MINIMUM DISTANCE QUERIES FOR HAPTIC RENDERING

by

David Edward Johnson

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

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

May 2005

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

David E. Johnson

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

_____	_____
	Chair: Elaine Cohen
_____	_____
	Richard Riesenfeld
_____	_____
	Samuel Drake
_____	_____
	William Thompson
_____	_____
	John Hughes

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of David E. Johnson 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

Christopher Johnson
Chair

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

Finding the closest points between two modeled objects is a fundamental operation in robotics, computer graphics, and computational geometry. This dissertation is motivated by the use of distance functions in haptic interfaces for virtual prototyping, where distance measures provide the basis for forces that are applied to a human user. The requirements for haptic interfaces mean that these distances must be computed both quickly and robustly.

This dissertation begins by exploring the robustness of simple numerical methods for finding the minimum distance between a point and a curve. A geometric analysis of the convergence conditions yields an algorithm for precomputing a set of starting values with robustness guarantees. Embedding this simple local method within a geometric convergence test then provides some guarantees of global convergence.

The requirements of haptic interfaces motivate another approach, based on normal cones, for global search of local minima. This technique extends to surfaces and mixed with numerical methods allows a haptic rendering system for NURBS models.

Finally, the normal cone approach is applied to polygonal models, which provides the basis for a general 6DOF haptic interface for virtual prototyping. These methods provide significant performance and reliability benefits over existing haptic rendering techniques.

For Susan and Lucy

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I am grateful to the members of the Virtual Prototyping Group for initiating and developing an interesting research endeavor in haptics, in particular, Elaine Cohen, John Hollerbach, and Bill Thompson. My student colleagues in haptics, Tom Thompson and Don Nelson, provided great insight as well as hard work to advance haptic rendering. The members of my committee have provided valuable feedback on this research and dissertation. This work could not have been completed without the support of several agencies, the National Science Foundation and Army Research Office in particular, and I am grateful for their support.

CHAPTER 1

INTRODUCTION

The practical pursuit of a computer environment for virtual prototyping guides this research into the more abstract realm of geometric computations. In particular, adding force-feedback human-computer interfaces to virtual prototyping environments, so that a person's sense of touch can guide placement of virtual objects, has motivated the development of efficient algorithms to compute the distance and interactions between these objects. The resulting distance algorithms are basic computational building blocks that are useful not only in force-feedback interfaces, but also for a broad class of geometric computations.

There are three main areas to this dissertation. The first area examines distance algorithms for distance queries between a point and a curve and produces algorithms for local and global distance solutions. The insights from this investigation guide the development of distance algorithms for surfaces in the second area and culminate in a practical implementation of a distance algorithm for NURBS models suitable for haptic rendering of NURBS surfaces. The third area applies the ideas of the local minima method algorithms to purely geometric algorithms and shows how these concepts can be applied to polyhedral models. These polyhedral algorithms are adapted for use in a virtual prototyping system with force-feedback.

The next few sections provide more detail for why virtual prototyping is an interesting problem, and how virtual prototyping requirements are connected to the computation of distance minima.

Virtual Prototyping

Mechanical designers have employed computer-aided design (CAD), computer-aided engineering (CAE) and computer-aided manufacture (CAM) in their design processes. This integration allows products to be designed and analyzed on computers before manufacture takes place. However, physical prototypes are typically used at some point in the process to validate the proposed design. In industries such as the car industry, designers may cycle between computer models and physical prototypes multiple times. These cycles may take considerable time and expense due to the difficulty in translating the model between the virtual and physical worlds.

Virtual prototyping replaces physical prototypes with virtual objects in a computer. Ideally, the virtual prototyping environment can use the same computer models that are used in the design processes, else a cumbersome translation from one computer model format to another is needed. The virtual prototyping environment must provide the same design evaluation functionality as the physical prototype it replaces. Typically, designers use physical prototypes to evaluate model aesthetics, ergonomics, and assembility.

Virtual prototyping systems often depend heavily on advanced displays, such as head mounted displays (HMD) or wall displays like the CAVE, to give a proper sense of scale to the virtual objects. These systems have focused primarily on evaluation of the visual aesthetic qualities of the design. However, for many objects other senses play an

important role in the aesthetic quality of a design. In car design, the way a door feels and sounds as it closes imparts sensations of solidity and reliability. The feel of the seats similarly influences perceptions of luxury. In these examples, the sense of touch plays a major component in the aesthetic evaluation of a prototype.

For other prototyping tasks, the sense of touch is similarly important. Sensations of contact guide the assembly process when checking how various parts of a model fit together. Without a way to reflect those sensations to the user of the virtual prototyping environment, they must be mapped into new sensory channels, such as alarm sounds or changes of color for the colliding objects. This remapping, as well as the lack of expected touch cues, can be disconcerting for the users of the system, creating a less effective user experience.

Similarly, the controls of many objects are designed to be manipulated by hand with adjustments guided by the sense of touch. To properly evaluate the ergonomic quality of these controls, the virtual prototype must engage this additional sensory channel. The predominance of visual feedback in current virtual prototyping systems makes testing the ergonomic quality of a design difficult.

Haptic Interfaces

Haptic interfaces provide a means of engaging the sense of touch in a human-computer interface, and thus provide a means of addressing the lack of touch cues in virtual prototyping environments. Haptic means “relating to or based on the sense of touch” and haptic interfaces may engage a person’s tactile or kinesthetic sense of touch. A haptic interface is a robotic mechanism controlled by computer and attached to a person. This mechanism can reflect forces simulated in a virtual prototyping

environment, such as those computed from collisions of virtual objects, back to the person attached to the device. Thus, a person's hand or arm may be prevented from moving in a certain direction when doing so would require that objects in the virtual world would interpenetrate.

This reflection of simulated forces is known as haptic rendering. Just as rendering in computer graphics simulates the interaction of light with virtual objects and presents that simulation through a visual display, haptic rendering simulates the forces of contact and presents that simulation through a haptic device. A key computational component of the force simulation is the ability to measure the distance between two virtual objects.

Distance Measures

Computing the distance between virtual objects, variously referred to as the minimum distance problem or the closest point problem, is a generalization of collision detection, a long-studied problem in robotics and computer graphics that determines if two objects are in contact. When objects are interpenetrated, measuring the penetration depth is a related computation to the minimum distance problem.

Our interest in computing the distance between objects rather than just their collision status derives from two main reasons. First, distance measures are predictive. They can report not only when two objects are in contact, but also when two objects are close and likely to collide, or even when two objects are distant and are unlikely to have any influence on each other. Second, when objects are in contact, distance measures can give a sense of how badly things have gone awry and even how to rectify the situation by providing a direction to collision response methods. This second reason is a key component of the haptic rendering computation.

Summary

This introduction has provided a chain of dependency from the utility of virtual prototyping, to the need for haptic interfaces in these environments, and finally to the dependency of haptic rendering on fast and robust minimum distance computation between virtual objects. The remainder of the document will focus primarily on distance computations, but many of the decisions made in algorithm design will be motivated by their eventual use in a virtual prototyping system.

CHAPTER 2

BACKGROUND

Prior art in the area of distance computation draws from a number of fields, including computer graphics[1][2], robotics[3][2][4], computational geometry[5][6], and haptics[7][8][10][11]. This chapter provides an overview of distance computations, then devotes more detail to distance computations in support of haptic rendering.

Distance

In the Euclidian space $\mathbf{R}^n$ the Euclidian distance between two points x and y of $\mathbf{R}^n$ is the L^2 distance

$$D(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2} : x, y \in \mathbf{R}^n, \quad \textbf{(1)}$$

or more compactly,

$$D(x,y) = \|x - y\|. \quad \textbf{(2)}$$

The Euclidian distance between two subsets S and T of a Euclidian space is the *infimum* of the vector lengths between all points in the two sets, or

$$D(S,T) = \inf\{D(s,t) : s \in S, t \in T\}. \quad \textbf{(3)}$$

The *closest points* between the subsets *S* and *T* are the points *s* in *S* and *t* in *T* whose distance is *D(S,T)*, if there are any. There may be multiple pairs of closest points, or none at all in the case of open sets. For the remainder of this thesis assume *S* and *T* are closed, a valid assumption for the points, triangular models, and NURBS curves and surfaces used as models in this dissertation.

Modeling

In geometric modeling, these subsets of Euclidian space are often represented as a boundary representation model. The minimum distance problem then becomes finding the closest points on the surfaces representing the model boundaries.

Prior minimum distance algorithms have used different approaches depending on the types of models being queried. Distance algorithms for polygonal models have favored geometric pruning methods[12][13][14], while algorithms for parametric surfaces have concentrated on numerical techniques[15][1][16].

Distance Queries for Polygonal Models

Polygonal models are typically composed of collections of triangles, and most distance algorithms for polygonal models deal with triangle primitives. The model may contain topological connectivity information. Models without connectivity are known as a triangle cloud, and ones with are properly described as a triangle mesh.

Lin[17] and Gilbert, Johnson, and Keerthi[18] developed fast minimum distance methods for convex polygonal models. Since local gradient search produces a global minimum for convex objects, their algorithms can converge quickly.

Quinlan[14] developed a spherical bounding hierarchy for general triangle clouds. The bounding hierarchy was used to determine an upper bound on minimum distance between the two models, and then to prune away portions of each model with lower bounds on distance larger than the upper bound. The PQP package, by Larsen et al.[19], followed the successful application of oriented bounding boxes to collision detection[20] by using swept sphere volumes as a bounding hierarchy for triangle clouds. These volumes can control their aspect ratio to more tightly bound contained geometry than sphere bounds, which provided faster distance queries.

More recently, the distance methods for convex model distance queries have been applied to convex decompositions of triangular models[12]. Essentially, this method reduces the number of leaf nodes by replacing triangles with convex sets.

For these general polygonal models, the predominant techniques create bounding volume hierarchies, and the advancements have come mostly from improving the tightness of the bounding volumes. This approach differs markedly from techniques used for parametric models.

Parametric Models

Parametric models are composed of smooth surface patches, and typical models have fewer primitives than polygonal models. The emphasis in research, then, has not been on efficient means of pruning large numbers of primitives. Instead, methods have explored various numerical techniques for quickly and reliably solving systems of equations derived from setting up minimum distance conditions between two parametric models[15][21].

The distance between two parametric models $\mathbf{F}(u,v)$ and $\mathbf{G}(s,t)$ can be computed by finding the shortest length vector of the difference function.

$$\bar{D}(u,v,s,t) = F(u,v) - G(s,t)\,. \tag{4}$$

In [16], a bound and subdivide scheme explicitly searched this four-dimensional space for a minimum length difference vector. More commonly, distance minima are expressed as minima of the scalar valued distance squared function, and computed by finding coincident zeros from the set of its partial derivatives, as in

$$\begin{aligned} (\mathbf{F}-\mathbf{G})\cdot\mathbf{F}_u &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{F}_v &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{G}_s &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{G}_t &= 0 \end{aligned}\,. \tag{5}$$

The system of equations for distance extrema has also been variously defined as sets of cross-products[15] or augmented with explicit normal collinearity conditions[22]. These extrema conditions have been solved by employing symbolic computation[21], interval methods[23], and Newton-Rapheson iteration[24]. Having the advantage of high speed and rapid convergence, the latter has been a practical choice for many implementers.

Other Representations

Other model representations that provide unique capabilities have also been used. Implicit models provide easy intersections and Boolean operations. Since implicit models are the zero set of a scalar function, evaluating that function at a point in space is itself a type of distance function, even though it need not be an exact match to Euclidian distance.

CSG models are typically combinations of primitives, such as spheres, cones, cubes, and tori. Each primitive needs a custom distance function. Most CSG models are not made up of large numbers of primitives, so efficient pruning methods are not needed.

Haptic Rendering

Each proposed model representation has different trade-offs in terms of control of shape, surface smoothness, complexity of data structures, and memory requirements. These various trade-offs have been carefully studied for visual rendering of models. However, haptic rendering has a different set of requirements, namely, the update rate for haptic systems must be much higher than for visual systems. Typically, force computations must be updated at 1,000 times per second[25] for stable haptic rendering, whereas visual updates at 60Hz are generally considered adequate. However, since a user cannot touch the whole environment at once, haptic interaction is much more local than for visual rendering. These different requirements mean these representational trade-offs may have different impacts on a haptic rendering system than on a visual one. Thus, haptic rendering must develop its own distinct set of techniques to take best advantage of a model representation.

Haptic rendering computes the restoring forces needed to generate a sense of contact with a model. These restoring forces typically depend on the depth of penetration of the virtual hand into the model and the direction to apply the restoring force. Because of the difficulty of computing contact between complex models, the virtual hand in the environment is often represented as a point or collection of points called the end effector points[25]. In this thesis, the term *query points* will be most often used instead of that robotics specific term.

Computing the depth of penetration now reduces to finding the closest point on the model to the query points. As the virtual hand moves, this closest point changes and must be updated at haptic rates. Various techniques have been developed that compute and track the closest point on the model for different model representations. Below, we outline the development of haptic rendering methods from early stateless volume approaches to more current surface boundary techniques.

An early approach for determining the restoring force filled the modeled object with a vector field corresponding to the desired restoration force[26]. Typically, the interior of the model was subdivided into regions with a common direction and containing force vectors with lengths proportional to the distance to the surface. Yet this approach poses several difficulties as noted in [27]: creating the vector field is non-trivial for complex shapes, force discontinuities may occur when crossing internal field boundaries, and thin objects do not have enough depth to allow for an adequate force vector field. In the worst case, where the virtual hand penetrates too deeply, the volume method may accelerate the user from one side of the object to the other. Because of these difficulties, other techniques like boundary methods have largely supplanted volume approaches.

The *intermediate plane*[25] is an early type of boundary representation for haptic rendering. The intermediate plane approximates the local, underlying geometry of the model with a single plane. This plane updates as the virtual hand moves, usually much slower than the haptic update rate. This method maintains high update rates for the force computation by decoupling the cost of the depth computation from the complexity of the

underlying geometry and is also suitable for distributed computations, an especially attractive feature when the haptic controller runs on specialized hardware.

This time-varying approximation of the surface works best for low curvature surfaces; otherwise, noticeable force discontinuities may occur. An approach for alleviating these discontinuities "fades-in" the computed force[8]. The intermediate plane approach was most recently applied to NURBS surfaces[28]. Importantly, using an intermediate plane does not eliminate the need to track the closest point on the surface; rather, it simply alleviates the high haptic rate demands. As such, recent efforts have focused on speeding the more fundamental closest point tracking algorithms so that they may apply directly[11], rather than continuing efforts to mask some of the deficiencies of the intermediate plane approach.

Haptic Rendering of Polygonal Models

Polygonal models are the first of the true boundary surface representations we consider. Much work in computer graphics and robotics has focused on polygonal models; the haptics community has leveraged this research to good effect. Polygonal models are attractive because they readily lend themselves to fast computations[20]. In addition, current graphics displays essentially force surfaces to be converted to polygons, so polygonal haptic approaches share a common representation with the visual display.

Zilles and Salisbury[27] developed an approach for haptic rendering of simple polygonal models. Some history of the haptic interaction, they argue, avoids the problems related to the volume rendering methods mentioned earlier. Their haptic rendering method tracked the closest point on the surface with a simple method of determining collision with the facets of the model, an approach that limited model size to a few

hundred polygons. Ruspini[10] employs a more advanced collision detection method[14] to increase model size to tens of thousands of polygons and allowed a spherical end effector, instead of merely a point. He adopts the term *proxy* to refer to the constrained surface point that maintains the closest point. A competing method [29] combines spatial decomposition and oriented-bounding boxes to efficiently test intersection of a model with the end effector motion vector.

Complex interactions are difficult to model with a simple point end effector. As computing power has increased and haptic devices have improved, more general model-model interactions have become possible.

An extension to a general collision detection and response system[30] to haptic environments allowed the moving model and the virtual environment to be composed of the union of convex polygonal models. The computational burden of the collision method limits the scene to tens of polygons, but interactions using this extension are richer then with single query point methods.

More recently, research has looked at collections of convex bodies[31], as well as incremental methods for computing the penetration depth[32]. Most recently, the convex decomposition approach has been extended with perceptual level of detailing to accelerate haptic rendering for larger models[33].

Although limited as a surface representation due to compactness and smoothness concerns, these polygonal methods currently dominate haptic rendering. Their simple structure facilitates the development of fast algorithms for contact and depth computation as needed for haptic systems.

