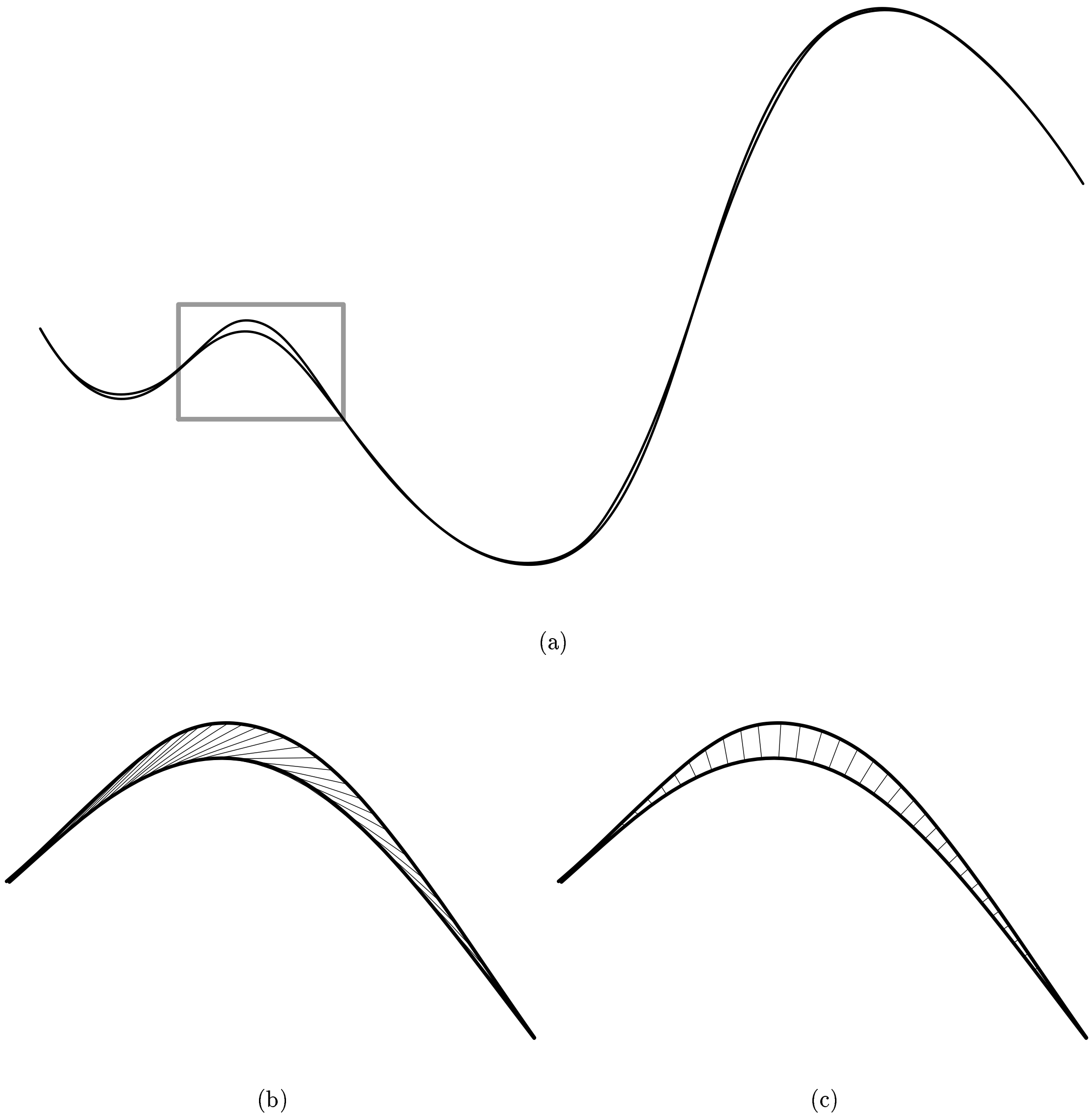


Figure 6.13. Example using the Frechet distance algorithm of section 4.4. (a) Two approximations to the same curve but with different parametrizations. Rectangle indicates area enlarged in (b) and (c). (b) Correspondence in the original parametrizations. (c) Correspondence after reparametrization to establish a user defined tolerance on Frechet distance.

6.3.2 Radial Reparametrization

Figure 6.14 shows examples of the use of the radial reparametrization algorithm of section 5.2.2. Error plots are shown in Figure 6.15. Both tolerance values and error are measured in radial distance which, as defined in section 5.2.2, is the central angle at the origin between corresponding points on the two curves measured in degrees. Figure 6.16 plots the reparametrization function used in the example of Figure 6.14(f). Figure 6.17 shows the effects of applying different methods to estimate derivatives for the change of parameter function. In this example, the linear rational fitting technique of section 4.4.4 yields better results than are obtained using the projection method, also described in section 4.4.4.

Detailed results for these examples are given in Table 6.6.

6.3.3 Contact Intervals

Figures 6.18 through 6.24 show examples using the algorithm of section 5.2.1 for finding contact intervals between NURBS curves. Figure 6.18 shows two curves with a single contact interval. The resulting description of the contact interval is given in Figure 6.19. Figure 6.20 shows two curves with three contact intervals. The straight line segment at the bottom of this figure has a cubic, nonlinear, parametrization which is shown in Figure 6.21. The straight line segments of the other curve are linearly parametrized. Figure 6.22 gives the description of the contact intervals for this example. The curves are opposed on one of the intervals and move in the same sense in the other two. Figure 6.23 shows two circles that have been rotated relative to one another. Because of the seams in each curve there are two contact intervals, described in Figure 6.24.

Detailed results for the above examples are given in Table 6.7.

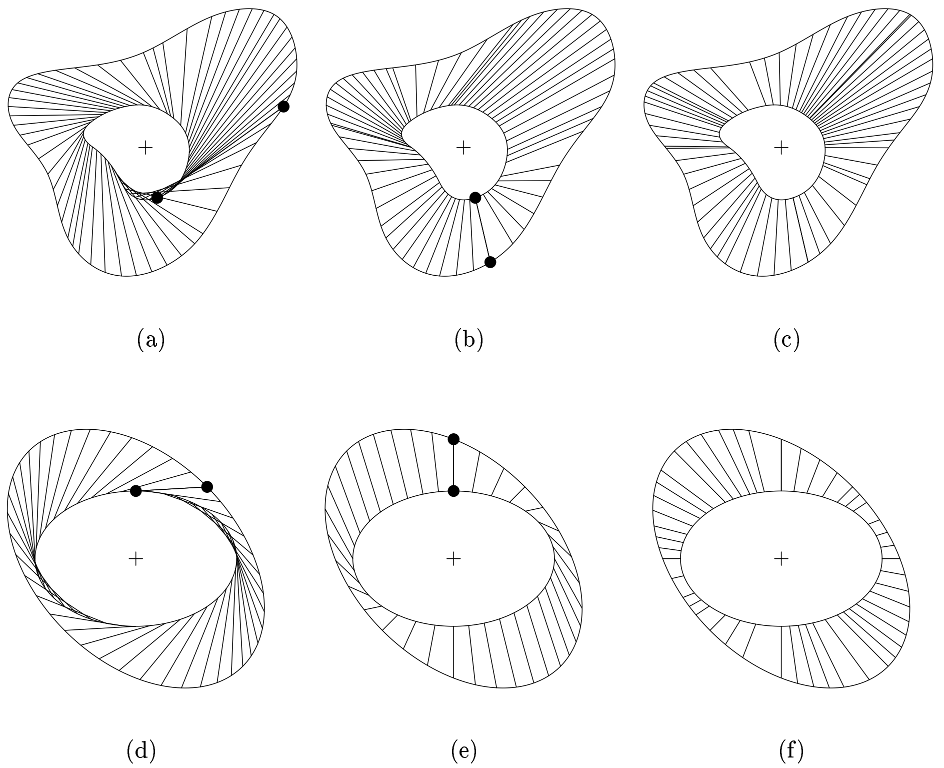


Figure 6.14. Two examples of the use of the radial reparametrization algorithm of section 5.2.2. Figures (a) and (d) show the parametric correspondence between the original curves. Dots indicate the start/end points on the curves. Crosses ("+") indicate the origin used for radial correspondence. Figures (b) and (e) show the parametric correspondence between the curves once start/end points of the curves have been aligned radially. This is accomplished in each case by a single ray/curve intersection operation to radially project the start/end points of the inner curves onto the outer curves. This is followed by subdivision of the outer curves at the points of projection. Figures (c) and (f) show the results of running the radial reparametrization algorithm on the curves in (b) and (e) respectively.

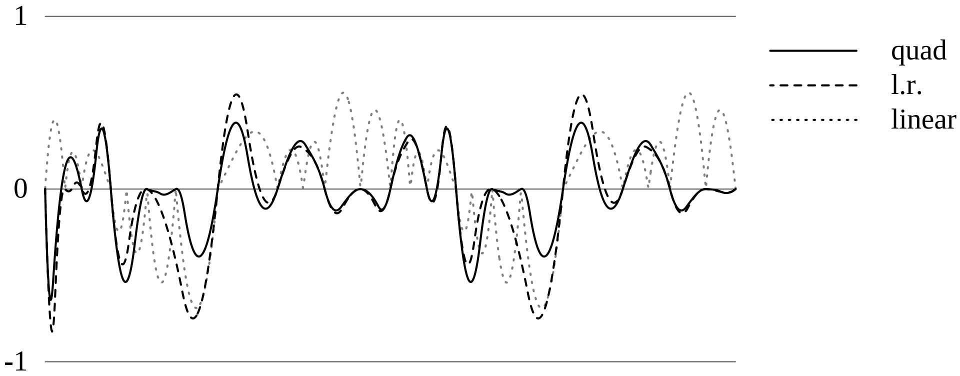


Figure 6.15. Error functions for the example of Figure 6.14(f). Error functions are shown for the use of Method 1 with piecewise linear, C^1 piecewise linear rational, and C^1 piecewise quadratic interpolation schemes. An error tolerance of 1.0 degree was used for the approximations. The linear rational fitting technique of section 4.4.4 was used for derivative estimation.