Sculptured Surfaces Methods

Sculpted surfaces represent smoothly curved surfaces in a natural way, thus avoiding some of the difficulties associated with polygonal representations. In addition, they are often more compact than a high-resolution polygonal model, so more complex environments can be accommodated in comparable computer storage space.

The success and ubiquity of NURBS in CAD and graphics indicates NURBS as the surface representation of choice for precise shape control (along with the more general subdivision surfaces representation). Advantages of NURBS include compact representation, higher order continuity, and exact computation of surface normals. For haptics, a further advantage is being able to directly manipulate CAD models without first having to create a polygonal representation.

Some review of NURBS terminology is appropriate for our discussion. NURBS surfaces are piecewise-polynomial vector-valued functions of two parametric variables that form the domain of the surface. The *control mesh* influences the shape of the surface and each *control point* of the control mesh provides a vector coefficient for the basis functions of the surface. Each *polynomial piece* of the surface is influenced by a local set of control points and the convex hull of that set completely contains that piece. The "parametric nodes" of the surface are readily computable first-order approximations to the parametric value of the closest points on the surface to each control point. Through a process known as refinement, which embeds the surface into a new, higher-dimensional parameter space, more degrees of freedom can be added to the control mesh of the surface. One can add trimming curves to NURBS to represent holes and other sharp surface boundaries that do not fall along the parametric directions[34].

As seen from this synopsis, the mathematics for NURBS is considerably more complex in comparison to the underlying mathematics for polygonal models. Real-time computations using NURBS can seem formidable. Indeed, in the past, simpler representations such as polygons or intermediate planes derived from a NURBS surface seemed necessary for haptic rendering[35]. However, algorithmic advances, together with inexorable improvements in computing power, have made direct computation on NURBS models possible[11].

Some of the lessons from polygonal haptic rendering methods apply to techniques for NURBS surfaces. At a high level, the proxy point transforms a global minimum distance solution to a local, continuous solution so that the restoring force direction remains continuous. On a NURBS surface an analogous situation exists. Haptic rendering needs a solution that is in the local neighborhood of the previous time step's solution. Local root finding methods, given an initial starting point, should converge to the nearest root, and these local root methods encapsulate many of the qualities of the proxy point. Thus, assuming the necessary components can be computed quickly enough, they are appropriate for tracing along a NURBS surface. This dichotomy between global and local closest points also suggests an approach for haptic rendering of NURBS surfaces – one could use global closest point methods to provide initial starting points and local closet point methods for tracking the surface point when the query point moves inside the model.

Discussion

Haptic rendering has progressed quickly from rather simple models such as cubes and spheres to quite general models made from tens of thousands of polygons or full-

featured, trimmed, NURBS models. This dissertation presents a number of new distance computations for both polygonal and parametric models that increase the complexity and robustness of haptic interaction with complex environments.

CHAPTER 3

DISTANCE TO PARAMETRIC MODELS

As discussed in the background section, the geometric approaches for computing the minimum distance between polyhedral objects differ markedly from the numerical approaches used for sculptured models. The material in this chapter bridges the background chapter and upcoming research chapters by delving more deeply into distance to parametric models, with the goal of developing intuition and definitions that will be used in later chapters. The following sections deal with models defined by parametric NURBS curves and surfaces. In all our discussion of sculptured models, we assume the model is a regular curve or surface with at least C^2 continuity, although the approaches can be adapted to piece-wise lower continuity models by subdivision.

Distance from a Point to a Curve

The vector difference between a point in space **P** and a point $\boldsymbol{\gamma}(t)$ on a planar space curve (Figure 1.A), is

$$\bar{\mathbf{D}}(t) = \boldsymbol{\gamma}(t) - \mathbf{P}\,. \tag{6}$$

For a fixed **P** and varying t, the vector difference can be thought of as a sequence of vectors from **P** to each point on the curve $\boldsymbol{\gamma}(t)$ (Figure 1.B).

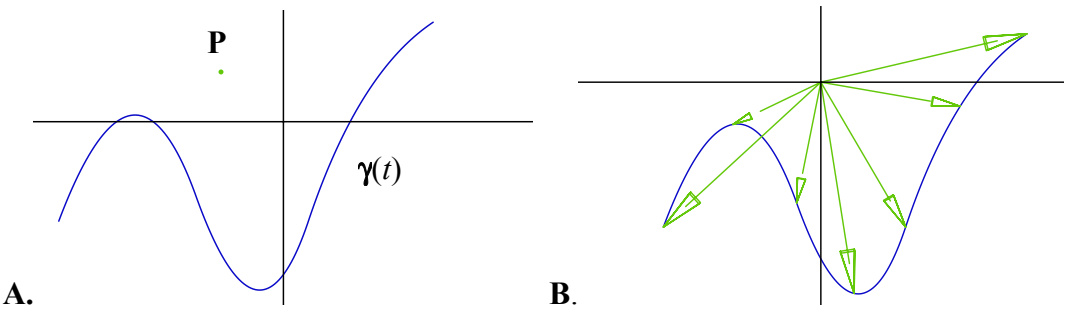


Figure 1: The difference between a point and curve is a vector valued function. (A) The point P and a curve. (B) The sequence of vectors between the origin and the curve translated by –P.

The distance between **P** and the curve can now be expressed as a parametric function,

$$D(t) = \left\| \left(\boldsymbol{\gamma}(t) - \mathbf{P} \right) \right\|, \tag{7}$$

and minimizing *D(t)* finds the minimum distance. Since *D(t)* can be rewritten as

$$D(t) = \left(\left(\boldsymbol{\gamma}(t) - \mathbf{P} \right) \cdot \left(\boldsymbol{\gamma}(t) - \mathbf{P} \right) \right)^{\frac{1}{2}}, \tag{8}$$

the distance squared,

$$D^2(t) = \left(\boldsymbol{\gamma}(t) - \mathbf{P} \right) \cdot \left(\boldsymbol{\gamma}(t) - \mathbf{P} \right), \tag{9}$$

shares common extrema parameters with *D(t)* and avoids the square root. The distance-squared function for the example given in Figure 1 is shown in Figure 2.

The minimum of $D^2(t)$ can be found by computing all its extrema and choosing the smallest. Extrema of $D^2(t)$ occur when the derivative is zero, as in

$$\frac{dD^2(t)}{dt} = 2\left(\boldsymbol{\gamma}(t) - \mathbf{P} \right) \cdot \frac{d\boldsymbol{\gamma}}{dt} = 0. \tag{10}$$

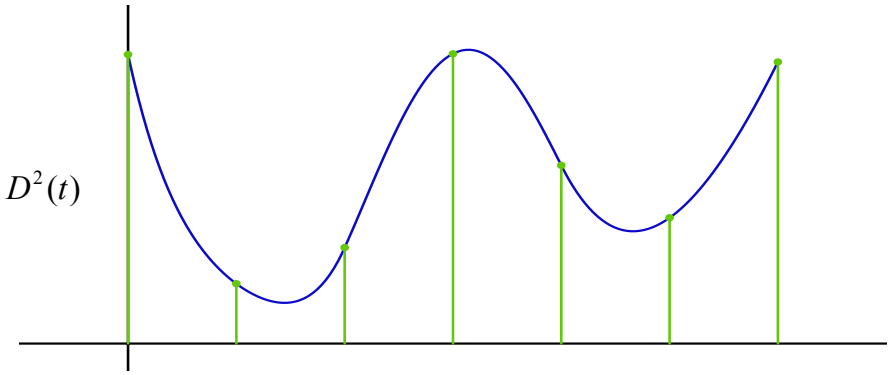


Figure 2: The function $D^2(t)$, with heights shown at the same *t* values used for the sample vectors in Figure 1.

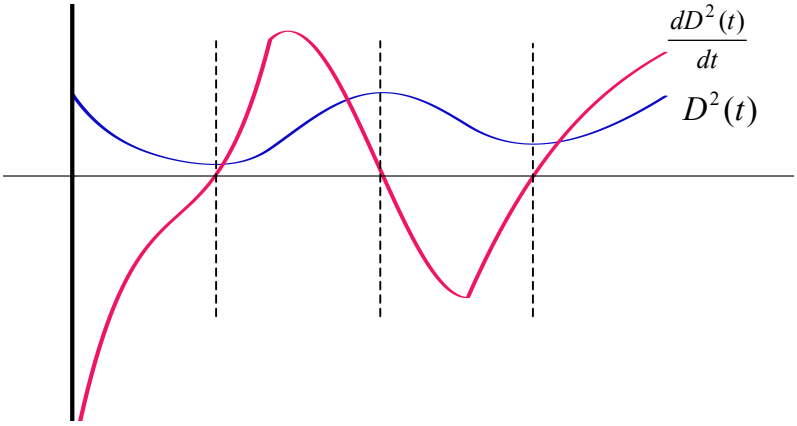


Figure 3: The derivative of $D^2(t)$ overlaid on $D^2(t)$. Zero crossings correspond to extrema in distance.

In the general case, extrema also occur at endpoints of the curve or at non-differentiable points. A curve with tangent discontinuities can be split into multiple curves, each of which is considered independently. The endpoints of the curve can also be checked as a special case. For now, we concern our analysis to the interior of a model.

Figure 3 shows the Eq. 10 superimposed on the squared distance function of Figure 2. The parameter values along the x-axis where the derivative curve crosses zero correspond to minima or maxima of the squared distance function.

The factor of two on the right-hand side of Eq. 10 is irrelevant to the finding of zeros. Thus, distance extrema are found at zeros of the simplified extrema equation

$$E(t) = \left(\boldsymbol{\gamma}(t) - \mathbf{P}\right) \cdot \frac{d\boldsymbol{\gamma}}{dt}. \tag{11}$$

This formulation shows that distance extrema occur at orthogonal projections of **P** onto the curve, which is where the projection vector is at right angles to the tangent at the projected point. Another way to think of this is that the extremal point normal must be collinear with the vector between the extremal point and the query point. This collinearity condition will be the basis for many of the techniques developed in this thesis.

Extremal Distance

The collinearity condition of Eq. 6 makes no distinction between a local minimum and a local maximum of the squared distance function. The maximum is also known as the extremal distance[1]. There is a geometric relationship between the curvature of the curve, $\kappa(t)$, at the extremum, the distance to the extremum, and the side of the curve on which the query is made that determines whether the distance is a minimum or a maximum (Figure 4).

When the curve at the extremum is convex relative to the query point, then the extremum is always a local minimum, since the curve in this region bends away from the query point. When the curve at the extremum is concave relative to the query point, then it bends towards the query point. If the radius of curvature, $\frac{1}{\kappa(t)}$, at the extremum is larger than *D(t)*, then none of the local region of curve is nearer than the extremum, so it is a minimum. If the radius of curvature is smaller than the distance, then the curve bends

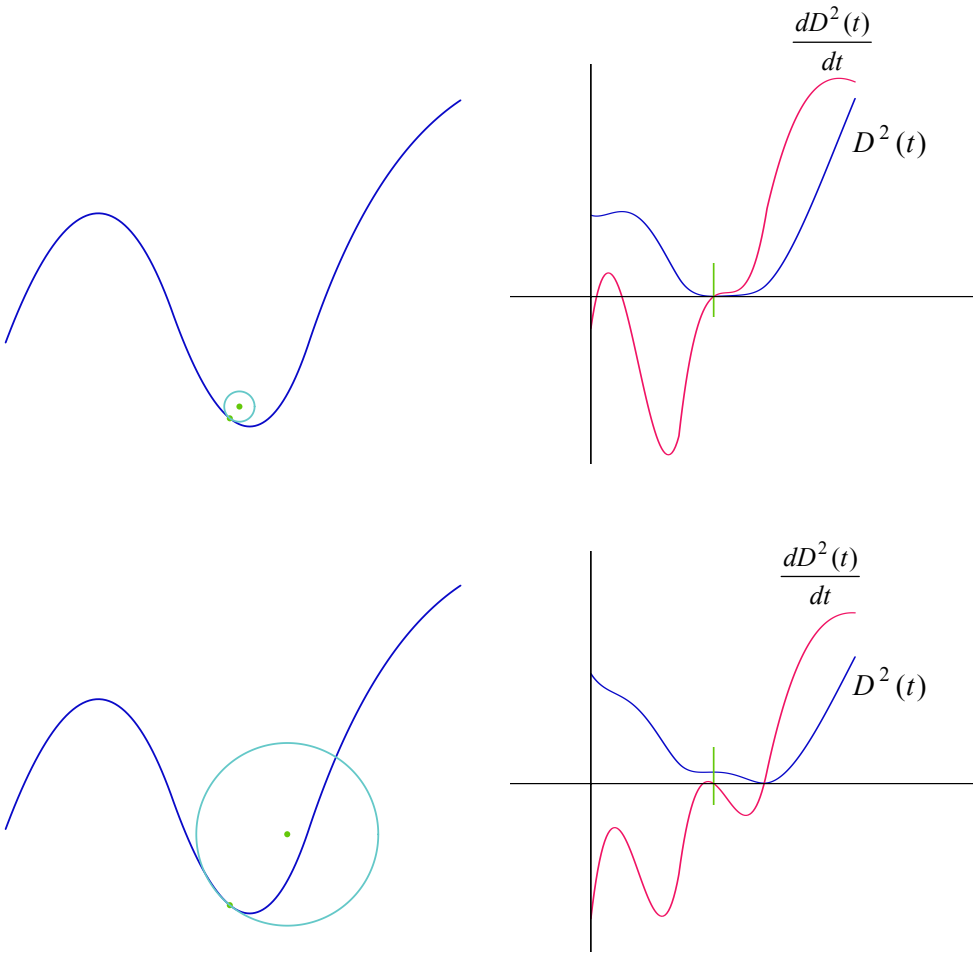


Figure 4: As the query point moves away from the curve (left column), the squared distance function (right column) changes from a minimum to an extremal distance at the same zero crossing of the derivative function.

inside the distance bound, and the extremum is a local maximum. These conditions are illustrated in Figure 4. Notice how a circle centered at the query point of radius equal to the extremal distance just touches the curve at the extremal point (top row of Figure 4), with the rest of the local curve further away, making a local minimum. As the query point moves away from the curve, the circle expands. When the query point is further away then the radius of curvature, then the circle expands past the local curve (bottom row of Figure 4), implying that the curve on either side of the solution is closer and the solution is a local maximum.

Distance from a Point to a Surface

The minimum distance between a point in space, $\mathbf{P}$, and a bivariate parametric surface $\mathbf{S}(u,v)$ (Figure 5) is the minimum of the distance function

$$D(u,v) = \left\|\mathbf{S}(u,v) - \mathbf{P}\right\|. \tag{12}$$

Following the approach for computing the minimum distance between a point and a curve, the distance squared to a surface

$$D^2(u,v) = \left(\mathbf{S}(u,v) - \mathbf{P}\right)\cdot\left(\mathbf{S}(u,v) - \mathbf{P}\right) \tag{13}$$

shares the same parameters at extrema as the distance and has a less complex formulation. In Figure 5, local distance minima between the query point and the surface are visible in the visualization of the distance squared function as two bumps, each corresponding to a local closest point on the original surface to **P**.

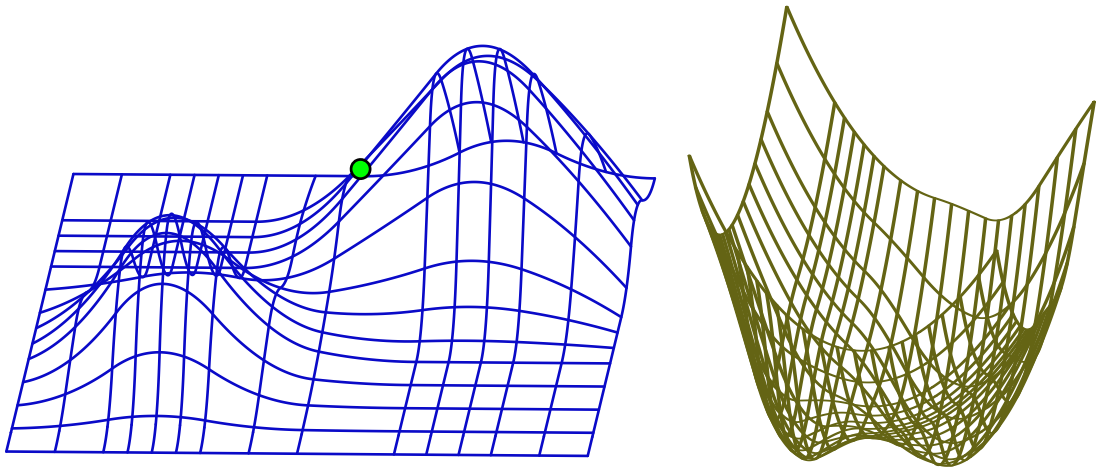


Figure 5: The squared distance between the point and surface at left is visualized on the right as a function mapping parameter value vs. squared distance.