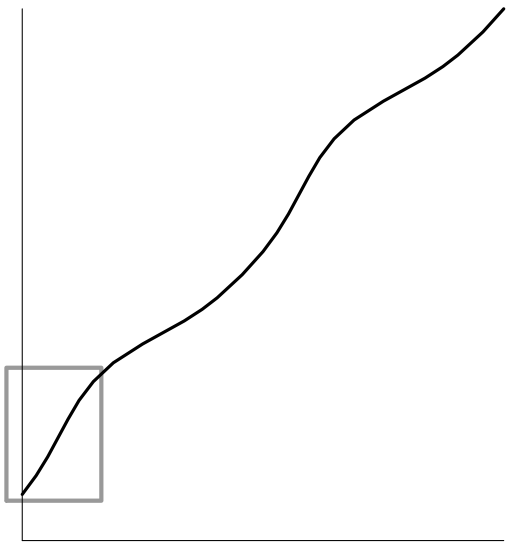


Figure 6.16. Change of parameter function used in reparametrizing the outer curve of Figure 6.14(e) into the outer curve of Figure 6.14(f). Rectangle indicates area enlarged in Figure 6.17.

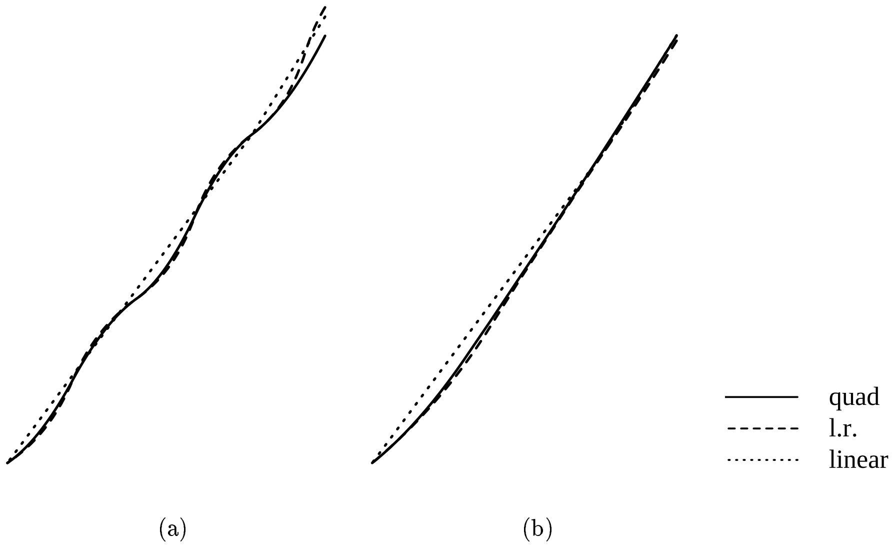


Figure 6.17. Closeup of initial portions of the change of parameter functions used to reparametrize the outer curve of Figure 6.14(e) into the outer curve of Figure 6.14(f). Shown are piecewise linear, C^1 piecewise linear rational, and C^1 piecewise quadratic change of parameter functions. (a) Shows the results of using the projection method of section 4.4.4 to estimate derivatives for the change of parameter functions in the linear rational and quadratic cases. (b) Better results are obtained, for this example, by using the linear rational fitting technique, also discussed in section 4.4.4.

Table 6.6. Results for radial reparametrization example of Figure 6.14(f)

method	ϵ_1[a]	type	deriv type[b]	iterations	entries	order[c]	size[c]	time
1	1.0	linear	-	3	25	3	49	0.197
1	1.0	l.r.	1	4	31	3	121	1.228
1	1.0	quad	1	4	29	5	174	1.979
1	1.0	l.r.	2	3	18	3	69	0.556
1	1.0	quad	2	3	19	5	114	1.032

[a]Error tolerance in degrees.
[b]Type of derivative estimation: 1 projection, 2 fitting.
[c]Order and size for the reparametrized curve. Original curve order: 3, original curve size: 14.

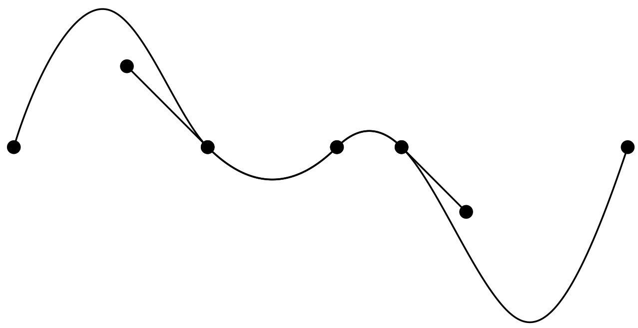


Figure 6.18. Quadratic and cubic NURBS curves with a single contact interval. Dots indicate the locations of signal values on the curves.

```
contact interval 1:
  curve 1 int = [1.000000, 3.000000]
  curve 2 int = [2.000000, 4.000000]
  opposed = FALSE
  approximation = FALSE
  reparam function = <scalar curve object: ...>
```

Figure 6.19. The description of the contact interval for the curves of Figure 6.18. This description consists of: the parametric interval on curve 1 and the corresponding interval on curve 2, a flag indicating that the curves in this example are traced in the same direction on the contact interval, the change of parameter function that relates the interval on curve 2 to the interval on curve 1, and a flag indicating that the change of parameter in this example is an exact representation. The printed representation of the change of parameter function has been truncated in the description above.

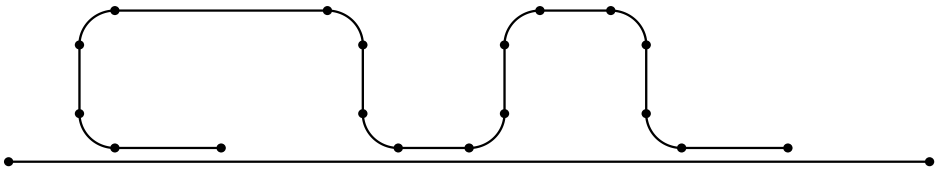


Figure 6.20. Curves with three contact intervals. In one interval the curves are traced in opposed directions. The straight line segment at the bottom is represented as a cubic NURBS with nonlinear parametrization. The curves have been offset slightly in this figure; they intersect in the actual example. Dots indicate the locations of signal values on the curves.

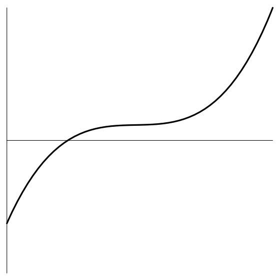


Figure 6.21. The X coordinate function for the straight line curve of Figure 6.20.

```
contact interval 1:
  curve 1 int = [0.000000, 1.000000]
  curve 2 int = [0.045818, 0.103739]
  opposed = TRUE
  approximation = TRUE
  reparam function = <scalar curve object: ...>

contact interval 2:
  curve 1 int = [8.000000, 9.000000]
  curve 2 int = [0.291601, 0.699110]
  opposed = FALSE
  approximation = TRUE
  reparam function = <scalar curve object: ...>

contact interval 3:
  curve 1 int = [16.000000, 17.000000]
  curve 2 int = [0.893092, 0.944999]
  opposed = FALSE
  approximation = TRUE
  reparam function = <scalar curve object: ...>
```

Figure 6.22. The description of the contact intervals for the curves of Figure 6.20. Note that the description for the first contact interval flags the curves as having opposed directions.

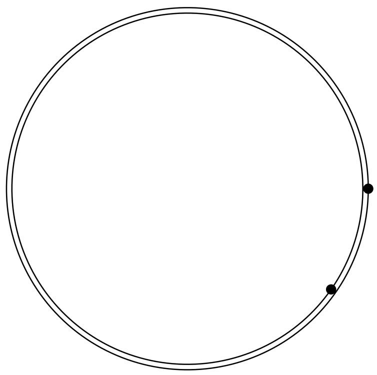


Figure 6.23. Two circles which have been rotated relative to one another. The dots represent the start/end points of each circle. The circles have been offset slightly in this figure; they are coincident in the actual example. Results for this case, given below, were obtained using the general algorithm of section 5.2.1.1 and not the extension of section 5.2.1.2.

```
contact interval 1:
  curve 1 int = [0.000000, 3.605610]
  curve 2 int = [0.394392, 4.000000]
  opposed = FALSE
  approximation = TRUE
  reparam function = <scalar curve object: ...>

contact interval 2:
  curve 1 int = [3.605610, 4.000000]
  curve 2 int = [0.000000, 0.394392]
  opposed = FALSE
  approximation = TRUE
  reparam function = <scalar curve object: ...>
```

Figure 6.24. The description of the contact intervals for the curves of Figure 6.23. Two contact intervals are found because of the "seam" (at the start/end points) in each curve.

Table 6.7. Results for contact interval examples

example	method	ϵ_1	type	time
fig 6.18	1	.05	linear	0.0090
fig 6.18	1	.05	l.r.	0.0095
fig 6.18	1	.05	quad	0.0096
fig 6.18	1	.01	linear	0.0109
fig 6.18	1	.01	l.r.	0.0106
fig 6.18	1	.01	quad	0.0095
fig 6.20	1	.05	linear	0.0410
fig 6.20	1	.05	l.r.	0.0484
fig 6.20	1	.05	quad	0.0352
fig 6.20	1	.01	linear	0.0565
fig 6.20	1	.01	l.r.	0.1393
fig 6.20	1	.01	quad	0.0694
fig 6.23	1	.05	linear	0.1691
fig 6.23	1	.05	l.r.	0.1999
fig 6.23	1	.05	quad	0.1994
fig 6.23	1	.01	linear	0.4548
fig 6.23	1	.01	l.r.	0.4006
fig 6.23	1	.01	quad	0.4662

CHAPTER 7

APPLICATION EXAMPLES

This chapter presents some example applications of the algorithms of Chapters 4 and 5. Section 7.1 uses the contact intervals algorithm of section 5.2.1 to determine topological information at surface boundary edges. Section 7.2 uses the reparametrization by axis algorithm of section 5.1 to reconstruct three dimensional curves from two dimensional orthographic views. Section 7.3 discusses applications of the arc length reparametrization algorithm of section 4.2 to achieve uniform spacing, or density, of features with respect to arc length. Finally, section 7.4 gives further uses for the approximation operators of section 4.1.2 for establishing correspondences between curves.