The distance-squared function generates a system of equations that are satisfied at an extremum.

$$\begin{aligned}\left(\mathbf{S}(u,v)-\mathbf{P}\right)\cdot\mathbf{S_u}&=0\\ \left(\mathbf{S}(u,v)-\mathbf{P}\right)\cdot\mathbf{S_v}&=0\end{aligned} \qquad \textbf{(14)}$$

This system is an analogue to the extrema equation for minimum distance to a curve, as it shows that the closest point on the surface is also an orthogonal projection of the query point onto the surface.

These partial derivatives are complex three-dimensional surfaces and difficult to visually understand. However, mapped as $(u,v,\frac{\partial D^2(u,v)}{\partial u})$ and $(u,v,\frac{\partial D^2(u,v)}{\partial v})$, their zero crossings generically form curves in the *uv* plane (Figure 6), and the intersections of each zero set are solutions to Equation 8.

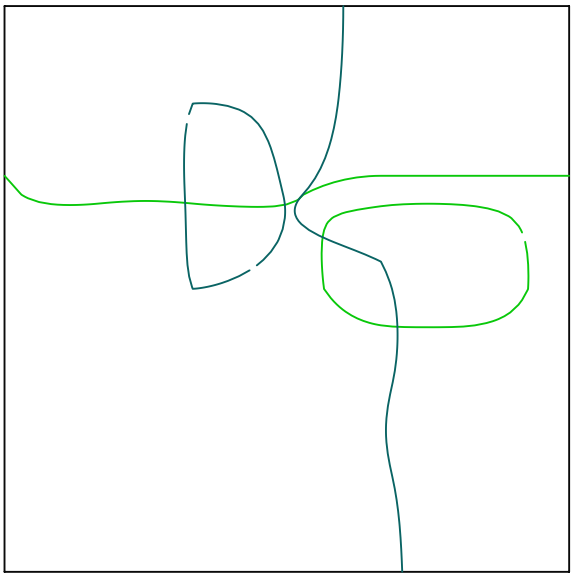


Figure 6: The zero crossings of each partial of the distance-squared function generically form curves in the *uv* plane. The intersections between the two curves are local extrema of the distance-squared function.

Distance Between Two Surfaces

Distance queries between surfaces extend naturally from the query for a point to a surface. The distance function for two surfaces $\mathbf{F}(u,v)$ and $\mathbf{G}(s,t)$ is

$$D(u,v,s,t) = \left\|\mathbf{F}(u,v) - \mathbf{G}(s,t)\right\|. \tag{15}$$

As before, distance extrema occur at zeros of the set of partial differentials of the squared distance

$$\begin{aligned} (\mathbf{F}-\mathbf{G})\cdot\mathbf{F}_u &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{F}_v &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{G}_s &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{G}_t &= 0 \end{aligned} \tag{16}$$

However, Equation 16 does not naturally express the possible configurations two surfaces can have relative to each other. Two surfaces can interpenetrate each other, and an extremum of distance is at the maximum penetration. While this extremum is a valid root for Equation 16, another set of roots occurs along the curve of intersection between the two surfaces during interpenetration, where $(\mathbf{F}-\mathbf{G})$ is zero. Numerical methods will tend to find a solution in the set of roots associated with this intersection curve rather than the extremum in distance. Defining the distance as an extremal distance, rather than the minimum distance, helps to avoid these problems.

Extremal Distance Formulation

Following [1], the extremal distance can be defined as the minimum distance between the two models when they are disjoint, zero during tangential contact, and the

locally maximum penetration depth when they interpenetrate. This measure reflects the possible configurations of two surfaces more accurately than the minimum distance.

The extremal distance between parametric surfaces $\mathbf{F}(u,v)$ and $\mathbf{G}(s,t)$ is the following scalar valued equation:

$$E(u,v,s,t) = \left(\mathbf{F}(u,v) - \mathbf{G}(s,t)\right) \cdot \vec{\mathbf{N}}(u,v). \tag{17}$$

Extrema of Equation 17 are at simultaneous roots of its partials, which are

$$\begin{aligned} \mathbf{F}_u \cdot \vec{\mathbf{N}} + \left(\mathbf{F} - \mathbf{G}\right) \cdot \vec{\mathbf{N}}_u &= 0 \\ \mathbf{F}_v \cdot \vec{\mathbf{N}} + \left(\mathbf{F} - \mathbf{G}\right) \cdot \vec{\mathbf{N}}_v &= 0 \\ -\mathbf{G}_s \cdot \vec{\mathbf{N}} &= 0 \\ -\mathbf{G}_t \cdot \vec{\mathbf{N}} &= 0 \end{aligned} \tag{18}$$

This can be simplified by noting that the normal $\vec{\mathbf{N}}$ is always orthogonal to the tangent plane formed by the partials $\mathbf{F}_u$ and $\mathbf{F}_v$, so the $\mathbf{F}_u \cdot \vec{\mathbf{N}}$ and $\mathbf{F}_v \cdot \vec{\mathbf{N}}$ terms are always zero and may be removed. Additionally, the partials of $\vec{\mathbf{N}}$ lie in the tangent plane of $\mathbf{F}(u,v)$, so replacing the partials of $\vec{\mathbf{N}}$ with the partials of $\mathbf{F}(u,v)$ forms an equivalent constraint. These substitutions form this simplified set of equations

$$\begin{aligned} \left(\mathbf{F} - \mathbf{G}\right) \cdot \mathbf{F}_u &= 0 \\ \left(\mathbf{F} - \mathbf{G}\right) \cdot \mathbf{F}_v &= 0 \\ \vec{\mathbf{N}} \cdot \mathbf{G}_s &= 0 \\ \vec{\mathbf{N}} \cdot \mathbf{G}_t &= 0 \end{aligned} \tag{19}$$

The first two partials constrain the solution line to lie along the normal of $\mathbf{F}(u,v)$, and the second two maintain collinearity of the two surface normals. An intersection of

the two surfaces, where $(\mathbf{F}-\mathbf{G})$ is zero, no longer fully satisfies the set of partials, and is not a solution to the extremal distance formulation.

Discussion

This chapter develops sets of equations for the minimum distance between points and curves, points and surfaces, and two surfaces, as well as a special set of constraints for the extremal distance between two surfaces. The next chapters delve into techniques for solving these equations using numerical and geometric methods, eventually applying them to haptic rendering.

CHAPTER 4

THE SCALED EVOLUTE BOUND FOR RELIABLE CONVERGENCE OF POINT-CURVE MINIMUM DISTANCE QUERIES

Minimum distance queries between a point and a curve can be computed symbolically for curves of low degree[21]. For higher degree curves, a mixture of symbolic and numerical computation can be used. In many cases, a local solution is desired, and purely numerical approaches are feasible. This last approach also lends itself to rapid computation.

This chapter analyzes a convergence condition of Newton's method, a standard numerical approach, for minimum distance queries between a point and a curve. Based on that analysis, we develop an algorithm for computing a set of starting parametric values and an associated geometric bound on query point location that ensures convergence of Newton's method during minimum distance queries. We restrict our analysis to planar, twice-differentiable, regular curves. An example of this type of curve is a planar, B-spline curve, commonly used in CAD, modeling, and animation, so this restriction is not onerous.

Because this chapter requires significant background and mathematical analysis before developing the main argument, we summarize the approach here:

1. Convergence of Newton's method depends on its initial parameter and the query point.
2. A geometric interpretation of a convergence condition for Newton's method yields safe spatial regions for point queries relative to the current estimated closest point.
3. These regions are largely dependent on the radius of curvature at that point on the curve. By properly computing sets of starting parameters, the convergence condition will hold for all query points to one side of a scaled evolute curve and Newton's method will converge.

This result is different from the typical analysis of the convergence properties of Newton's method. In point estimation theory[36], convergence conditions are derived for a particular query point relative to a curve, based on the initial parameter value. Our approach determines for which families of curves (corresponding to different query points) will a set of initial parameter values provide adequate assurance of convergence. This approach allows the initial computational cost of analysis to be amortized over many queries, and to know a priori whether a minimum distance query will, in fact, converge.

Newton's Method for Minimum Distance Queries

Newton's method is a standard approach for local root finding, especially when derivative information is readily available[37]. For a univariate parametric function *F(t)*, Newton's method solves for a change in *t*, such that iterations of Newton's method should converge to a root. The new parameter value depends on the previous iteration, the function value, and its derivative at the prior parameter value, specifically

$$t^{j+1} = t^{j} - \frac{F(t^{j})}{F'(t^{j})}, j = 0,1,2... \quad \textbf{(20)}$$

First, it should be noted that there is no guarantee that Newton's method will find a root. It may diverge due to poor starting conditions; cycle, rather than converge; or just fail when $F'(t) = 0$. Additionally, Newton's method may find a root, but not one near the starting parameter. In this case, the method essentially diverges for a step or more, and then happens to land in the basis of attraction for another root.

Careful choice of parameter value to initialize the Newton iteration helps avoid these problems. For minimum distance queries, starting parameters can be generated as needed[24], or precomputed. In neither case do these methods provide any assurances that Newton's method will converge as desired. For precomputed starting parameters, a standard algorithm[34] evaluates the curve at multiple evenly spaced potential starting parameters, or *seed points*. A *seed parameter, t,* is a parameter value used to initialize Newton's method. A particular seed parameter out of an ordered set of seed parameters is indicated by t_i. A seed parameter also defines a *seed point*, $\boldsymbol{\gamma}(t_i)$.

The standard distance algorithm

1. computes a set of evenly spaced seed parameters on the curve,
2. finds the closest seed point to the query point,
3. solves for a root of the extrema equation (Eq. 11) using Newton's method initialized with the seed parameter corresponding to the nearest seed point,
4. returns the distance between the query and $\boldsymbol{\gamma}(t)$, where *t* is the root of the extrema equation from step 3.

However, there are no guarantees that Newton's method will actually converge given a particular starting seed parameter. This shortcoming motivates this chapter's

development of an algorithm for generating seed parameters with known convergence properties for an associated spatial region of possible query points.

Visualizing the Convergence of Newton's Method

A distance map is an image that represents the distance from the center of a pixel to an object of intekrest by pixel intensity. The top of Figure 7 shows a distance map for a sample curve. Another useful mapping is to associate a gradient of color with the curve parameter, so that the start of the curve is associated with black, the end with white, and the portion in the middle a linear interpolation between the two. With this mapping the intensity of the pixel in the image is proportional to the parametric value of the closest point on the curve (Figure 7 bottom), thus we refer to it as a *closest point map*.

An analogous idea provides visual evidence for when the standard algorithm succeeds or fails. Calling the standard algorithm at every pixel and using its distance result, rather than the correct distance used in Figure 7, provides a visual representation of its success. Furthermore, since Newton's method can detect when it fails to converge, that result can be shown in a warning color. Figure 8 shows distance maps using the standard algorithm with varying numbers of seed points. Figure 9 shows the equivalent closest point maps.

These images clearly show that an inadequate number of seed points on the curve can result in unreliable convergence of Newton's method. However, they also show that even when there are large numbers of seed points, troublesome, albeit small, regions of divergence remain.

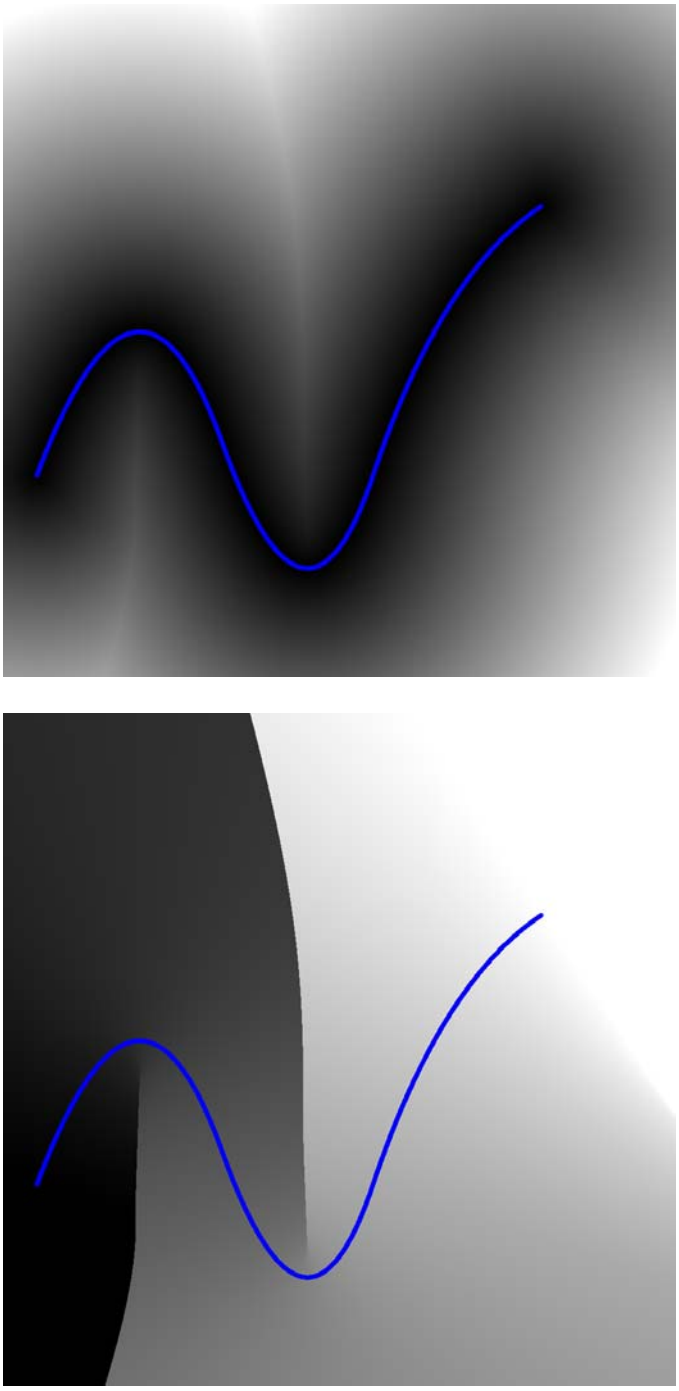


Figure 7: Two different distance visualizations. (top) A distance map for a sample curve. (bottom) A closest point map for the same curve.