7.1 Automatic Topology Declarations

This section discusses an algorithm that uses the curve/curve correspondence information found by the algorithm of section 5.2.1 to automatically determine all surface boundary edge adjacencies within a group of surfaces.

One use for such an algorithm is to automatically determine topology information when forming the boundary representation of solids in a modeling system. Surface/surface intersection and Boolean operations in CAD systems may require that adjacencies be declared between the separate surfaces that make up the boundary of a solid and that share some portion of their surface boundary edges. This adjacency information may require the location and extent of the shared common boundary, as well as the change of parameter function that allows mapping from one parametric domain into the other when crossing the shared boundary.

Below is an algorithm that determines edge adjacency information between surfaces. The restriction of section 5.2.1 regarding where contact intervals may begin and end is assumed by this algorithm (see section 5.2.1 for details).

The basic structure of the `find-adjacencies` routine below is an $\mathcal{O}(n^2)$ algorithm that makes surface edge against surface edge comparisons. The boundary edges are defined in the parametric domain of each surface. For tensor product surfaces these boundary edges exist as isoparametric curves in a rectangular parametric domain and can be denoted as *left*, *right*, *top*, and *bottom* with respect to that domain.

The input to the algorithm is a list of surfaces needing edge adjacency declarations. This list of surfaces is assumed to be preprocessed as follows: Each surface in the list will have an associated list of edge objects. Each edge object contains a list of edge intervals that have **not** yet been matched (declared adjacent) with any other edge interval. Each edge interval consists of the geometric curve representing that portion of the edge not yet matched and a back pointer to the parent edge object. Initially each edge list contains a single interval consisting of the entire boundary edge.

For simplicity of presentation the contact interval information and the tolerance `tol` are assumed to be global rather than passed as arguments.

```
void
find-adjacencies( surface_list )
{
  initialize the global contact interval information;

  foreach srf1 in surface_list
  | foreach srf2 in surface_list
  | | if not disjoint( srf1, srf2 )
  | | | foreach edge1 in srf1 edges
  | | | | foreach edge2 in srf2 edges
  | | | | | /* Don't compare an edge of a surface with itself. */
  | | | | | if not ( edge1 == edge2 OR disjoint( edge1, edge2 ) )
  | | | | | {
  | | | | |   edge2_list = list of unmatched intervals in edge2;
  | | | | |   foreach edge1_int in list of unmatched intervals in edge1
  | | | | |     match-edge-int-vs-edge-int-list( edge1_int, edge2_list );
  - - - - - }

  return the global contact interval information;
}
```

The `match-edge-int-vs-edge-int-list` routine, called above, matches an edge interval against a list of edge intervals. The actual curve/curve interval comparisons are made by the `match-edge-int-vs-edge-int` routine. This second routine is described first.

The `match-edge-int-vs-edge-int` routine finds all contact intervals between two edge intervals using the routine `find-contact-intervals` which in turn uses the algorithm of section 5.2.1.1 to return the contact intervals between the two edge intervals.

In general there may be any number of contact intervals between two edge intervals. This may result in the "fragmentation" of both edge intervals. That is, each edge interval may fragment into any number of subintervals that remain unmatched. If a match occurs the `match-edge-int-vs-edge-int` routine places contact interval information on the global list and prepares output arguments which contain lists of the remaining unmatched subintervals (fragments) on each of its edge interval arguments.

```
boolean
match-edge-int-vs-edge-int( edge1_int,
                            edge2_int,
                            output edge1_fragments,
                            output edge2_fragments )
{
  /* Initialize to indicate no fragments. */
  edge1_fragments = NULL;
  edge2_fragments = NULL;

  if disjoint( edge1_int, edge2_int ) return FALSE;

  contacts = find-contact-intervals( edge1_int, edge2_int, tol );
  if ( contacts == NULL ) return FALSE;

  add information on contacts to global list;
  edge1_fragments = unmatched subintervals remaining on edge1_int;
  edge2_fragments = unmatched subintervals remaining on edge2_int;
  return TRUE;
}
```

The conceptual complexity of the `match-edge-int-vs-edge-int-list` routine, described below, stems from the management of the edge1 and edge2 subinterval fragments. These new intervals must be managed so that all necessary comparisons are made while redundant comparisons are avoided. The implementation described by the pseudo-code below is recursive and thus uses the program stack to help with the appropriate bookkeeping.

```
void
match-edge-int-vs-edge-int-list( edge1_int, edge2_list )
{
  if disjoint( edge1_int, edge2_list ) return;
  edge2_int = first of edge2_list;
  got_match = match-edge-int-vs-edge-int( edge1_int,
                                          edge2_int,
                                          edge1_fragments,
                                          edge2_fragments );
  if not ( got_match )
  {
```

```
    /* Argument edge1_int might be a new fragment. */
    if ( rest of edge2_list == NULL )
      add edge1_int to edge1_list if not already an element;
    else
      match-edge-int-vs-edge-int-list( edge1_int, rest of edge2_list );
  }
  else
  {
    remove edge1_int from parent edge interval list;
    replace edge2_int with edge2_fragments in parent edge interval list;

    if ( rest of edge2_list == NULL )
      add edge1_fragments to parent edge interval list of edge1_int;
    else
      edge1_int = foreach interval in edge1_fragments
        match-edge-int-vs-edge-int-list( edge1_int, rest of edge2_list );
  }
}
```

Figures 7.1 and 7.2 show an example of the use of this algorithm. A cylinder is formed from eight separate ruled surfaces: four ruled surface panels each with a separate surface forming a cut-out at the panel tops. The cylinder is capped at the top with a *boolean sum* surface (or bilinearly blended Coons patch [17]) formed from four 90 degree arc edges. This cap is rotated relative to the side panel seams by an arbitrary angle of 27 degrees. Figure 7.2 depicts the adjacencies found for this collection of surfaces.

7.2 Reconstructing 3D Curves From 2D Views

This section applies the reparametrization by axis algorithm of section 5.1 to the reconstruction of 3D curves from 2D orthographic views. This technique is useful for recovering 3D curves from scanned part drawings or blueprints. This technique may also be useful as a design paradigm for 3D curves.

Assume that the separate views are orthographic projections of a curve onto the XY and XZ planes. Denote these separate curves as $\gamma^{XY}(u)$ and $\gamma^{XZ}(v)$ respectively. **No** parametric correspondence between these curves is assumed (which would be the case if their parametric representations resulted from curve fitting of scanned data). The curves are both reparametrized along the X axis, resulting in $\hat{\gamma}^{XY}(t)$ and $\hat{\gamma}^{XZ}(t)$. The 3D curve is now reconstructed as:

$$\alpha(t) = (\hat{\gamma}_x^{XY}(t), \hat{\gamma}_y^{XY}(t), \hat{\gamma}_z^{XZ}(t))$$

where subscripts denote coordinate functions.

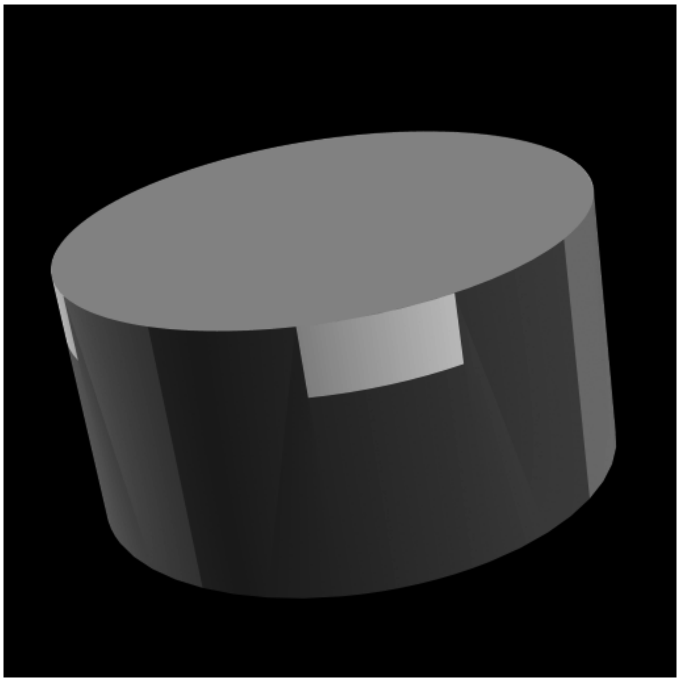


Figure 7.1. A capped cylinder made up of nine separate surfaces.

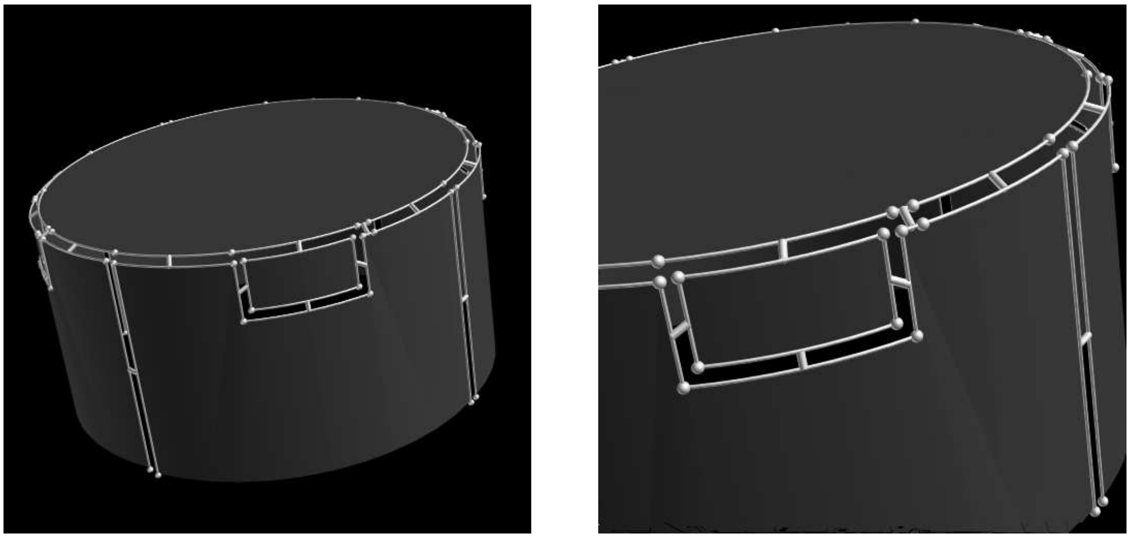


Figure 7.2. Topological information found by the `find-adjacencies` routine. Edge adjacency information has been determined along the highlighted edges. Different edge adjacencies are declared for the edge pieces demarcated by the balls on these edges. Links are shown between adjacent edge pieces of separate surfaces.

This technique has limitations. We must assume that the X coordinate functions for the two curves, $\gamma_x^{XY}(u)$ and $\gamma_x^{XZ}(v)$ in this example, are strictly monotonic. It should be possible to extend this technique to piecewise monotonic coordinate functions however.

Figures 7.3 and 7.4 show the reconstruction process. Figure 7.3 shows the orthographic projections onto the XY and XZ planes. Dots are shown at equally spaced points in the parametric domain of each curve to indicate that the parametrizations of the two curves are unrelated.

Figure 7.4 shows these curves after they have been reparametrized along the X axis. The space curve is then reconstructed from the reparametrized curves as discussed above.

7.3 Arc Length Spacing and Density

The algorithm of section 4.2 can be used to generate approximations to the inverse arc length function for NURBS curves. In addition to applications in motion synthesis these maps can be used to ensure proper spacing of parametric and geometric features. Below are two examples of this application.

Figure 7.5 shows geometric features equally spaced in arc length along a curved surface. This kind of distribution is often important, for example, in the placement of bolt holes on mating parts, the placement of embedded instrumentation such as miniaturized strain sensors, or the location of fasteners for the attachment of metal skins to bulkheads.

Figure 7.6 shows the effects of NURBS curve refinement on the results of a shape *warping* operator where knot density with respect to arc length is used as the refinement criterion. Loosely speaking, an attempt is made to add some uniform measure of "flexibility" along the spline curve per unit of its length. This refinement is accomplished by using a change of parameter map created by the arc length reparametrization algorithm of section 4.2. The effects of this refinement are compared to a standard refinement technique where additional knots are distributed uniformly over the original domain space.

In these examples the curve shapes are modified using the warp operator of Cobb [11]. The warp's center and direction are indicated by an arrow in the figure. The warp operator works by displacing the control points of the spline curve, and its effects are very sensitive to the degree of refinement and placement of knots in the curve. The warp results converge rapidly for the arc length technique.

Refinement by using densities with respect to arc length may be particularly useful when applied to models that maintain their arc length when manipulated. Such models include the flexible wire and tube elements of [59].

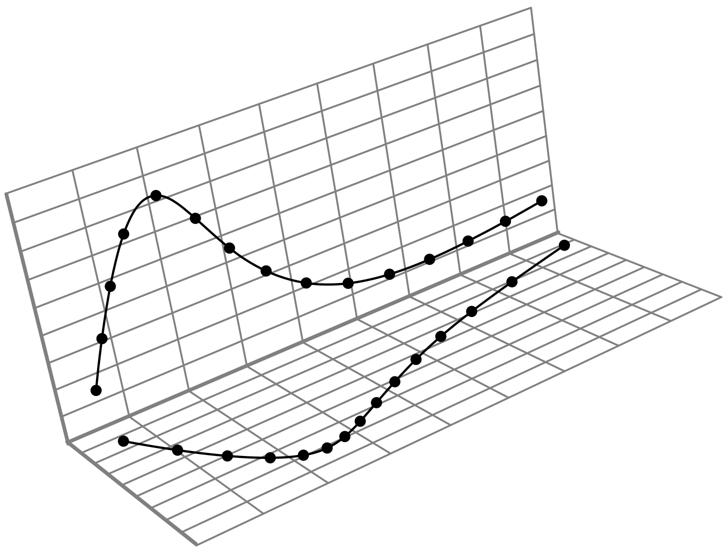


Figure 7.3. Example data for the reconstruction of 3D curves from 2D data. Dots on the curves indicate their relative parametrization.

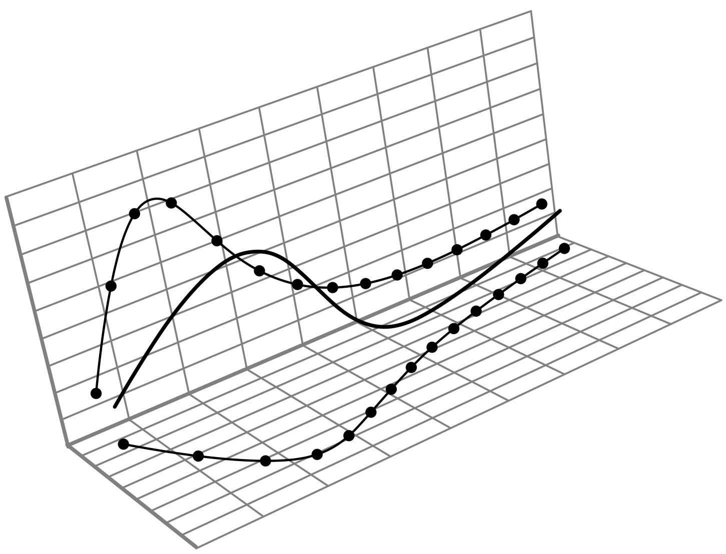


Figure 7.4. Same curves as in Figure 7.3 after reparametrization along the X axis. A 3D curve is reconstructed from the reparametrized 2D curves.

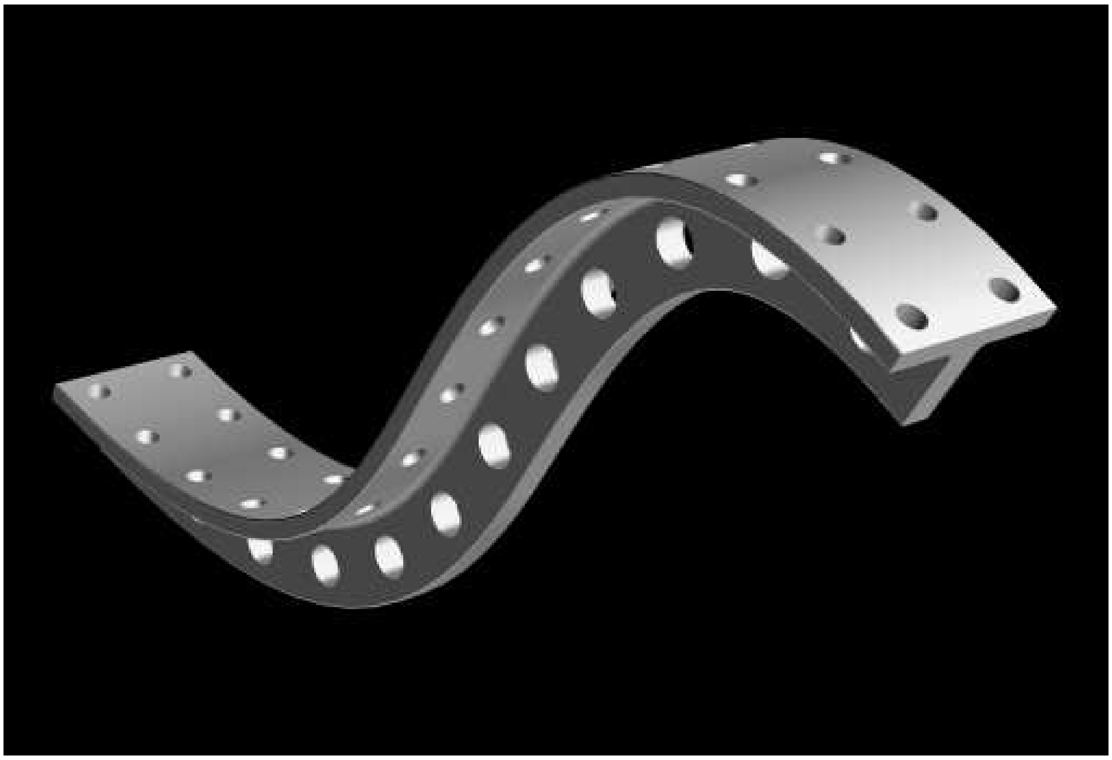


Figure 7.5. Structured space frame member showing rivet holes for the attachment of skins along the top surface and weight reduction holes along the lower surface. These holes are distributed evenly in arc length along the frame.

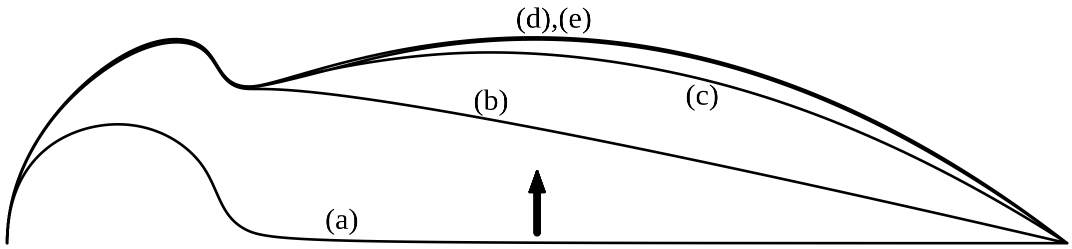


Figure 7.6. Effects of a warp operator on a curve after different knot refinement techniques are applied. (a) The original curve. (b) The effects of the warp operator with no additional knots. (c) The effects after the addition of 12 knots using a uniform knot density measure in the original parametric space. (d) The effects of the warp after the addition of 12 knots using a knot density measure with respect to arc length. (e) The effects of the warp after the addition of over 1000 knots using a knot density measure with respect to arc length. Curves (d) and (e) are hard to distinguish from one another in this figure. The arrow indicates the center and direction of the warp.