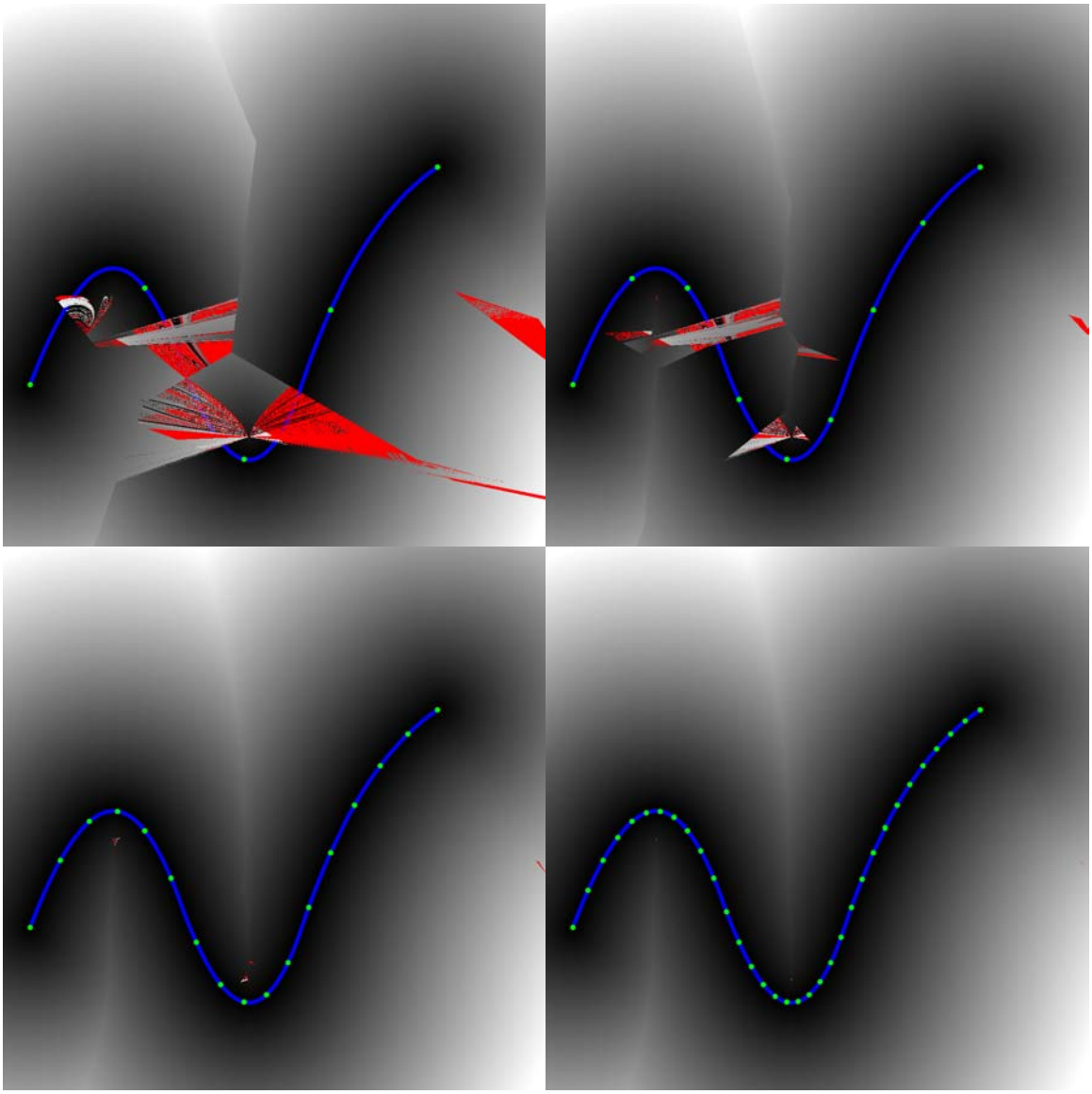


Figure 8: Distance maps show distance to the curve by mapping pixel intensity to distance. The number of seed points is increased in each successive image, and places where Newton's method fails to converge map to a warning color.

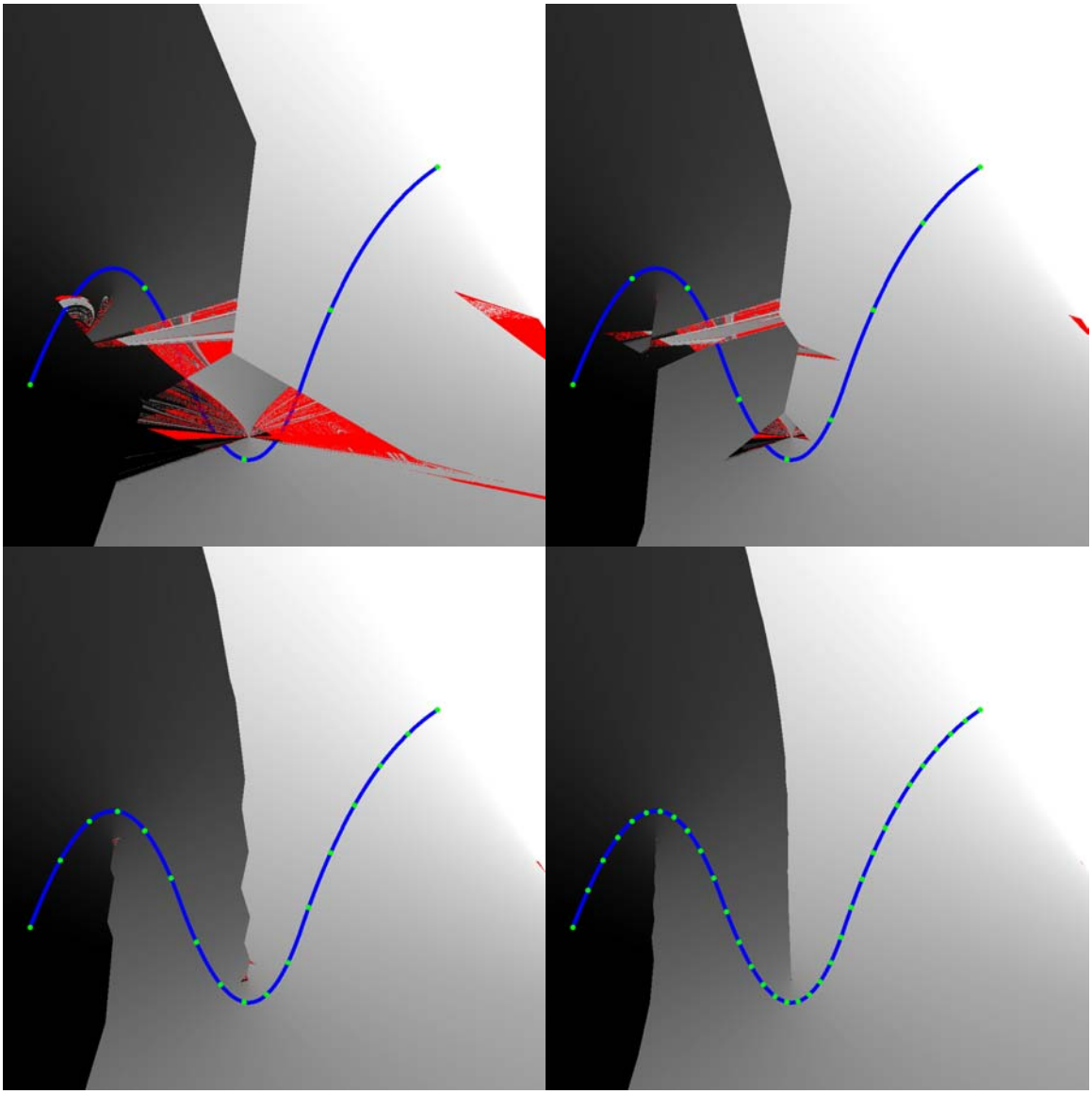


Figure 9: Closest point maps set pixel intensity to the parametric value of the closest point on the curve. This mapping provides more information about where the standard algorithm converged to than the distance map.

Convergence of Distance Queries Between a Point and Curve

The convergence of numerical methods in general and Newton's method in particular has been thoroughly analyzed. In this section, a convergence condition from the literature[37] is studied in detail. A geometric interpretation of those conditions provides a basis for an algorithmic means of generating a set of seed points on a specified curve with known convergence properties for regions of query points.

Eq. 20 can be rewritten as

$$G(t) = t - \frac{F(t)}{F'(t)}. \tag{21}$$

Newton's method will converge when for each iteration[37],

$$\left|G'(t)\right| < 1. \tag{22}$$

Using the quotient rule for differentiation on Eq. 21 and then simplifying,

$$G'(t) = \frac{FF''}{F'^2} \tag{23}$$

When looking at the specific problem of minimum distance, where the function to be minimized is $E(t)$ (from Eq. 6), the convergence condition becomes

$$G'(t) = \frac{EE''}{E'^2}. \tag{24}$$

Derivatives of the Distance Extrema Equation

The functions for distance extrema and its derivatives are needed to expand the convergence condition. *E(t)* in dot product bracket notation (and without explicit parameters) is

$$E = \langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}' \rangle, \tag{25}$$

and, by the chain rule, its first and second derivatives are

$$E' = \langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}'' \rangle + \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle, \tag{26}$$

$$E'' = \langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}''' \rangle + 3\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle. \tag{27}$$

Using these derivatives, each step of Newton's method (Equation 21) expands to

$$\Delta t = -\frac{\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}' \rangle}{\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}'' \rangle + \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle} \tag{28}$$

and the value of G' (Eq. 24) used in the convergence condition for Newton's method when used for the extrema distance equation is then

$$G' = \frac{\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}' \rangle \left[\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}''' \rangle + 3\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \right]}{\left[\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}'' \rangle + \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle \right]^2}. \tag{29}$$

Visualizing the Convergence Conditions for Newton's Method

A technique similar to a distance map can be used to visualize the value of G' during application of the standard algorithm. Rather than mapping distance to pixel intensity, the visualization technique maps the value of G' at the initial seed, based on the seed parameter and the query point. Magnitudes of G' between zero and one map to image intensity between black and white. Magnitudes greater than or equal to one, where Newton's method is not expected to reliably converge, map to a warning color (Figure 10). This visualization helps us understand convergence properties during the initial step of Newton's method. During later steps, new G' values would be associated with the current parameter value of the estimated closest point.

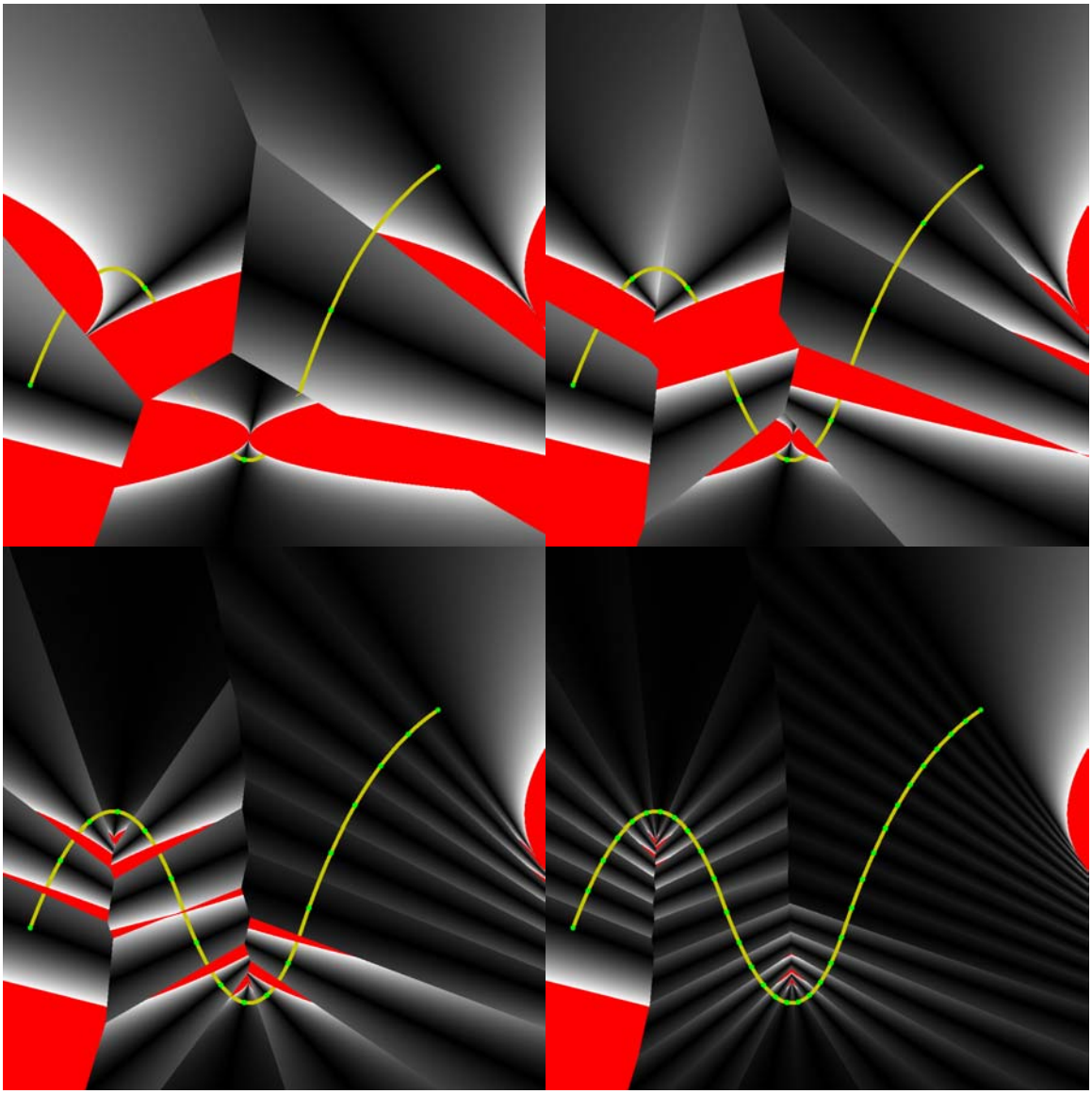


Figure 10: Regions of $|G'| \geq 1$ are easily visualized by finding the closest seed point on the curve to each pixel and computing the value of G' based on the intrinsic properties of the curve. Magnitudes larger or equal to one are colored. The images above show visualizations of G' based on varying numbers of seed points.

Analysis of Degenerate Conditions

These magnitude images of G' reveal several features that are worth exploring mathematically. An analysis of these conditions will eventually yield a more tractable form of Equation 29.

The worst case for G' is when it is degenerate, and our analysis starts there. The degeneracy condition for G' is when its denominator is zero. Thus when

$$\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}'' \rangle = -\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle, \tag{30}$$

G' is degenerate. Note that this is also the same condition where Newton's method (Equation 28) becomes degenerate. For a fixed value of t, Equation 30 determines a line, defined by the vector from $\mathbf{P}$ to $\boldsymbol{\gamma}(t)$ projecting onto the normalized $\boldsymbol{\gamma}''(t)$ with the same magnitude as the length of $\boldsymbol{\gamma}'(t)$ squared. A query point on this line produces a degeneracy in G', for the parameter used in that iteration of Newton's method.

This line of degeneracy forms the region's center backbone (Figure 11). The area where this line crosses the normal to the seed point warrants additional attention.

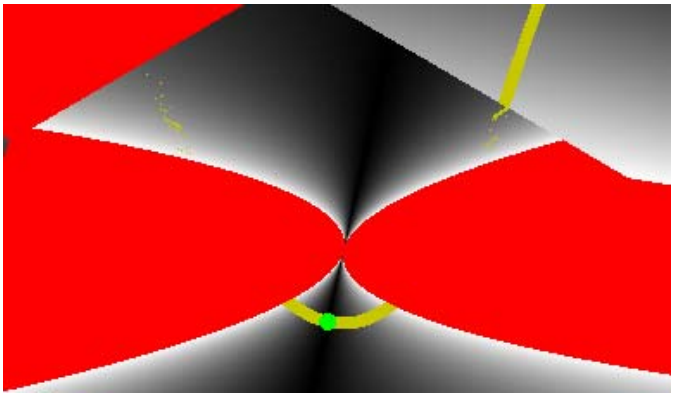


Figure 11: The area of $|G'| \geq 1$ for one seed point along the curve is shown. Some portions for other seed points are visible outside the Voronoi region of the seed point.

Restricting **P** to lie along the normal, **N**, at the current $\boldsymbol{\gamma}(t)$, as in

$$\boldsymbol{\gamma} - \mathbf{P} = y\mathbf{N}, \tag{31}$$

and substituting in Eq. 29,

$$G' = \frac{\langle y\mathbf{N}, \boldsymbol{\gamma}' \rangle \left[\langle y\mathbf{N}, \boldsymbol{\gamma}''' \rangle + 3\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \right]}{\left[\langle y\mathbf{N}, \boldsymbol{\gamma}'' \rangle + \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle \right]^2}, \tag{32}$$

immediately shows that the numerator of G' is always zero for this restricted placement of **P** along the normal, since the dot product of the normal and tangent in the first factor is always zero. This is also demonstrated in Figure 11, which shows a line of black (with 0 value) extending along the normal from the seed point. However, the degeneracy in the denominator still exists. By using a relation for curvature, κ, and the normal of a curve[38],

$$\begin{aligned} \kappa\mathbf{N} &= \frac{\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle \boldsymbol{\gamma}''}{\|\boldsymbol{\gamma}'\|^4} - \frac{\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \boldsymbol{\gamma}'}{\|\boldsymbol{\gamma}'\|^4} \\ \kappa\mathbf{N} &= \frac{\boldsymbol{\gamma}''}{\|\boldsymbol{\gamma}'\|^2} - \frac{\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \boldsymbol{\gamma}'}{\|\boldsymbol{\gamma}'\|^4} \end{aligned} \tag{33}$$

the denominator, d, of G',

$$d = \left[\left\langle y\mathbf{N}, \frac{\boldsymbol{\gamma}''}{\|\boldsymbol{\gamma}'\|^2} \right\rangle + 1 \right]^2 \|\boldsymbol{\gamma}'\|^4, \tag{34}$$

can be rewritten in terms of the curvature,

$$d = \left[\left\langle y\mathbf{N}, \kappa\mathbf{N} + \frac{\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \boldsymbol{\gamma}'}{\|\boldsymbol{\gamma}'\|^4} \right\rangle + 1 \right]^2 \|\boldsymbol{\gamma}'\|^4. \tag{35}$$

Again, since the first derivative and normal at a point on the curve are orthogonal, then

$$d = \left[\langle y\mathbf{N}, \kappa\mathbf{N}\rangle + 1\right]^2 \|\boldsymbol{\gamma}'\|^4 . \tag{36}$$

Since we are discussing regular curves, $\|\boldsymbol{\gamma}'\|$ is never zero, and only the first factor can go to zero, which occurs when

$$y = -\frac{1}{\kappa}. \tag{37}$$

This implies G' is degenerate and Newton's method fails when

$$\mathbf{P} = \boldsymbol{\gamma} + \frac{1}{\kappa}\mathbf{N}, \tag{38}$$

or when **P** lies at the center of the osculating circle at $\boldsymbol{\gamma}(t)$. This result, along with the line of degeneracy from Equation 30, is shown in Figure 12.