7.4 Curve Matching Via Point/Point Specifications

The algorithms of Chapters 4 and 5 change the parametrization of curves by using: a) metrics that measure how far curves are from the desired parametrization and b) iterative techniques to ensure convergence to within a user specified tolerance. Close adherence to a specific metric may not be an important consideration for some design work. The user may only desire that a given curve have a "better" parametrization; e.g., a better correspondence with another curve.

One method for accomplishing this is to specify the relative correspondence of two curves by using sparse pairs of matched points. These points are either specified directly by the user (for example by allowing the user to designate locations on a curve using a tablet or mouse) or specified by rules (these rules can be embodied by procedures written in a modeler command language). There are several benefits to using this technique.

- Certain correspondences may be very difficult to discover algorithmically. Hence a modeling system should allow the user an efficient method for directly specifying them.

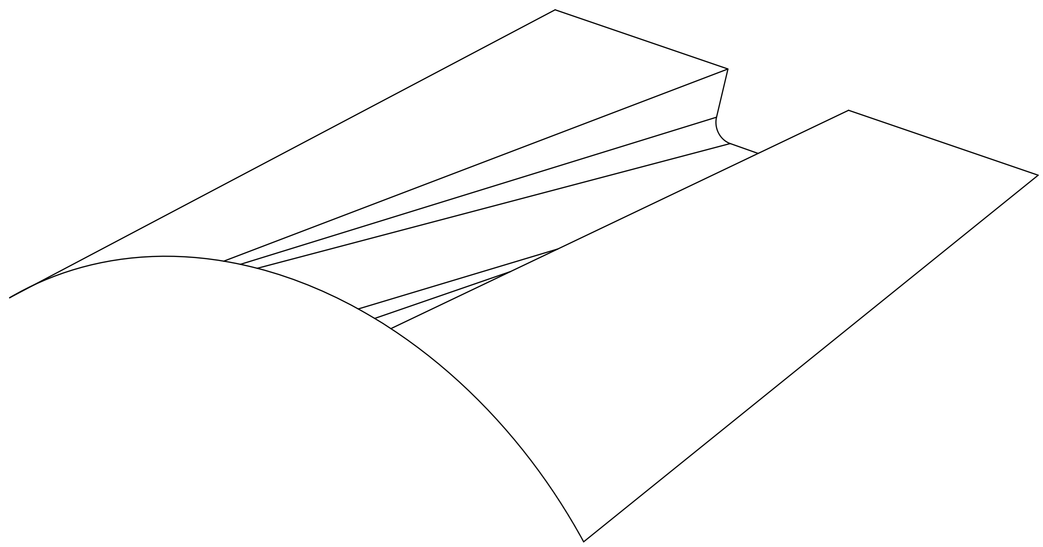
- This technique may have applications to user editing of automatically generated correspondences that are unsatisfactory only in limited intervals of the correspondence.

- The combination of sparse point matches generated by command language procedures with algorithmic reparametrization of curves allows for a wide class of experimentation with correspondence methods.

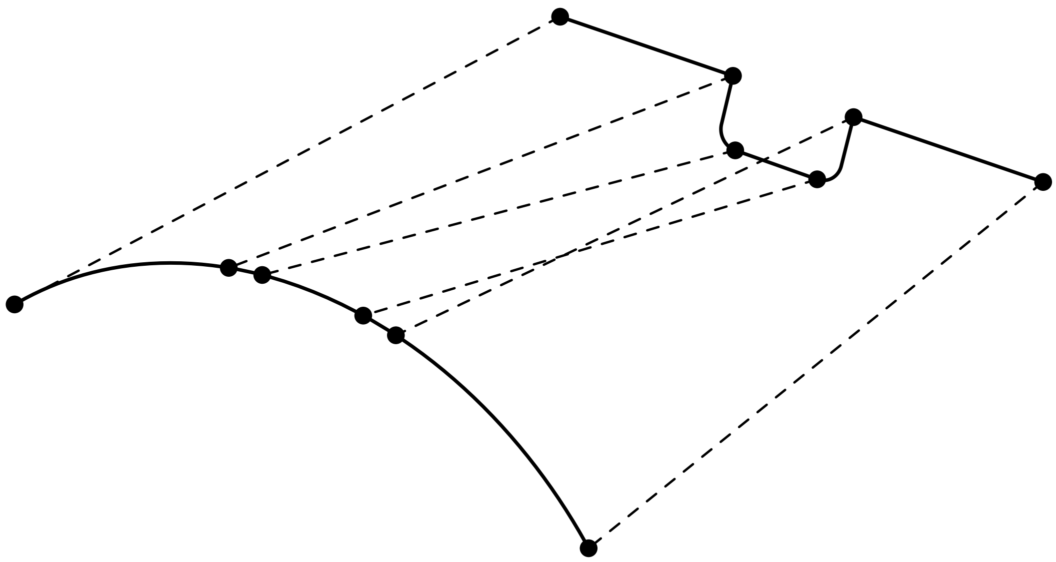
The change of parameter functions will be generated from the point matches using the available monotone shape preserving interpolants, $\mathcal{M}$, of section 4.1.2.

Figure 7.7 shows a surface designed for manufacture with wire EDM (electro-discharge machining), a process used for the fabrication of ruled surface designs. The parametric correspondence between the defining curves of the ruled surface affects the specific geometry of the result. Here the correspondence has been established using pairs of matched points as shown by the linked balls on the two curves.

Figure 7.8 shows examples of curves in radial correspondence. Rather than specifying a metric to which the reparametrized curves must adhere (the approach taken in sec-



(a)



(b)

Figure 7.7. Candidate shape for a transition surface to be manufactured using a wire EDM process. (a) This shape was modeled as a ruled surface generated from two planar curves. (b) To form the shape, the correspondence between the curves was established using the sparse point to point matching indicated by the linked balls on the two curves.

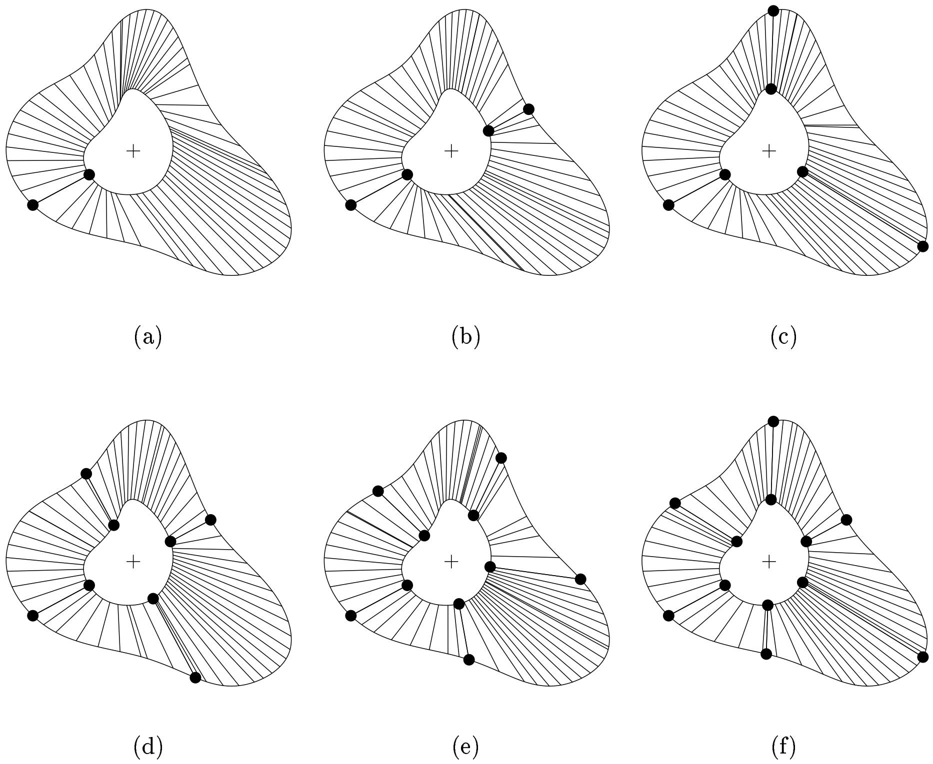


Figure 7.8. Radial reparametrization of one curve to match another. The reparametrization was accomplished using sparse point to point matches on the two curves. These matched points were generated by a procedure written in a geometric modeler's command language. Figures (a) through (f) show, respectively, the results obtained using one to six point to point matches, indicated by the corresponding dots on the curves. Crosses ("+") indicate the origin used for radial correspondence.

tion 5.2.2), a small number of matched points have been placed with approximately equal radial spacing about the central point. The matched points were generated with a user defined procedure written in a geometric modeler's command language [1].

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

8.1 Summary

This thesis argues that the reparametrization of curves is an important and practical approach to solving a variety of problems in CAD environments. With this approach users can:

- transform geometric representations into spaces better suited to establishing desired properties,
- relate properties and geometry in correspondence, and
- explore the design space.

Practical uses are surprisingly varied, from generating speed profiles along tool-paths, to approximating the inverses of functions, to determining topological relationships between surfaces in a boundary representation. As useful as these operations are, reparametrization is often lacking or underutilized in CAD environments.

To take advantage of the potential of these operations, reparametrization must be incorporated into CAD systems as practical design tools. Doing this requires that we address challenges apt to arise in any reparametrization scheme.

Users must be afforded the means of specifying change of parameters in ways that make sense to them. The results of these operations must be predictable and take usable forms.

In CAD systems it is necessary to compose the operations provided by the system to build up complex geometry. This makes closure of operations under a system's chosen representation of great importance. It may not be possible, however, to formulate exactly the desired reparametrizations within a CAD modeler's representation space. Algorithms must then create and use appropriate spaces of approximation for their results – spaces which are usually unknown initially. Errors in these approximations must be subject to control by the user in intuitive ways.

In most approximation schemes, a balance must be achieved between accuracy, computational cost, and complexity of the result. Many schemes to achieve approximate function composition increase the data complexity of results, in terms both of polynomial order and number of parametric spans in the result. This increased complexity carries the additional burden that it is often passed on to other entities through geometric constructions (for example surface constructions from reparametrized curves).

This thesis has presented new algorithms for NURBS curve reparametrization for important cases including: approximate reparametrization by arc length, approximate reparametrization by inverse functions, and reparametrization to establish bounds on the Frechet distance between curve mappings. These algorithms have true error bounds in terms of meaningful metrics for design and can be implemented as practical tools in CAD environments. Though important in their own right, these algorithms are representative of a wider class of reparametrization algorithms in the methods they employ to solve the challenges stated above. This thesis has demonstrated the utility of these algorithms by giving a series of applications and examples.

These algorithms were presented from within an overall framework that unifies the approach to seemingly disparate operators, and that yields practical solutions to the challenges mentioned above. The reparametrization algorithms presented in this thesis result from specialization of this framework. The core strategy afforded by the framework is to create adaptive algorithms that attempt to do work only where needed. This serves to restrain the growth in data complexity of the resulting approximations.

The framework for these algorithms also provides the user with trade-offs in computing results. The user can trade accuracy for complexity of result and can place the resulting reparametrized curve into spaces of any desired polynomial order. The alternative general strategies provided by Methods 1 and 2 give the user further choices over properties of the result. The user can control the accuracy of approximations by specifying error bounds in meaningful metrics which the algorithms maintain.

The applications presented in this thesis demonstrate a range of practical uses for curve reparametrization in CAD environments. The need for equally spaced features (such as bolt holes), or the need for controlling the speed of cutting tools while machining along feature curves, are common problems in CAD. So too is the need to correct improper correspondence between curves in surface constructions. Allowing users to specify, quickly and succinctly, appropriate correspondences is important also; examples of this capability given in the thesis include explicit point to point specifications and point to point matches established by the application of rules. Automatically determining approximate equivalence of curves, as in the algorithms of sections 4.4 and 5.2.1, is also an important operation that can be used to overcome common problems in CAD.

8.2 Extensibility

It is our belief that the framework given by Methods 1 and 2 is quite extensible. Its basic requirements are straightforward: a metric for measuring distance from a desired parametrization, a method for refining the space of approximation for the reparametrized curve (or change of parameter function), and a method for generating more data for the approximation. The substitution of different elements into this framework can lead to quite different algorithms. In section 5.2.2, for example, the Frechet distance algorithm was extended to an algorithm for computing relative radial reparametrizations, solely through the consideration of a different metric.

Although the algorithms given in this thesis have been developed in the context of NURBS curves, it may be possible to use different curve representations. In this regard it is instructive to consider what properties of the NURBS representation the general framework uses. These properties include:

- closure under function composition,
- closure under certain algebraic operations,
- the ability to bound properties of functions, especially function and derivative values,
- the existence of a refinement operation causing the coefficients of the representation to converge to the underlying function.

Although these properties characterize the NURBS representation, they also guide the search for other representations in CAD.

Extending the framework given by Methods 1 and 2 to surface representations is another possibility. Such an extension could be used to give surfaces better properties with respect to grid generation or texture mapping. Extending the framework to surfaces could also be used to establish correspondences between geometrically equivalent, or nearly equivalent, surfaces that have very different parametrizations.

8.3 Future Work

More experiments should be done in order to find better performance trade-offs regarding run time and data complexity. However, in view of the fact that a continued price is paid for increased data complexity, because of propagation of this complexity to dependent geometric constructions, it makes sense to spend more computational effort to decrease the data complexity of the result.

As one example of this, consider the arc length reparametrization algorithm of section 4.2. This algorithm develops a point-wise arc length approximation in parallel to the composed function approximation. Though computationally more expensive, it is a worthwhile experiment to see if doing this **serially** would decrease the data size of resulting arc length approximations; that is, start with an already very accurate approximation for the arc length function at the start of the algorithm.

Other techniques could be used to help decrease the data size of results, again at the cost of increased run time. Optimization techniques might prove useful in this regard as, for example, iteratively optimizing over the placement of free knots for the approximations r^i and a^i; similar to techniques used in [21]. Using data and degree reduction techniques [48, 49, 69, 68, 22, 57, 8] as postprocess steps for Methods 1 and 2, may also represent a worthwhile investment in computation in order to decrease resulting data complexity.

Users should be able to cascade, or compose, the effects of reparametrization operators. Consider, as an example, the arc length correspondence of two curves composed with point to point matches specified by the user. This kind of composition is possible in the current implementation; the user would first run the arc length algorithm on the curves, and then the point to point matching algorithm on the results. Such an approach, however, would increase the data complexity at each step. It may prove important to develop techniques where the effects of such cascaded operations can be carried out without the cost in data complexity of separate invocations of the algorithms.

Allowing users to edit explicitly the results of automatically generated reparametrizations is another example of such cascaded operations. Only relatively small intervals on a curve may need alteration. The desired change of parameter function would be the identity except on those intervals. We would like to minimize the penalty for the application of such change of parameter functions, paying the cost only in intervals where alteration is required. However, applying a sufficiently smooth change of parameter function might raise the polynomial order of the composed curve, effecting an increase in data complexity **everywhere** on the curve.

APPENDIX A

COMPOSITION OF NURBS

This section presents an algorithm for computing the knot vector used to represent the composition of two NURBS curves. Consider NURBS curve $c(u) : I_u \rightarrow \boldsymbol{R}^n$ and NURBS function $r(t) : I_t \rightarrow I_u$ a piecewise allowable change of parameter with $r(I_t) = I_u$. Assume that r is polynomial and c either polynomial or rational.

Let r be order l defined over knot vector τ and c order m defined over knot vector $\hat{\mu}$. In order to form the composition $c(r(t))$ the knots of c must be mapped through $r^{-1}(t)$. Let $\mu = r^{-1}(\hat{\mu})$ be the knot vector resulting from mapping each knot of $\hat{\mu}$ through the inverse of r. The composed function $c(r(t))$ is a polynomial or rational spline of order $o = (l-1)(m-1)+1$ defined over knot vector σ given by the algorithm below.

For a general polynomial spline function $r(t)$ the inverse problem $\mu = r^{-1}(\hat{\mu})$ can be solved only approximately. This inverse problem can be avoided, however, if enough is known about the function r. If r is formed using interpolation to discrete values that include the knots of c (as in sections 4.2.3, 4.3.2 and 4.4.4) then $\mu = r^{-1}(\hat{\mu})$ is already known.

In the following algorithm first-μ-knot(t_μ, m_μ) and next-μ-knot(t_μ, m_μ) assign the first (respectively next) **unique** knot value in μ to the variable t_μ and the multiplicity of this value in μ to m_μ. The routines first-τ-knot(t_τ, m_τ) and next-τ-knot(t_τ, m_τ) assign the first (respectively next) unique knot value in τ to the variable t_τ and the multiplicity of this value in τ to m_τ. Routines next-μ-knot and next-τ-knot return FALSE if their respective knot vectors are exhausted, otherwise they return TRUE.

Input:

m the order of $c(u)$,
$\hat{\mu}$ the knot vector for $c(u)$,
l the order of $r(t)$,
τ the knot vector for $r(t)$.

Output:

The knot vector σ for $c(r(t))$.

Algorithm:

```
μ = r⁻¹(μ̂);
o = (l − 1)(m − 1) + 1;
first-μ-knot( t_μ, m_μ );
first-τ-knot( t_τ, m_τ );
got-μ-knot = TRUE;
got-τ-knot = TRUE;

while ( got-μ-knot or got-τ-knot )
{
  if ( not got-μ-knot )
  {
    add t_τ to σ with multiplicity m_τ − l + o;
    got-τ-knot = next-τ-knot( t_τ, m_τ );
  }
  else if ( not got-τ-knot )
  {
    add t_μ to σ with multiplicity m_μ − m + o;
    got-μ-knot = next-μ-knot( t_μ, m_μ );
  }
  else if ( t_τ ≈ t_μ )
  {
    add t_τ to σ with multiplicity max(m_τ − l, m_μ − m) + o;
    got-τ-knot = next-τ-knot( t_τ, m_τ );
    got-μ-knot = next-μ-knot( t_μ, m_μ );
  }
  else if ( t_τ < t_μ )
  {
    add t_τ to σ with multiplicity m_τ − l + o;
    got-τ-knot = next-τ-knot( t_τ, m_τ );
  }
  else
  {
    add t_μ to σ with multiplicity m_μ − m + o;
    got-μ-knot = next-μ-knot( t_μ, m_μ );
  }
}
```

APPENDIX B

MONOTONE APPROXIMATION OPERATORS

This section discusses two candidates for the monotone approximation operator $\mathcal{M}$ of section 4.1.2. These are the C^1 linear rational spline interpolation scheme of [32] and the C^1 quadratic spline interpolation scheme of [60].

The purpose here is to give an outline of the properties and data requirements of these two schemes and to indicate useful modifications that allow their incorporation into the algorithms of Chapters 4 and 5. The linear rational and quadratic spline schemes share very similar requirements and properties and hence will be discussed together.

These schemes interpolate scalar valued data at specified locations. These data consist of function and derivative values specified as $\{(t_i, v_i, \dot{v_i})\}_{i=0}^{n}$. Here t_i is the parametric location at which the constructed interpolant will assume function value v_i and derivative value $\dot{v_i}$. For our purposes, these data are assumed to be consistent with a strictly monotonic increasing function; i.e.: the locations t_i must be distinct, the v_i must increase with increasing values of t_i, and the $\dot{v_i}$ must be positive.