A Geometric Interpretation of the Unsafe Convergence Region

For reliable use of Newton's method, determining degenerate locations is not enough – all query point locations where the magnitude of G' is greater than or equal to one for all parameter values used by Newton's method need to be avoided. However, the style of analysis used to determine the degeneracy condition is applicable in this more general case as well.

The vector from **P** to $\boldsymbol{\gamma}(t)$ can be represented in terms of a reference frame at $\boldsymbol{\gamma}(t)$, formed by the tangent direction and normal at $\boldsymbol{\gamma}(t)$, as in

$$\boldsymbol{\gamma} - \mathbf{P} = x\boldsymbol{\gamma}' + y\mathbf{N} . \tag{39}$$

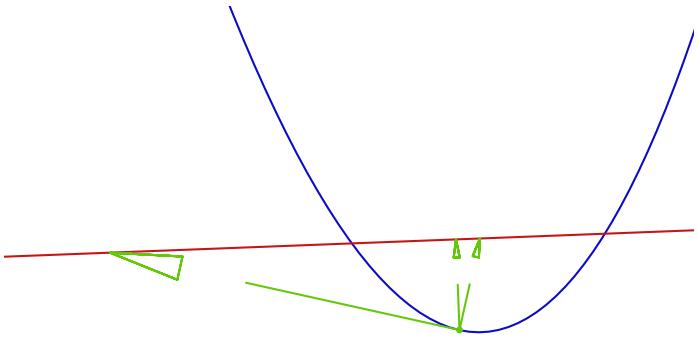


Figure 12: Placements of P that cause degeneracies in G' for a given parameter value lie along the line orthogonal to the second derivative.

Using this representation, G' becomes

$$G' = \frac{\langle x\boldsymbol{\gamma}' + y\mathbf{N}, \boldsymbol{\gamma}' \rangle \left[\langle x\boldsymbol{\gamma}' + y\mathbf{N}, \boldsymbol{\gamma}''' \rangle + 3\langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \right]}{\left[\langle x\boldsymbol{\gamma}' + y\mathbf{N}, \boldsymbol{\gamma}'' \rangle + \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}' \rangle \right]^2} \tag{40}$$

and expanding, yields

$$G' = \frac{x^2 |\boldsymbol{\gamma}'|^2 \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}''' \rangle + 3x|\boldsymbol{\gamma}'|^2 \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle + xy |\boldsymbol{\gamma}'|^2 \langle \mathbf{N}, \boldsymbol{\gamma}''' \rangle}{x^2 \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle^2 + y^2 \langle \mathbf{N}, \boldsymbol{\gamma}'' \rangle^2 + 2xy \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle \langle \mathbf{N}, \boldsymbol{\gamma}'' \rangle + 2x \langle \boldsymbol{\gamma}', \boldsymbol{\gamma}'' \rangle |\boldsymbol{\gamma}'|^2 + 2y \langle \mathbf{N}, \boldsymbol{\gamma}'' \rangle |\boldsymbol{\gamma}'|^2 + |\boldsymbol{\gamma}'|^4} \tag{41}$$

.

We call a **P** that yields a G' magnitude less than one for a given parameter a *safe position* of **P**. Then the curve representing the boundary between unsafe and safe positions of **P** is where

$$|G'(t)| = 1, \tag{42}$$

which forms two implicit curves defined by

$$\begin{aligned} G'(t) &= 1 \\ &\textit{or} \\ G'(t) &= -1 \end{aligned} \tag{43}$$

Choosing the positive condition forms a quadratic equation in x and y in the local tangent-normal reference frame

$$x^2\left[\langle\boldsymbol{\gamma}',\boldsymbol{\gamma}''\rangle^2-\langle\boldsymbol{\gamma}',\boldsymbol{\gamma}'''\rangle|\boldsymbol{\gamma}'|^2\right]+y^2\langle\mathrm{N},\boldsymbol{\gamma}''\rangle^2+xy\left[2\langle\boldsymbol{\gamma}',\boldsymbol{\gamma}''\rangle\langle\mathrm{N},\boldsymbol{\gamma}''\rangle-\langle N,\boldsymbol{\gamma}'''\rangle|\boldsymbol{\gamma}'|^2\right]+$$
$$x\left[2\langle\boldsymbol{\gamma}',\boldsymbol{\gamma}''\rangle|\boldsymbol{\gamma}'|^2-3\langle\boldsymbol{\gamma}',\boldsymbol{\gamma}''\rangle|\boldsymbol{\gamma}'|^2\right]+2y\langle\mathrm{N},\boldsymbol{\gamma}''\rangle|\boldsymbol{\gamma}'|^2+|\boldsymbol{\gamma}'|^4=0. \quad \textbf{(44)}$$

The implicit equation describing the negative boundary is similarly formed. Together, these two boundaries (Figure 13) form the outline of the regions of poor convergence from a single step of Newton's method, matching those from the G' visualization (Figure 11).

This form of the convergence condition is more amenable to computation than Eq. 29 and will be used in support of this chapter's motivating problem – how should seed points be distributed on the curve to support regions of safe positions of **P** for all steps of Newton's method during convergence?

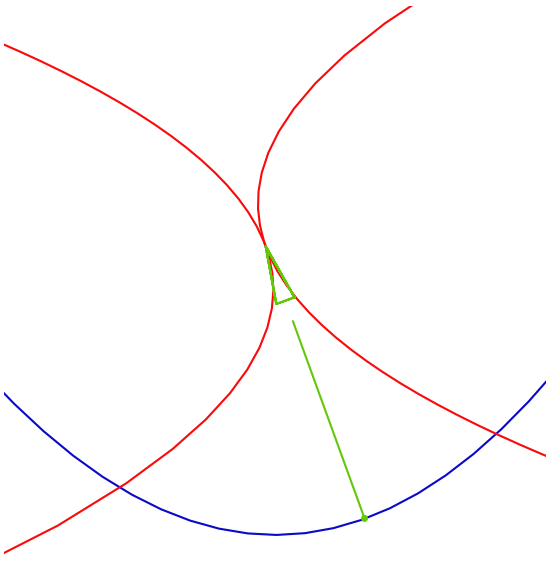


Figure 13: The implicit form of $|G'|=1$ may be easily graphed and used in computations, as opposed to the purely visual representation in Figure 11.

Computing Seed Points

Tools are now in place to obtain the goal in this chapter – namely, to compute a set of seed points that provides guaranteed convergence of Newton's method for query points inside a spatial bound. Constrained by the previously discussed degeneracy conditions, convergence for Newton's method is impossible to guarantee for all query points, independent of the number of initial seed points used. Instead, given the derived geometric interpretation of the convergence conditions, we develop techniques to compute seed points such that Newton's method will converge for all query points within a spatial bound that we call the *scaled evolute* bound.

Scaled Evolute Bound

As shown by Eq. 38, Newton's method becomes degenerate when the query point is at the center of the osculating circle at $\boldsymbol{\gamma}(t)$. The locus of points fulfilling this condition forms a possibly discontinuous curve called the curve *evolute* (Figure 14 left), defined as

$$\mathbf{R}(t) = \boldsymbol{\gamma}(t) + \frac{1}{\kappa(t)}\mathbf{N}(t). \tag{45}$$

Any spatial region bounding query points that includes any of this curve cannot guarantee avoiding degeneracy in Newton's method. Therefore, we use a *scaled evolute (SE)* curve that lies between the curve and its evolute (Figure 14 bottom), defined as

$$\mathbf{S}(t,\alpha) = \boldsymbol{\gamma}(t) + \frac{\alpha}{\kappa(t)}\mathbf{N}(t). \tag{46}$$

The parameter α, restricted to values between zero and one, determines the interpolation between the original curve and its evolute.

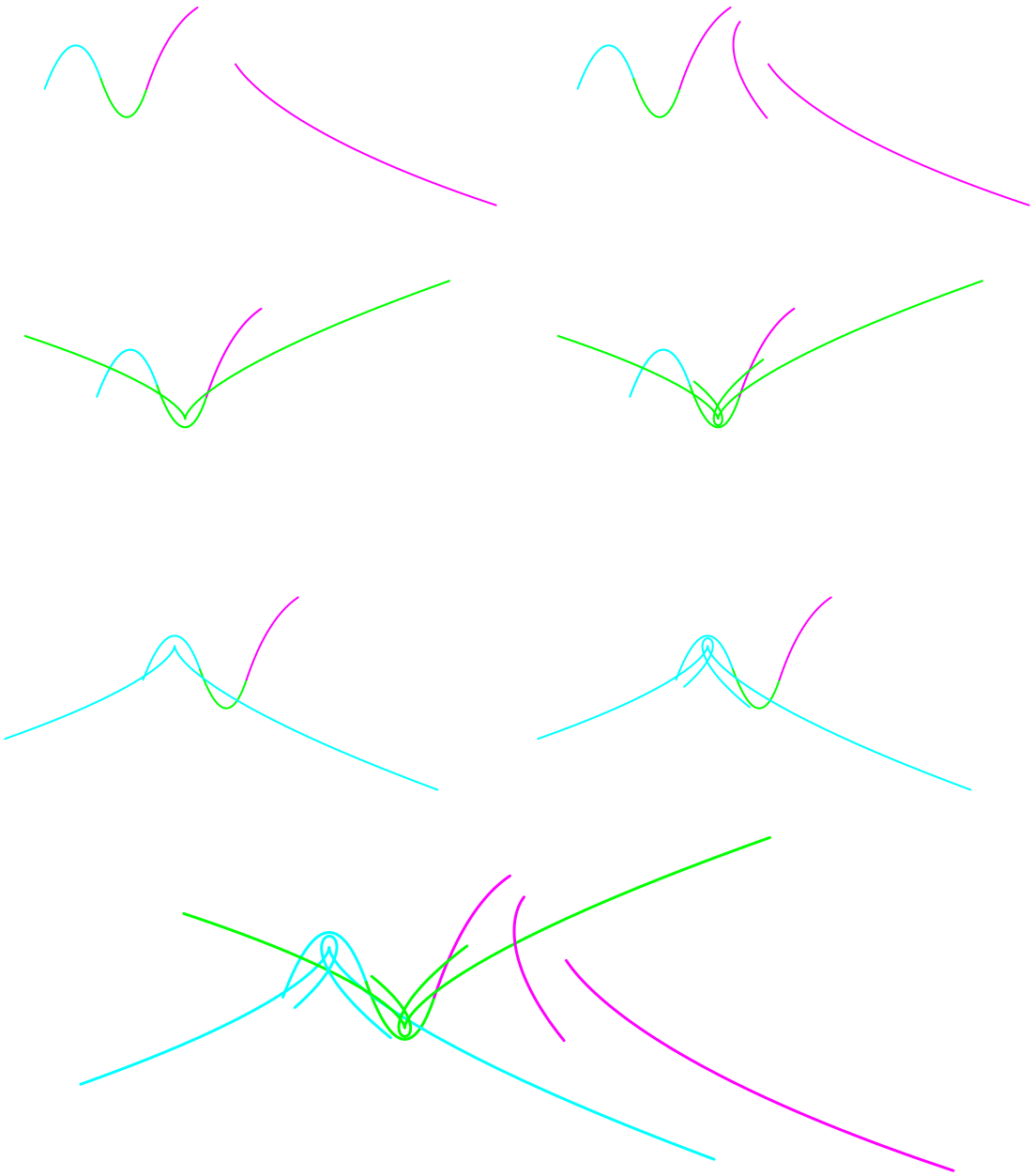


Figure 14: The curve and the three discontinuous parts of the evolute (left) and a scaled evolute in-between the curve and its evolute (right). The curve, entire evolute, and scaled evolute are shown at bottom.

Given the SE-curve, we bound query point locations to the disjoint regions bounded on one side by the SE-curve, and extending infinitely in the direction from the SE-curve to the original curve, or the bivariate (for a fixed α) planar regions defined by

$$\mathbf{B}(t,u) = \boldsymbol{\gamma}(t) + \frac{\alpha}{\kappa(t)}\mathbf{N}(t) - u\mathbf{N}(t), \quad u \geq 0\,. \tag{47}$$

These scaled evolute regions are shown in Figure 15.

One further subdivision of the scaled evolute regions will be useful. A *seed point interval (SPI) region* is a piece of the scaled evolute region bounded on each side by lines through neighboring seed points of the curve and parallel to the seed point normals (Figure 16), or just $\mathrm{B}(t,u)$ evaluated over the defining seed point interval $t_i \leq t \leq t_{i+1}$.

Now, given these regions, we want to compute a set of seed points on the curve guaranteeing convergence of Newton's method for minimum distance queries between query points in the regions and the curve. A summary of the approach is

1. Compute the seed points such that there is no overlap between a SPI region and the $|G'(t)| \geq 1$ unsafe convergence regions over the defining seed point intervals, as in

$$|G'(t)| \geq 1 \cap B(t,u) \equiv \phi, \qquad t_i \leq t \leq t_{i+1}\,. \tag{48}$$

2. Show that a query point inside a SPI region will converge to a closest point on the curve on its defining interval and that during convergence, the estimated closest point will not leave the interval.

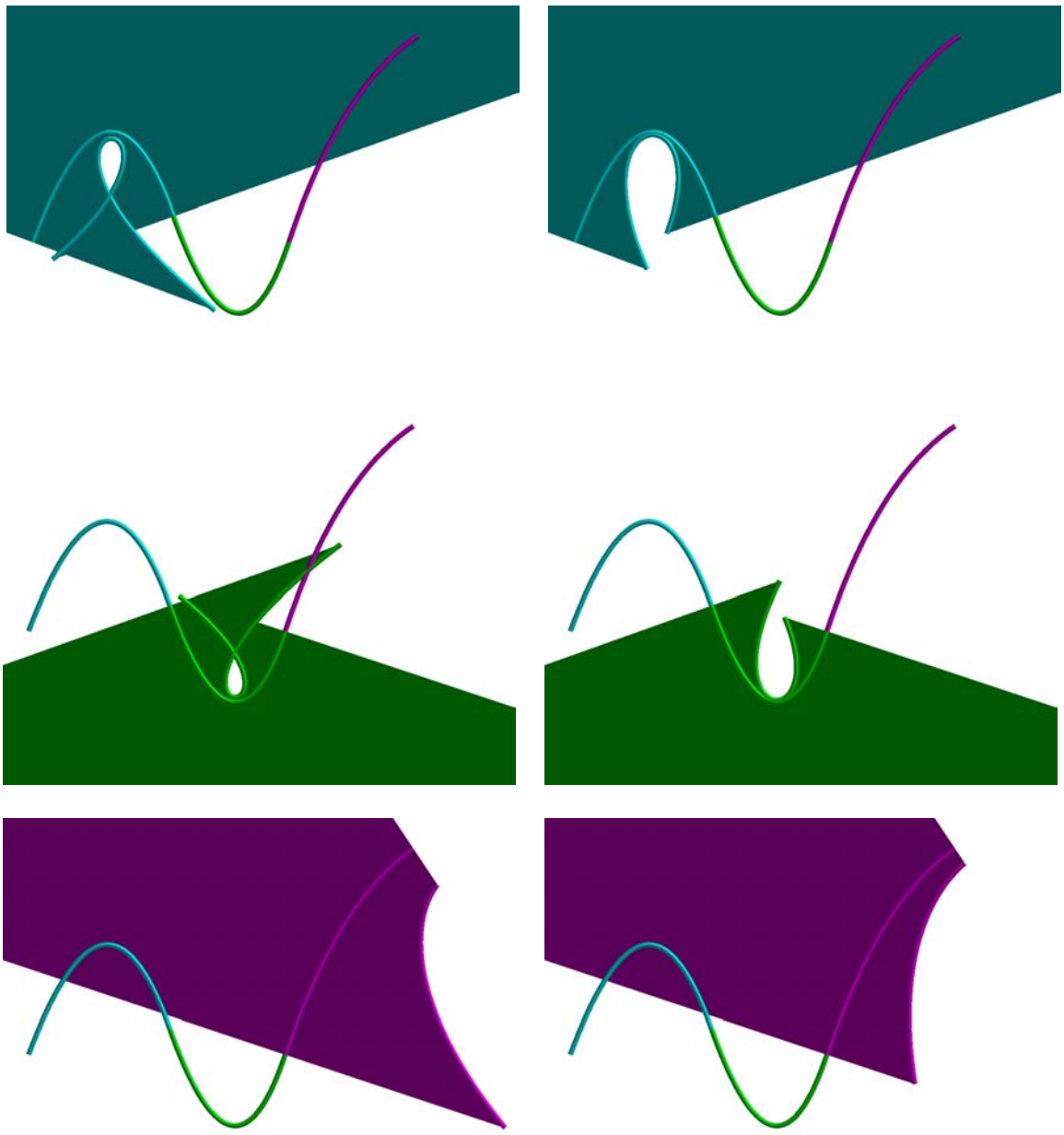


Figure 15: The three disjoint scaled evolute regions that bound query point location. (left) The scaled evolute regions with an alpha of 0.25. (right) The regions with an alpha of 0.1, which simplifies the structure of the regions for visualization.