A scalar valued spline is created that interpolates these data at knots of the spline. The quadratic scheme creates an interpolant that is **constrained** to be C^1, whereas the linear rational scheme creates an interpolant which is C^1 **by construction only**.

It is shown in [32] that the linear rational interpolation scheme never fails to preserve monotonicity in the data. The quadratic scheme attempts to preserve both monotonicity **and** convexity present in the data, but may fail to preserve these properties. Failure to preserve either property can be detected in this scheme and reported to the calling program (see [60] for details).

Figure B.1 diagrams the layout of data relative to knot locations for the interpolants in both schemes. As indicated above, the t_i are parametric locations where data are to be interpolated. The v_i and $\dot{v_i}$ are, respectively, the function and derivative values to be interpolated. Singleton knots are placed at all interior t_i locations. The s_i represent inter-

v_1		v_2		...	v_{n-1}		v_n
$\dot{v_1}$		$\dot{v_2}$		...	$\dot{v_{n-1}}$		$\dot{v_n}$
$\wedge^3$	$\wedge$	$\wedge$	$\wedge$	...	$\wedge$	$\wedge$	$\wedge^3$
t_0	s_0	t_1	s_1	...	t_{n-1}	s_{n-1}	t_n

Figure B.1. Data layout for both the linear rational and quadratic interpolation schemes. The symbol "$\wedge$" indicates the placement of singleton knots; "$\wedge^3$" indicates triple knots at the ends for the quadratic scheme. For the linear rational scheme the end knots would be double.

mediate locations where additional singleton knots are placed. For open end conditions order-fold knots are placed at locations t_0 and t_n. Figure B.1 shows the placement of threefold knots at these locations for the quadratic scheme. For the linear rational scheme, double knots would be placed at t_0 and t_n. The knot vector for the spline interpolant is therefore (in the quadratic case):

$$(t_0, t_0, t_0, s_0, t_1, s_1, ..., t_{n-1}, s_{n-1}, t_n, t_n, t_n).$$

Figure B.2 shows the layout of data with the inclusion of equality constraints. The symbol "$=$" indicates that equality of function values is constrained at the specified parametric location. The symbol "$\doteq$" indicates that equality of derivative values is constrained at the specified parametric location. The symbols v_i and $\dot{v_i}$ count as value and derivative constraints respectively. For each parametric span there is a total of three degrees of freedom for both the quadratic and linear rational interpolation schemes. It is readily verified that the total number of constraints equals the total number of degrees of freedom for these schemes and that these constraints can be satisfied locally on an interval $[t_i, t_{i+1}]$.

The linear rational scheme will maintain monotonicity of the data regardless of where the intermediate knot values s_i are placed. For a given i the quadratic scheme of [60], on the other hand, computes the knot location $s_i \in (t_i, t_{i+1})$ in an attempt to preserve monotonicity and convexity of the data represented by the tuples $(t_i, v_i, \dot{v_i})$ and

v_1	$=$	$v_2, =$	$=$	...	$v_{n-1}, =$	$=$	v_n
$\dot{v_1}$	$\doteq$	$\dot{v_2}, \doteq$	$\doteq$	...	$\dot{v_{n-1}}, \doteq$	$\doteq$	$\dot{v_n}$
$\wedge^3$	$\wedge$	$\wedge$	$\wedge$	...	$\wedge$	$\wedge$	$\wedge^3$
t_0	s_0	t_1	s_1	...	t_{n-1}	s_{n-1}	t_n

Figure B.2. Diagram of the constraints for the linear rational and quadratic interpolation schemes.

$$
\begin{array}{ccccccc}
\cdots & v_{i-1} & & v_i & & v_{i+1} & \cdots \\
\cdots & \dot{v}_{i-1} & & \dot{v}_i^{\,l}, \dot{v}_i^{\,r} & & \dot{v}_{i+1} & \cdots \\
\hline
\cdots & \wedge & \wedge & \wedge^2 & \wedge & \wedge & \cdots \\
\cdots & t_{i-1} & s_{i-1} & t_i & s_i & t_{i+1} & \cdots
\end{array}
$$

Figure B.3. Modification of the interpolation schemes for lowered continuity at an interpolation point. "$\wedge^2$" indicates the placement of a double knot. "$\dot{v}_i^{\,l}$" and "$\dot{v}_i^{\,r}$" indicate left and right derivative values respectively.

$(t_{i+1}, v_{i+1}, \dot{v}_{i+1})$. A failure to maintain monotonicity by this scheme can be "corrected" by the inclusion of additional degrees of freedom in the system. Two methods for doing this are indicated below.

If monotonicity over an interval $[t_i, t_{i+1}]$ fails **two** intermediate knots internal to this interval (rather than just one) can be added to maintain both monotonicity on the interval and the C^1 conditions at t_i and t_{i+1}. The resulting system becomes under constrained however. An alternative is to maintain monotonicity by lowering the continuity at the ends of the interval and inserting a linear segment on $[t_i, t_{i+1}]$.

Both the linear rational and quadratic schemes need to be modified in cases where the continuity should be lowered at a point of interpolation. If it is known that the interpolant should only be C^0 at a location (see for example sections 4.2.3 and 4.3.2) the scheme is modified as shown in Figure B.3. Here a double knot is inserted at a point of interpolation (one of the t_i) and both left and right derivative values are now required in the input data.

APPENDIX C

APPROXIMATING ARC LENGTH

Given $c(u) : I_u = [u_s, u_e] \rightarrow \boldsymbol{R}^3$, a NURBS curve of order o defined over knot vector τ, we want to approximate the arc length function $s(u) = \int_{u_s}^{u \in I_u} \| \frac{d}{d\mu} c(\mu) \| d\mu$ at a set of discrete points $\{u_j\}_j$ in the curve's parametric domain. A variety of techniques exist for approximating values on this integral (see Chapter 3). This appendix presents two methods, either of which can be readily incorporated into the algorithm of section 4.2.

The first method forms a refinement of the NURBS curve representation for $c(u)$ by knot insertion. This refined representation is in a space $\mathcal{S}_{o,\hat{\tau}}$ where knot vector $\hat{\tau}$ is a refined partition of τ and includes knots of multiplicity $o-1$ at each of the u_j. Additional knots of multiplicity one can be inserted between the u_j in order to make the approximation more accurate.

For a given u_j, let m be the index of the first knot in $\hat{\tau}$ with the value u_j and let $\{C_k\}_k$ be the set of control points for the refined representation of $c(u)$ defined on $\mathcal{S}_{o,\hat{\tau}}$. The control point with zero based index $m-1$ will interpolate the point $c(u_j)$. Thus an approximation for the arc length at u_j is given by the partial sum:

$$\sum_{k=1}^{m-1} \|C_{k-1} - C_k\|. \tag{C.1}$$

Taken over a series of refined partitions $\hat{\tau}^i$, the approximations given above will converge to values of the arc length function for $c(u)$ as the mesh norm of $\hat{\tau}^i$ converges to zero. It can be shown that this series of approximations forms an **upper** bound on the arc length function values at the u_j.

In general the knot vectors $\hat{\tau}^i$ should be refined over intervals of the parametric domain where the error in the arc length approximation is outside an acceptable tolerance. The metric of section 4.2.1 can serve to measure unacceptable error in the context of the arc length reparametrization algorithm of section 4.2.

Another approach to approximating the arc length function at the u_j is to form **lower** bound estimates using inscribed polygons that include the points $c(u_j)$ as vertices. Again a series of refined partitions of the domain can be used to determine points on c, between the $c(u_j)$, which are included as additional polygon vertices. Taken over a series of refined partitions, the approximations will converge to values of the arc length function for $c(u)$ from below.

Either of these techniques can be incorporated into the arc length reparametrization algorithm of section 4.2. The one sided nature of the techniques' arc length approximations, one from above and one from below, can induce different biases in the speed errors of approximate arc length parametrized curves.

REFERENCES

[1] ALPHA_1 RESEARCH GROUP. *Alpha_1 User's Manual.* University of Utah, Salt Lake City, Utah, 1999.

[2] ALT, H., AND GODAU, M. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications 5*, 1&2 (1995), 75–91.

[3] BEIER, T., AND NEELY, S. Feature-based image metamorphosis. *Computer Graphics, SIGGRAPH 92 Conference Proceedings 26*, 2 (July 1992), 35–42.

[4] BENNIS, C., VÉZIEN, J.-M., AND IGLÉSIAS, G. Piecewise surface flattening for non-distorted texture mapping. *Computer Graphics, SIGGRAPH 91 Conference Proceedings 25*, 4 (July 1991), 237–246.

[5] BESL, P. J., AND MCKAY, N. D. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (February 1992), 239–256.

[6] BLANC, C., AND SCHLICK, C. Accurate parametrization of conics by NURBS. *IEEE Computer Graphics and Applications 16*, 6 (November 1996), 64–71.

[7] BLOOMENTHAL, M. Approximation of sweep surfaces by tensor product B-splines. Tech. Rep. UUCS-88-008, Department of Computer Science, University of Utah, Salt Lake City, Utah, August 1988.

[8] BRAUN, J. Optimal degree reduction of freeform curves. Master's thesis, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, January 1995.

[9] CAMERON, S. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation 13*, 6 (December 1997), 915–920.

[10] CASCIOLA, G., AND MORIGI, S. Reparametrization of NURBS curves. *International Journal of Shape Modeling 2*, 2&3 (1996), 103–116.

[11] COBB, E. S. *Design of Sculptured Surfaces Using the B-spline Representation.* PhD thesis, University of Utah, Salt Lake City, Utah, June 1984.

[12] COHEN, E., LYCHE, T., AND RIESENFELD, R. F. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing 14*, 2 (October 1980), 87–111.

[13] Cohen, E., Lyche, T., and Schumaker, L. L. Algorithms for degree-raising of splines. *ACM Transactions on Graphics 4*, 3 (July 1986), 171–181.

[14] Cohen, E., and Schumaker, L. L. Rates of convergence of control polygons. *Computer Aided Geometric Design 2*, 1-3 (September 1985), 229–235.

[15] Cohen, S., Elber, G., and Yehuda, R. B. Matching of freeform curves. *Computer-Aided Design 29*, 5 (May 1997), 369–378.

[16] Cohen-Or, D., Levin, D., and Solomovici, A. Three-dimensional distance field metamorphosis. *Transactions on Graphics 17*, 2 (April 1998), 116–141.

[17] Coons, S. A. Surface patches and B-spline curves. In *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. Academic Press, New York, 1974, pp. 1–16.

[18] Coquillart, S. A control-point-based sweeping technique. *IEEE Computer Graphics and Applications 7*, 11 (November 1987), 36–45.

[19] de Boor, C. *A Practical Guide to Splines*, vol. 27 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1978.

[20] de Boor, C., and Fix, G. J. Spline approximation by quasi-interpolants. *Journal of Approximation Theory 8* (1973), 19–45.

[21] Driskill, H. A. *COMA, Constrained Optimization for Modeling and Animation*. PhD thesis, University of Utah, Salt Lake City, Utah, December 1995.

[22] Eck, M. Degree reduction of Bezier curves. *Computer Aided Geometric Design 10*, 3-4 (August 1993), 237–251.

[23] Elber, G. *Free Form Surface Analysis using a Hybrid of Symbolic and Numeric Computation*. PhD thesis, University of Utah, Salt Lake City, Utah, December 1992.

[24] Elber, G. Symbolic and numeric computation in curve interrogation. *Computer Graphics Forum 14*, 1 (March 1995), 25–34.

[25] Emery, J. D. The definition and computation of a metric on plane curves. *Computer-Aided Design 18*, 1 (January/February 1986), 25–28.

[26] Ewing, G. M. *Calculus of Variations with Applications*. Dover Publications, New York, 1985.

[27] Farin, G. *Curves and Surfaces for Computer Aided Geometric Design – A Practical Guide*. Academic Press, San Diego, California, 1993.

[28] Farouki, R. T. Optimal parameterizations. *Computer Aided Geometric Design 14*, 2 (February 1997), 153–168.

[29] Farouki, R. T., and Rajan, V. T. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design 5*, 1 (June 1988), 1–26.

[30] FAROUKI, R. T., AND SAKKALIS, T. Real rational curves are not 'unit speed'. *Computer Aided Geometric Design 8*, 2 (May 1991), 151–157.

[31] FRITSCH, F. N., AND NIELSON, G. M. On the problem of determining the distance between parametric curves. In *Curve and Surface Design*, H. Hagen, Ed. Siam, 1992, ch. 7, pp. 123–141.

[32] FUHR, R. D., AND KALLAY, M. Monotone linear rational spline interpolation. *Computer Aided Geometric Design 9*, 4 (September 1992), 313–319.

[33] GILBERT, E. G., JOHNSON, D. W., AND KEERTHI, S. S. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation 4*, 2 (April 1988), 193–203.

[34] GODAU, M. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *STACS 91. 8th Annual Symposium on Theoretical Aspects of Computer Science Proceedings* (Hamburg, Germany, 14-16 February 1991), C. Choffrut and M. Jantzen, Eds., Springer-Verlag; Berlin, Germany, pp. 127–136.

[35] GORDON, W. J., AND RIESENFELD, R. F. B-spline curves and surfaces. In *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. Academic Press, New York, 1974, pp. 95–126.

[36] GUENTER, B., AND PARENT, R. Computing the arc length of parametric curves. *IEEE Computer Graphics and Applications 10*, 3 (May 1990), 72–78.

[37] HARTLEY, P. J., AND JUDD, C. J. Parametrization and shape of B-spline curves for CAD. *Computer-Aided Design 12*, 5 (September 1980), 235–238.

[38] HORSCH, T., AND JÜTTLER, B. Cartesian spline interpolation for industrial robots. *Computer-Aided Design 30*, 3 (March 1998), 217–224.

[39] HOSCHEK, J., AND LASSER, D. *Fundamentals of Computer Aided Geometric Design.* A K Peters, Wellesley, Massachusetts, 1993.

[40] JOHNSON, D. E., AND COHEN, E. Minimum distance queries for polygonal and parametric models. Tech. Rep. UUCS-97-003, Department of Computer Science, University of Utah, Salt Lake City, Utah, February 1997.

[41] JOHNSON, D. E., AND COHEN, E. A framework for efficient minimum distance computations. In *Proceedings of the 1998 IEEE International Conference on Robotics & Automation* (Leuven, Belgium, May 16-21 1998), pp. 3678–3684.

[42] KOSTERS, M. Curvature-dependent parameterization of curves and surfaces. *Computer-Aided Design 23*, 8 (October 1991), 569–578.

[43] LEE, E. T. Y., AND LUCIAN, M. L. Möbius reparametrizations of rational B-splines. *Computer Aided Geometric Design 8*, 3 (August 1991), 213–215.

[44] LEE, S.-Y., CHWA, K.-Y., SHINE, S. Y., AND WOLBERG, G. Image metamorpho-

sis using snakes and free-form deformations. *Computer Graphics, SIGGRAPH 95 Conference Proceedings 29* (August 1995), 439–448.

[45] Lévy, B., and Mallet, J.-L. Non-distorted texture mapping for sheared triangulated meshes. *Computer Graphics, SIGGRAPH 98 Conference Proceedings* (July 1998), 343–352.

[46] Lin, M. C., and Canny, J. F. A fast algorithm for incremental distance calculation. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation* (Sacramento, California, April 9-11 1991), vol. 2, pp. 1008–1014.

[47] Lipschutz, M. M. *Theory and Problems of Differential Geometry.* McGraw-Hill, New York, 1969.

[48] Lyche, T., and Mørken, K. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design 4*, 3 (November 1987), 217–230.

[49] Lyche, T., and Mørken, K. A data reduction strategy for splines with applications to the approximation of functions and data. *IMA Journal of Numerical Analysis 8* (1988), 185–208.

[50] Lyche, T., and Schumaker, L. L. Local spline approximation methods. *Journal of Approximation Theory 15*, 4 (December 1975), 294–325.

[51] Maillot, J., Yahia, H., and Verroust, A. Interactive texture mapping. *Computer Graphics, SIGGRAPH 93 Conference Proceedings* (August 1993), 27–34.

[52] Marsden, M. J. An identity for spline functions with applications to variation-diminishing spline approximation. *Journal of Approximation Theory 3*, 1 (March 1970), 7–49.

[53] Mastin, C. W. Parameterization in grid generation. *Computer-Aided Design 18*, 1 (January/February 1986), 22–24.

[54] Millman, R. S., and Parker, G. D. *Elements of Differential Geometry.* Prentice-Hall, Englewood Cliffs, NJ, 1977.

[55] Mørken, K. Some identities for products and degree raising of splines. *Journal of Constructive Approximation 7*, 2 (1991), 195–208.

[56] Munkres, J. R. *Topology a First Course.* Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[57] Piegl, L., and Tiller, W. Algorithm for degree reduction of B-spline curves. *Computer-Aided Design 27*, 2 (February 1995), 101–110.

[58] Piegl, L., and Tiller, W. *The NURBS Book.* Springer-Verlag, Berlin, 1997.

[59] Pugmire, D. *Thin Flexible Elements in CAGD.* PhD thesis, University of Utah, Salt Lake City, Utah, to be published.

[60] Schumaker, L. L., and Stanley, S. S. Shape-preserving knot removal. *Computer*

Aided Geometric Design 13, 9 (December 1996), 851–872.

[61] Sederberg, T. W., and Greenwood, E. A physically based approach to 2-D shape blending. *Computer Graphics, SIGGRAPH 92 Conference Proceedings 26*, 2 (July 1992), 25–34.

[62] Sharpe, R. J., and Thorne, R. W. Numerical method for extracting an arc length parameterization from parametric curves. *Computer-Aided Design 14*, 2 (March 1982), 79–81.

[63] Sloan, P.-P. J., Weinstein, D. M., and Brederson, J. D. Importance driven texture coordinate optimization. *Eurographics'98 17*, 3 (1998).

[64] Soni, B. K., and Yang, S. NURBS-based surface grid redistribution and remapping algorithms. *Computer Aided Geometric Design 12*, 7 (November 1995), 675–692.

[65] Srinivasan, L. N., and Ge, Q. J. Fine tunning of rational B-splines motions – DETC97/DAC03984. In *Proceedings of DETC'97, 1997 ASME Design Engineering Technical Conferences* (Sacramento, California, September 14-17 1997).

[66] Versprille, K. J. *Computer-Aided Design Applications of the Rational B-spline Approximation Form.* PhD thesis, Syracuse University, Syracuse, New York, 1975.

[67] Wang, F.-C., and Yang, D. C. H. Nearly arc-length parameterized quintic-spline interpolation for precision machining. *Computer-Aided Design 25*, 5 (May 1993), 281–288.

[68] Watkins, M. A., and Worsey, A. J. Degree reduction of Bezier curves. *Computer-Aided Design 20*, 7 (September 1988), 398–405.

[69] Wever, U. Global and local data reduction strategies for cubic splines. *Computer-Aided Design 23*, 2 (March 1991), 127–132.

[70] Wever, U. Optimal parameterization for cubic splines. *Computer-Aided Design 23*, 9 (November 1991), 641–644.

[71] Zhang, Z. On local matching of free-form curves. In *BMVC92. Proceedings of the British Machine Vision Conference* (Leeds, UK, 22-24 September 1992), D. Hogg and R. Boyle, Eds., Springer-Verlag; Berlin, Germany, pp. 347–56.

[72] Zhang, Z. Point matching for registration of free-form surfaces. In *Computer Analysis of Images and Patterns. 5th International Conference, CAIP '93 Proceedings* (Budapest, Hungary, 13-15 September 1993), D. Chetverikov and W. G. Kropatsch, Eds., Springer-Verlag; Berlin, Germany, pp. 460–467.