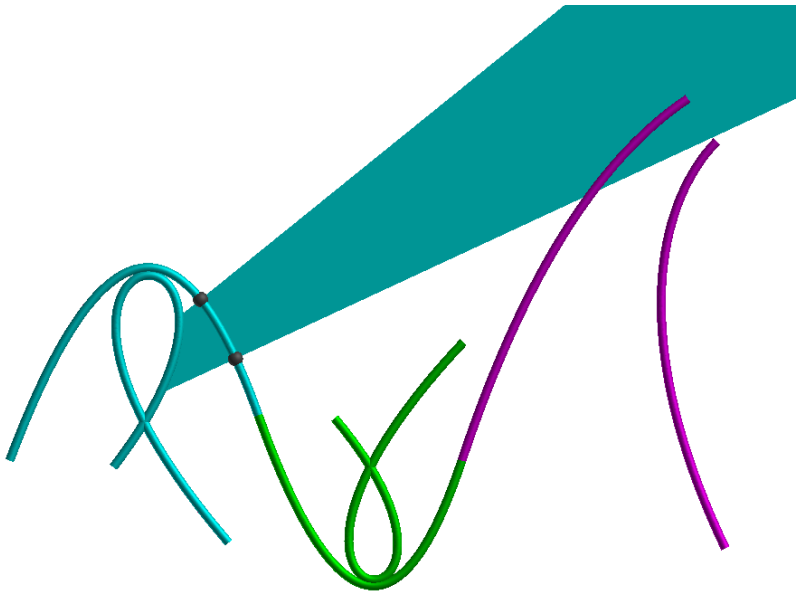


Figure 16: A seed point interval region is an interval of the scaled evolute region between two seed parameters.

Together, these steps show that during convergence, a curve parameter never yields an unsafe G' value for query points inside the SPI region, so Newton's method is guaranteed to converge.

Computing the Seed Points

The seed point computation is constructive in nature. The algorithm takes in a planar, twice-differentiable, regular curve and generates a set of seed points for the curve. As a preprocess, the algorithm breaks the curve at inflection points, so that the evolute of each subcurve, and its associated SPI-regions, is a continuous region.

The basic idea of the algorithm is to extend out a potential new seed point until it fails to create a safe SPI region for query points, at which point it stores the last parameter that created a safe SPI region as a new seed. A parameter, delta, determines how fast the new seed point extends, and there is an assumption that two seed parameters

separated by delta will pass the safe bounds test as described below. Because the algorithm is computationally efficient and a pre-process, a reasonable choice for delta is just a small floating point value, such as 0.00001, as long as it is a small fraction of the smallest difference between knot values. The algorithm, repeated on each subcurve, in pseudo-code is as follows:

```
t0 = the minimum parametric value for the subcurve
save_seed( t0 )
tlast = tnew = t0
while ( tnew <= maximum parametric value)
 bounds = seed_point_region( tlast , tnew + Δt )
 if (!safe( bounds ))
  save_seed( tnew ) # tnew is still safe
  tlast = tnew
      tnew = tnew + Δt
end while
save_seed( maximum parameter for subcurve )
```

For the *safe* test, recall that the convergence condition requires that $|G'(t)| < 1$ for all iterations of Newton's method. Since knowing the specific steps taken during convergence is equivalent to already knowing the solution, the test for a safe region uses the more conservative condition of Eq. 48, that $|G'(t)| < 1$ for all query points in the SPI region and for all parameter values in its defining interval.

The validity of this condition depends on Newton's method generating a series of t root estimates that stay within the interval. If the Δt generated by a step of Newton's method moves t outside the interval from t_i to t_{i+1}, then the $|G'(t)| \geq 1$ area associated with that t will not have been tested against the current SPI region.

To guard against this, we need to show that there is an extremum, and thus a place to which to converge, on a SPI region's defining interval for a query point in that region. Furthermore, Newton's method must move into the interval containing **P**, rather than the neighboring interval associated with the chosen seed point, and t must stay within that interval during the convergence process.

Lemma 1: *If the query point **P** is within a seed point region associated with an interval of the curve, then that interval also contains a local distance minimum to the query point.*

Proof: When **P** is within the SPI region, then

$$\left\langle \boldsymbol{\gamma}(t_i) - \mathbf{P}, \boldsymbol{\gamma}'(t_i) \right\rangle \leq 0 \text{ and } \left\langle \boldsymbol{\gamma}(t_{i+1}) - \mathbf{P}, \boldsymbol{\gamma}'(t_{i+1}) \right\rangle \geq 0 \quad \textbf{(49)}$$

A local extremal point is defined by a tangent orthogonal to the vector from the query point to the extremal point (Equation 6). Since the tangent is continuously varying within an interval, then the value of Equation 6 must cross zero as t goes from t_i to t_{i+1}, and the dot product with the tangent goes from negative to positive. When the dot product is zero, there is an extremal point on the curve. Because the query point is either on the convex side of the curve or closer to the curve than the evolute, that extremal point must be a minimum rather than a maximum.

By Lemma 1, if Newton's method converges, then the converged solution must lie on the defining interval of the SPI region that contains the query point. Additionally, Newton's method must start within the interval as it is initialized with a seed parameter that is one of the ends of the interval. Therefore, the only opportunity for the estimated closest point to escape the interval is during one of the intermediate steps.

Lemma 2: *The first parametric step of Newton's method is in the parametric direction from the initial seed parameter towards the other end of the interval.*

Proof: Recalling Eq. 28

$$\Delta t = -\frac{\langle \boldsymbol{\gamma} - \mathbf{P}, \boldsymbol{\gamma}' \rangle}{\langle \boldsymbol{\gamma} - \mathbf{P}, \gamma'' \rangle + \langle \gamma', \gamma' \rangle},$$

suppose the seed parameter t_i represents the parametric start of the interval. Then the seed point tangent points in the direction of the interval and the dot product of the tangent and the vector from **P** to $\boldsymbol{\gamma}(t_i)$ will be negative. The denominator of Eq.28 is positive because a point **P** on the curve yields a positive denominator and the denominator does not cross zero until **P** is at the radius of curvature, which is outside the SPI region. Therefore, Δt is positive. If the initial seed parameter is at the parametric end of the interval, then the numerator of Eq. 28 is positive, and Δt is negative.

Taken together, Lemmas 1 and 2 show that there must be a solution to converge to within the interval and that the first step moves the estimated root parameter from the seed parameter into the interval. Additional iterations must stay within the interval as the endpoints of the interval have already been tested to determine the starting parameter value and since the SPI regions are built such that $|G'(t)| < 1$ for all $t_i \le t \le t_{i+1}$, divergence cannot occur.

Having shown that $|G'(t)| < 1$ for all $t_i \le t \le t_{i+1}$ is a reasonable condition for testing the safety of a SPI region, we now need an implementation of that test. The safe test algorithm approximates the safety condition by sweeping a dense sampling of

parameter values from t_i to t_{i+1}. A sampling density of 0.00001 was used in our later examples; in general, the ends of the domain usually contain the problems so this is a robust test. At each sample t, a check is made to see if the $|G'(t)| \geq 1$ region lies completely outside a conservative approximation of the SPI region for that interval.

First, the $|G'(t)| = 1$ boundary curves are computed in the local reference frame of $\boldsymbol{\gamma}(t)$ (as in Equation 44). Then, a line segment (the *approximate scaled evolute curve*) connecting the endpoints of the scaled evolute curve for the defining interval is transformed into that local reference frame and intersected with the $|G'(t)| = 1$ boundary curves. An intersection with the line segment results in the safe test failing. Finally, the point of degeneracy along the normal is checked to make sure it lies outside the approximate scaled evolute curve. Passing these tests for all the sample t values shows that the condition $|G'(t)| < 1$ for all $t_i \leq t \leq t_{i+1}$ is satisfied to the level of approximation, and the spatial region is safe.

This check is adequate because of the nature of the $|G'(t)| \geq 1$ regions. Each is composed of two implicit quadratics with their point of maximum curvature placed along the normal to the estimated root, further away than the scaled evolute curve. The $|G'(t)| \geq 1$ region cannot completely contain the SPI region, because we know there is a line of $|G'(t)| = 0$ along the normal except at the point of degeneracy. The $|G'(t)| \geq 1$ region cannot bend back to intersect the sides of the SPI region because of the number of undulations in a quadratic. Finally, it is possible that the approximate scaled evolute curve diverges from the actual scaled evolute curve enough that the point of degeneracy along the normal lies inside the SPI region, and that case is explicitly tested. Therefore,

the only remaining possibilities are that the $|G'(t)| \geq 1$ region is completely outside the SPI region, or that the $|G'(t)| = 1$ boundary intersects the line segment approximating the scaled evolute curve. These possibilities are the ones that are checked by the safe region tests.

Results

This algorithm is straightforward to implement and runs quickly even though the repeated sweeps through each interval during testing may appear inefficient. Once a set of seed points is generated, repeated queries can use it, thereby amortizing the cost of generation.

Figure 17 shows the result of running the seed point generation algorithm on an example curve, with an $\alpha = 0.5$. The results set of seed points are densely placed in order to create safe regions, especially in areas of high curvature. The scaled evolute bound can be difficult to visualize, especially as it crosses inflection points. Figure 18 shows the same curve, but this time with $\alpha = 0.1$ instead. This keeps the bound closer to the curve. Note how the tighter bound requires fewer seed points to maintain safety.

To make the bounds even easier to visualize, we can clip the maximum distance of the scaled evolute bound from the curve. This keeps the safe region a consistent distance from the curve in regions of low curvature. These bounds are shown in Figure 19, along with a new visualization that computes the maximum magnitude of G' for all values of t in the interval nearest the query point. This visualization reassures that the computed seed points provide $|G'(t)| \leq 1$ for all t within that interval, and thus, that all t used by Newton's method will converge for query points within the safe spatial region.

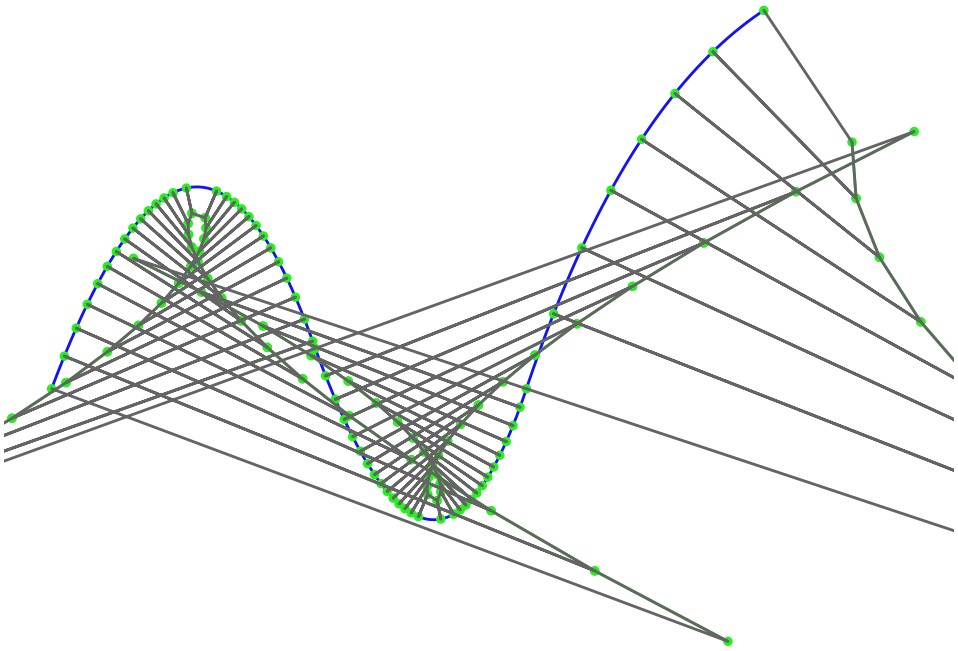


Figure 17: A sample curve with its approximate scaled evolute bound and generated seed points.

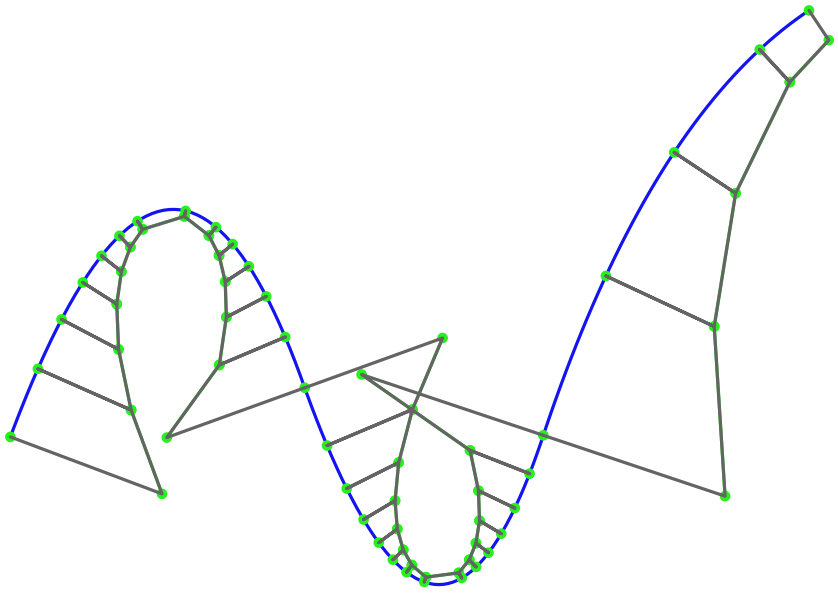


Figure 18: A tighter bound requires fewer seed points than for the bound in Figure 17.

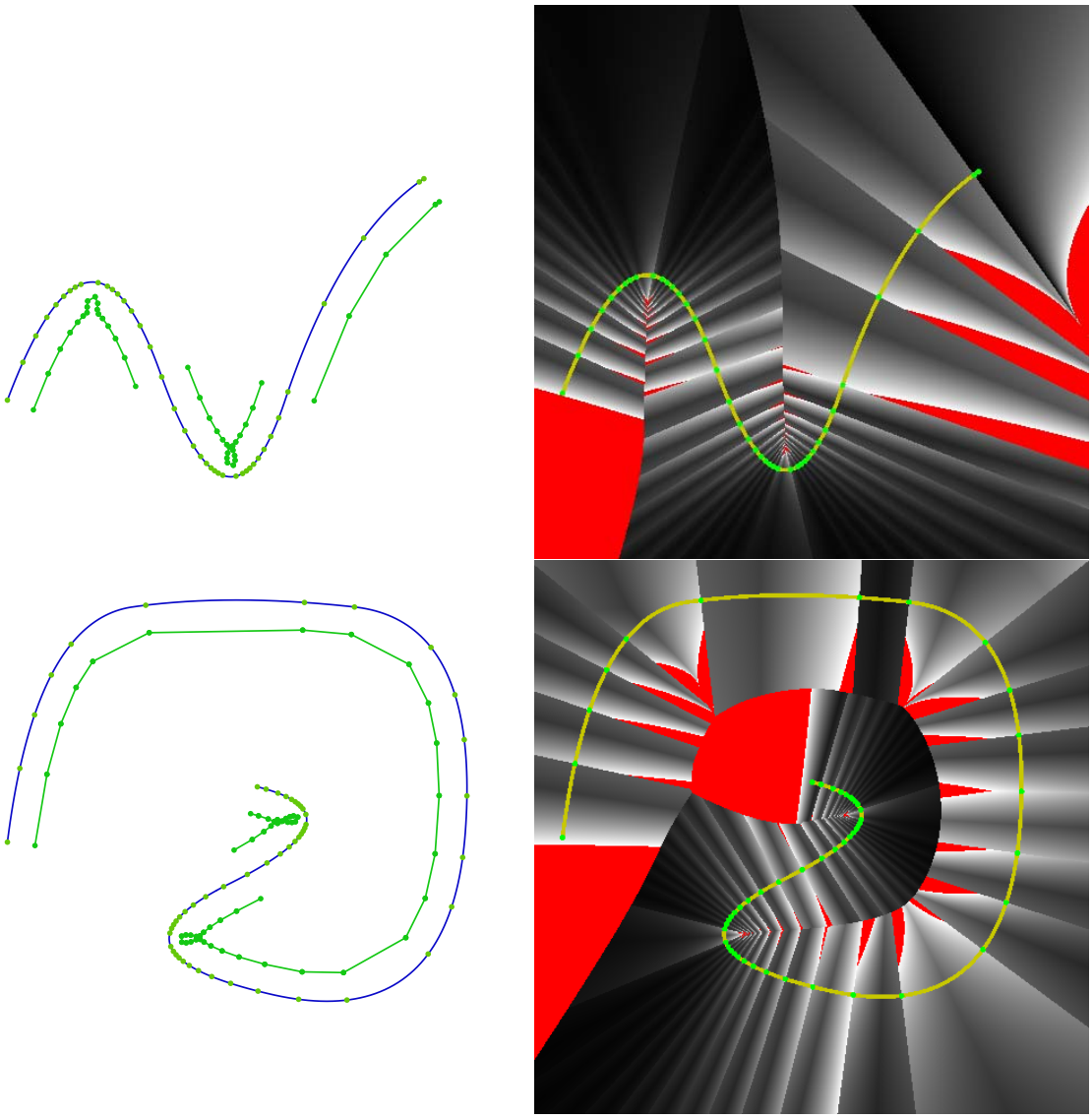


Figure 19: The generated seed points for two example curves, with the scaled evolute region clipped to a maximum distance. The left column shows the curve, seed points, and bounds. The corresponding maximum $|G'|$ values, visualized on the right, are computed by densely sampling the interval for each pixel, with $|G'|$ magnitudes greater or equal to one shown in a warning color. The maximal values approach but do not cross into the safe bounds, implying that repeated iterations of Newton's method will yield the closest point on the interval.

Discussion

This chapter has developed two main results – a geometric understanding of the convergence conditions for Newton's method for local minimum distance queries and an algorithm for generating seed points such that Newton's method is guaranteed to converge when the query point is within a safe spatial region associated with the seed points.

While the generated seed parameters provide assurance of local convergence, they do not provide any guarantees about global convergence. In the next chapter, methods are developed for global convergence of minimum distance queries of a point to a curve.

CHAPTER 5

LOWER BOUND PRUNING FOR GLOBAL MINIMUM DISTANCE QUERIES BETWEEN A POINT AND A CURVE

The last chapter derives a way to compute a set of initial seed points for Newton method, such that if the query point **P** lay within a spatial region based on a scaled evolute curve, then Newton's method would converge to a local minimum distance solution. However, the choice of initial seed point may lock the convergence process into the wrong region of the curve for a global minimum. This is most apparent along the medial axis of the curve as a zigzag pattern in the visualization of the closest point (Figure 20).

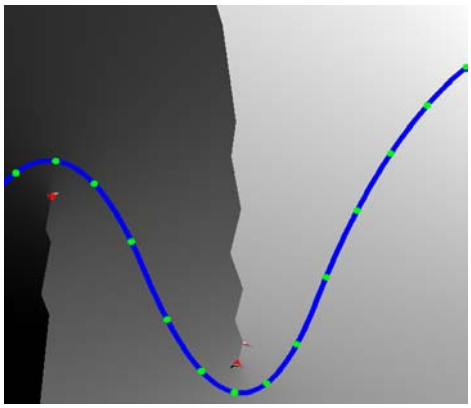


Figure 20: Choosing a seed point that converges only to a local minimum produces the zigzag pattern along the medial axis of the curve.

The solution is to not just use the nearest seed point to initialize Newton's method. Instead, Newton's method should test all seed points that might produce the nearest point on the curve. From the resulting set of local minima, the closest is chosen as the global solution. The basic steps of this process are:

1. Choose the closest untested seed point.
2. Find a local minimum using the chosen seed point, which establishes an upper bound on the minimum distance to the curve.
3. Compute a lower bound on distance to each interval of the curve associated with a seed point.
4. Remove from consideration seed points from intervals with a lower bound greater than the upper bound.
5. Repeat until all valid seed points have been tested.

The first two steps of this process are just the components of the minimum distance algorithm as developed in the previous chapter. The following section develops an efficient lower bound test for spline curves as needed for step 3.

Lower Bounds

Spline curves have the convex hull property, which states that the convex hull of the control polygon completely contains the curve. Furthermore, each polynomial piece (or rational piece in the positive weight NURBS case) of the curve is contained within the convex hull of the control vertices that influence its shape. This second property provides a tighter bound to the curve than the overall convex hull (Figure 21), but each polynomial piece may still contain several seed points, thus preventing a lower bound computation using the convex hull from distinguishing between them.

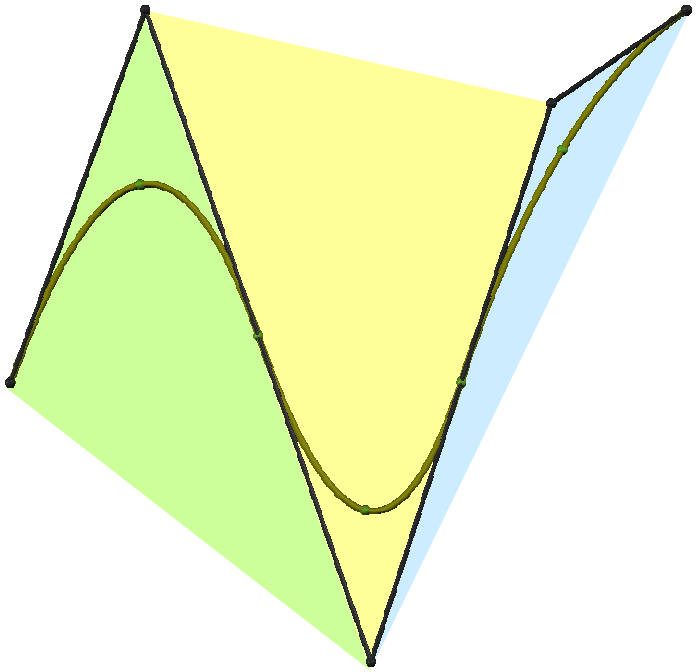


Figure 21: This quadratic curve has three polynomial pieces. The convex hull for each piece is shown as a different filled region. Each convex hull can enclose multiple seed points.

In order to provide tighter bounds, and thereby improve the algorithm's selectivity in pruning away seed points, a preprocess step to the algorithm refines the curve into intervals centered on each seed point (Figure 22). Note that these intervals are different from the safe region intervals used in the last chapter – those intervals spanned between two seed points. These new, refined intervals have control polygons that more tightly bound the intervals, and each convex hull bounds only one seed point.

The lower bound pruning algorithm may now efficiently determine which seed points will potentially yield a new global minimum and which ones may be removed from further consideration. In Figure 23, an upper bound has been found by using Newton's method on the nearest seed point. All other convex hulls that are within that

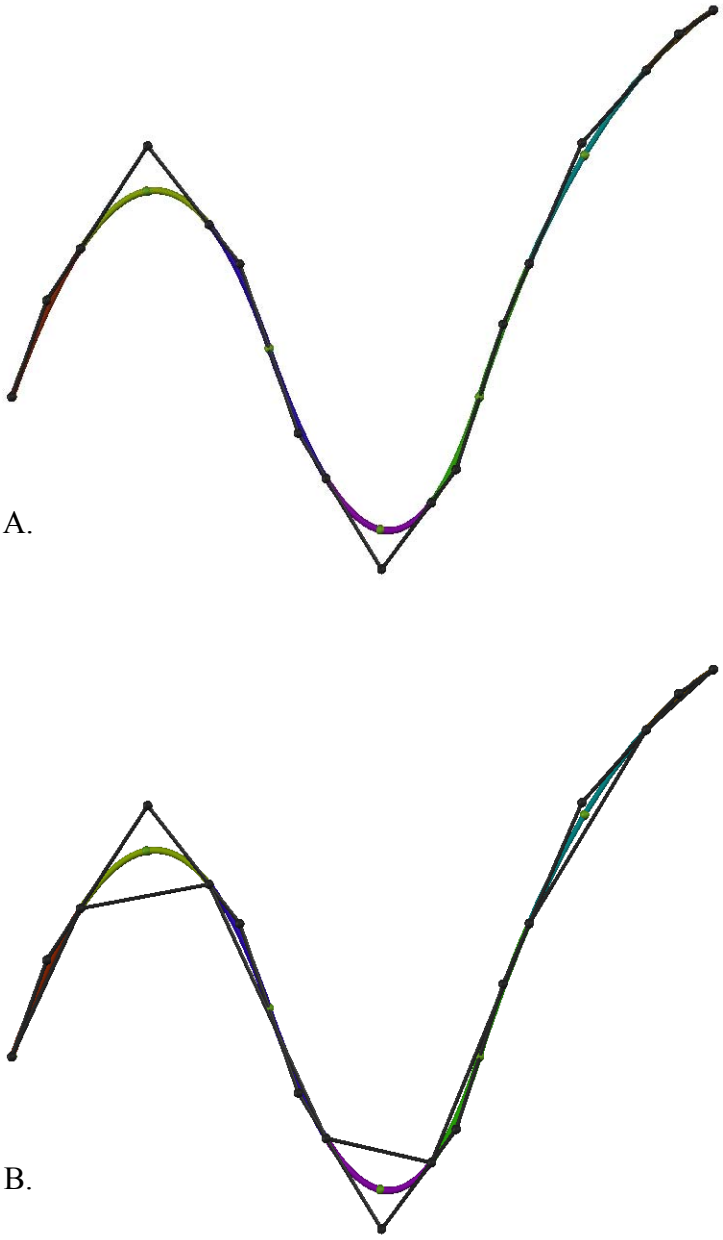


Figure 22: A refined curve provides a separate convex hull for each seed point. (A) The interval associated with each seed point colors each portion of the curve. (B) The refined control polygon produces tighter convex hulls that more closely bound the seed points.

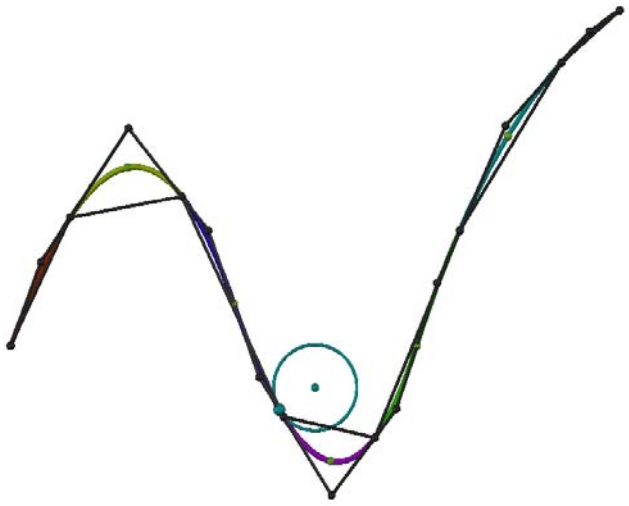


Figure 23: The circle shows the current upper bound on distance. The convex hull at the bottom of the curve overlaps that circle, showing that its seed point may produce a new global minimum.

upper bound distance may potentially contain portions of the curve that are closer than the existing upper bound point. This is visualized by drawing a circle around the query point with a radius equal to the upper bound distance. All convex hulls that overlap that circle contain seed points that are candidates for producing a new global minimum.

Fortunately, an explicit computation of the convex hull of each interval's control polygon is not needed, just the convex hull distance. The gjk package (web.comlab.ox.ac.uk/oucl/work/stephen.cameron/distances/) computes the minimum distance to the convex hull of a set of points by creating an implicit local portion of the hull while doing a simplex search of the point set for a minimum distance.

As new and better upper bounds are found, the circle shrinks, and more intervals of the curve (and associated seed points) can be pruned away. Eventually, all seed points are either tested or pruned away, and the final upper bound on distance can be used as the global minimum distance.

Results

Figure 24 demonstrates how lower bound pruning can yield global minima by testing several candidate seed points. The closest point visualization using lower bound pruning shows a smooth medial axis, while the visualization using a single seed point shows the effect of locking in convergence. In addition, the results using lower bound pruning show fewer regions of divergence, as the multiple seed points provide some redundancy in difficult regions.

In order to compare the computation costs, repeated queries like those in Figure 24 were used, but without actually creating the image. A 1000x1000 query grid was timed for Newton's method and for Newton's method augmented with lower bound pruning. Computation time was measured on a 1.1GHz PIII-M laptop. As Table 1 shows, using lower bound pruning adds only about 10% to the computation time.

Discussion

The lower bound pruning technique is independent of the proper computation of seed points as developed in the last chapter and provides useful redundancy in multiple calls to Newton's method when seed points are less carefully chosen. In practice, only a few seed points are needed to find a global minimum and the remaining points are pruned with only the additional cost of a minimum distance to convex hull call. However, when used with the safe seed point technique, and when the query point **P** is within the safe regions of all the used intervals, then the convex hull pruning creates a guarantee of global convergence for minimum distance queries.

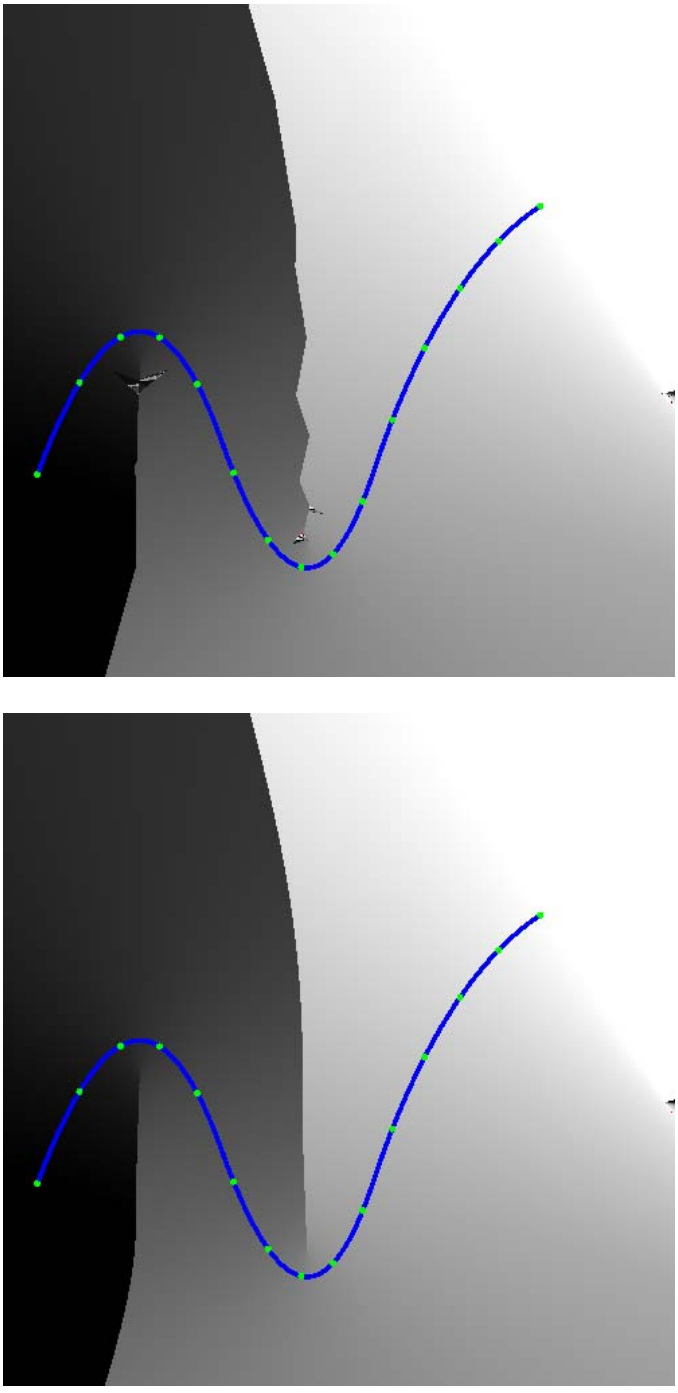


Figure 24: Distance fields computed using a single seed point and using lower bound pruning. (top) Only using the closest seed point yields incorrect global minima. (bottom) The lower bound pruning technique developed in this chapter produces global minima as well as improved convergence.

Table 1: Lower bound pruning is not much more expensive to use than standard Newton's method but provides much better reliability

Method	# seed points	precision	time (s)
Newton	15	10^{-6}	59.5
Lower bound pruning	15	10^{-6}	67.4

CHAPTER 6

GLOBAL SEARCH FOR POINT-CURVE DISTANCE MINIMA USING NORMAL CONES

The last chapter introduced a set of tests for reliable global distance queries between a point and a curve. This chapter develops a global method for reliably finding all the local extrema in distance between a point and a curve.

Prior global methods pruned curve intervals using lower and upper distance bounds and subdivision[39], or robust root finding using interval testing and subdivision[40]. However, from Equation 6

$$E(t) = \left(\boldsymbol{\gamma}(t) - \mathbf{P}\right) \cdot \frac{d\boldsymbol{\gamma}}{dt},$$

we see that collinearity between the solution point normal and the vector between the query and solution point is the key to a distance extrema. Pruning based on distance may find a global minimum, but does not use any of the curve normal information in that search. This chapter develops an approach based on bounded normals to prune the curve based on collinearity, rather than distance bounds.

Normal Cone Approach

Just as it did for the global minimum approach, curve subdivision plays an important role in this approach. Since the search is for all local extrema of the curve,

rather than comparing a distance measure of each subdivided interval to a global standard, each interval is judged independently. The outline of the normal cone approach is

1. Bound the range of normals and spatial extent of each interval.
2. See if the collinearity condition might be satisfied for the interval for a given query point.
3. Subdivide intervals that might meet the collinearity condition.
4. Repeat until the remaining curve intervals contain small normal ranges, then compute exact local extrema using local methods.

Bounding the Range of Normals with Normal Cones

The derivative of a NURBS curve is a vector-valued NURBS curve. Thus, the properties of NURBS curves, such as the convex hull property, also apply to the derivative curve. This can be used to bound the range of normals for that curve.

The curve subdivision process turns pieces of NURBS curves into rational Bézier curves, which are a subset of NURBS curves. For a Bézier curve $B(t)$ of degree *d,* with control points P_i, and basis functions $\beta_{i,d}(t)$

$$B(t) = \sum_i P_i \beta_{i,d}(t) \tag{50}$$

and its derivative is

$$B'(t) = \sum_i d(P_{i+1} + P_i)\beta_{i,d-1}(t)\,. \tag{51}$$

The derivative curve is also known as the *hodograph*, and is readily computed as scaled differences of adjacent control points of the original curve. Since the convex hull

property holds for the derivative curve, the range of tangent directions is bounded by the convex hull of the derivative curve control points.

These tangent directions are encapsulated as a cone that contains the range of tangent directions. The normalized tangent control points are averaged, and the spread of the cone is computed by finding the maximum angle between the average tangent and the tangent curve control points. The normal cone is just the tangent cone rotated by 90 degrees.

Since the collinearity condition depends not only on the curve normal, but also on the vector between the query point and the solution point, the normal cone alone is insufficient to compute bounds on the collinearity condition. The range of vectors between the query and possible solution points must also be taken into consideration. Those vectors are bounded by first bounding the spatial extent of the curve interval with a circle. This circle bounds the possible locations of the solution point on the curve. The range of vectors between the query point and solution point then forms a cone, called the *solution line cone*, between the query point and bounding circle (Figure 25).

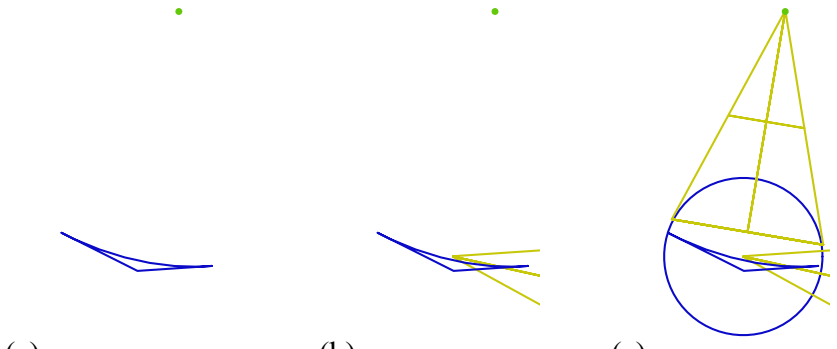


Figure 25: Building the bounds on normal and solution vectors.
(a) The query point and curve interval, with its control polygon.
(b) The tangent cone computed from differences of control points.
(c) The solution line cone encompasses the circle around the interval.

Checking the Collinearity Condition

Given the normal cone and the solution line cone, the collinearity check looks for overlap between the normal cone and the solution line cone. An overlap indicates a possible solution within that curve interval. Checking for a perpendicularity condition between the tangent and solution line cone rather than collinearity with the normal cone avoids the need for an additional cone in the negative normal direction to handle query points on both sides of the curve.

Intervals passing the collinearity test are subdivided. These subdivided curves have tighter bounds on their spatial extent, which produces tighter solution line cones and tighter bounds on their normal cone. These tighter bounds are better able to prune intervals during the next iteration (Figure 26).

Computing an Exact Solution

The subdivision terminates when the normal cone spread angle falls below a user specified epsilon. Exact local minima are computed from these intervals using local numerical methods. An interval accepted for exact tests performs nodal mapping between the query point and interval, followed by Newton's method. Our tests used an epsilon of 0.01, which implies that Newton's method is fed with an initial estimate at least that close to the exact solution.

Not all intervals will contain a valid local minimum because of the conservative nature of the bounding cone and bounding circle tests. Solutions that leave the interval during Newton's method can be removed, or converged solutions can be checked for redundancies. The latter approach is preferred since it potentially provides multiple initial guesses to Newton's method for each local minimum, improving robustness.

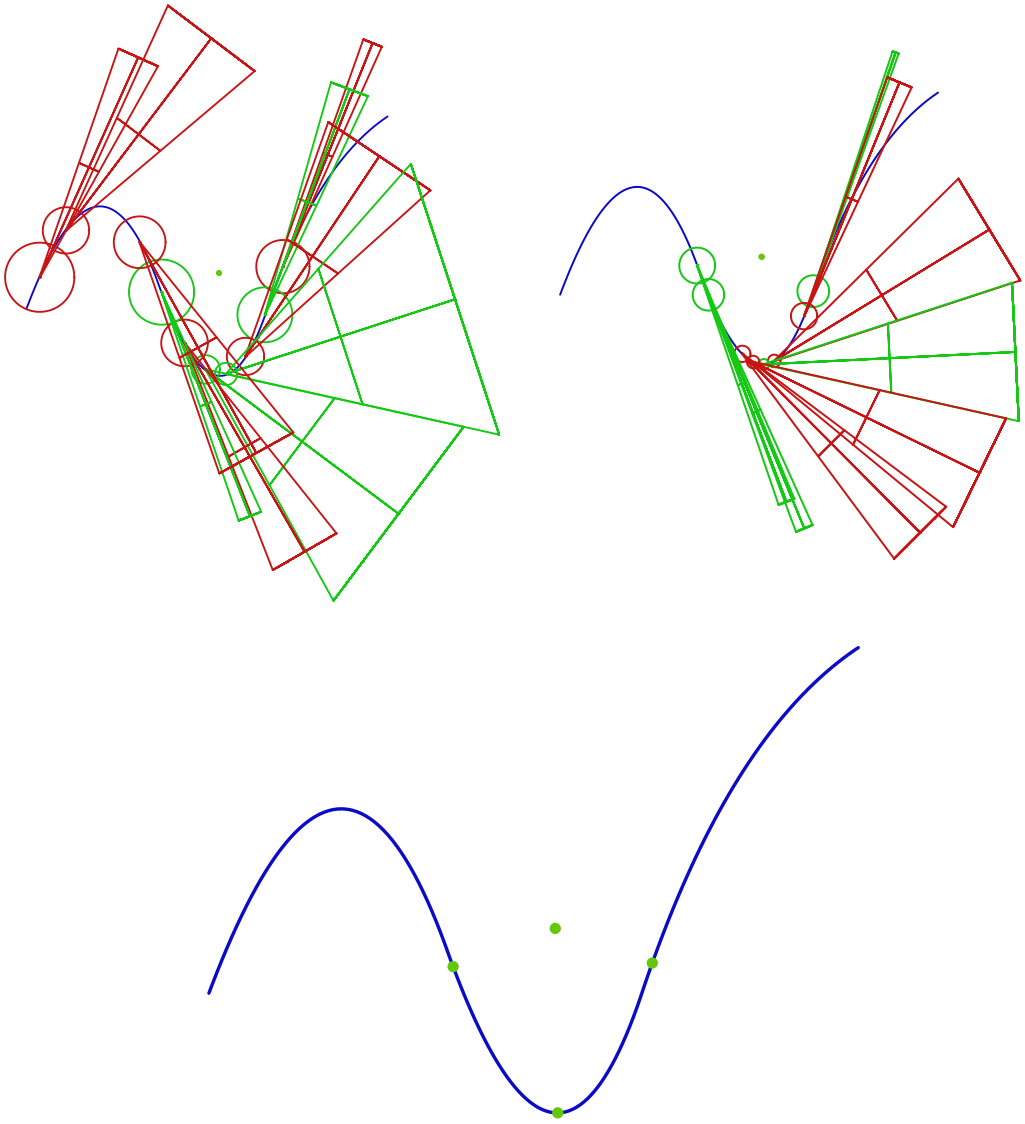


Figure 26: Two consecutive levels of subdivision during the normal cone approach and the final result are shown. The cones are the tangent cones computed from the derivative curve and the circles provide the basis for solution line cones. Intervals in that pass are subdivided and ones that fail are removed from further computation.

Results

The normal cone approach reliably finds all the local extrema. The speed of the method is roughly linearly dependent on the number of local minima. If the minima are closely spaced, then they may share common subdivided intervals, improving performance. In tests using the same curve as in the last chapter, the normal cone approach ran three times slower than the globally convergent approach; however, it is computing and returning more information about distance to the curve.

CHAPTER 7

MINIMUM DISTANCE QUERIES BETWEEN A POINT AND A SURFACE

Until this chapter, only points and curves have been the primitives under consideration for distance queries. The next chapters extend the approaches developed for curves to apply to surfaces and further to haptic applications. This chapter provides an analysis of the degeneracy conditions for Newton's method when solving for minimum distances between a point and a parametric surface, which will help motivate later development of robust geometric techniques.

Multidimensional Newton's Method

Multidimensional Newton's method solves for simultaneous zeros in a system of functions, $\mathbf{F}$,

$$\mathbf{J} \cdot \Delta\mathbf{p} = -\mathbf{F}, \tag{52}$$

where $\mathbf{J}$ is the Jacobian of $\mathbf{F}$, and $\Delta\mathbf{p}$ is the change in parameter needed to simultaneously move each function in $\mathbf{F}$ closer to zero[41]. The actual change in parameter is found by inverting the Jacobian and multiplying the inverse through the system of functions,

$$\Delta \mathbf{p} = -\mathbf{J}^{-1}\mathbf{F}. \tag{53}$$

Recall from Chapter 3 that the squared distance between a point in space, $\mathbf{P}$, and a regular parametric surface $\mathbf{S}(u,v)$ is

$$D^2(u,v) = \big(\mathbf{S}(u,v) - \mathbf{P}\big)\cdot\big(\mathbf{S}(u,v) - \mathbf{P}\big). \tag{54}$$

Minima are found by solving for simultaneous zeros of the two partials of $D^2(u,v)$,

$$\mathbf{F} = \begin{bmatrix} \big(\mathbf{S}(u,v) - \mathbf{P}\big)\cdot\mathbf{S_u} \\ \big(\mathbf{S}(u,v) - \mathbf{P}\big)\cdot\mathbf{S_v} \end{bmatrix}, \tag{55}$$

as well as checking the edges of the domain.

For the particular case of finding the roots of Equation 55, multidimensional Newton's method expands to

$$\begin{bmatrix} \dfrac{\partial}{\partial u}[(\mathbf{S}-\mathbf{P})\cdot\mathbf{S_u}] & \dfrac{\partial}{\partial v}[(\mathbf{S}-\mathbf{P})\cdot\mathbf{S_u}] \\ \dfrac{\partial}{\partial u}[(\mathbf{S}-\mathbf{P})\cdot\mathbf{S_v}] & \dfrac{\partial}{\partial v}[(\mathbf{S}-\mathbf{P})\cdot\mathbf{S_v}] \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = -\begin{bmatrix} (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_u} \\ (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_v} \end{bmatrix}, \tag{56}$$

or

$$\begin{bmatrix} (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_{uu}} + \mathbf{S_u}\cdot\mathbf{S_u} & (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_{uv}} + \mathbf{S_u}\cdot\mathbf{S_v} \\ (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_{uv}} + \mathbf{S_u}\cdot\mathbf{S_v} & (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_{vv}} + \mathbf{S_v}\cdot\mathbf{S_v} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = -\begin{bmatrix} (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_u} \\ (\mathbf{S}-\mathbf{P})\cdot\mathbf{S_v} \end{bmatrix}. \tag{57}$$

The rest of this section analyzes degeneracy conditions while using Newton's method to find roots of Eq. 55, in particular, by providing a geometric interpretation of allowable placements of **P**.

Degeneracy Conditions

As in the curve case, recasting the vector between the point and the surface to local coordinates on the surface helps create a geometric interpretation of degeneracy conditions. So, by using the tangent plane and normal at $\mathrm{S}(u,v)$ as a local frame

$$\mathbf{S}-\mathbf{P}=x\mathbf{S_u}+y\mathbf{S_v}+z\mathbf{N}, \tag{58}$$

the Jacobian in Eq. 57 becomes

$$\mathbf{J}=\begin{bmatrix} x\mathbf{S_u}\cdot\mathbf{S_{uu}}+y\mathbf{S_v}\cdot\mathbf{S_{uu}}+z\mathbf{N}\cdot\mathbf{S_{uu}}+\mathbf{S_u}^2 & x\mathbf{S_u}\cdot\mathbf{S_{uv}}+y\mathbf{S_v}\cdot\mathbf{S_{uv}}+z\mathbf{N}\cdot\mathbf{S_{uv}}+\mathbf{S_u}\cdot\mathbf{S_v} \\ x\mathbf{S_u}\cdot\mathbf{S_{uv}}+y\mathbf{S_v}\cdot\mathbf{S_{uv}}+z\mathbf{N}\cdot\mathbf{S_{uv}}+\mathbf{S_u}\cdot\mathbf{S_v} & x\mathbf{S_u}\cdot\mathbf{S_{vv}}+y\mathbf{S_v}\cdot\mathbf{S_{vv}}+z\mathbf{N}\cdot\mathbf{S_{vv}}+\mathbf{S_v}^2 \end{bmatrix}. \tag{59}$$

An iteration of multidimensional Newton's method becomes degenerate when **J** when

$$\det(\mathbf{J})=0, \tag{60}$$

which for our two-dimensional case is just

$$\det(\mathbf{J})=\mathbf{J}_{00}\mathbf{J}_{11}-\mathbf{J}_{01}\mathbf{J}_{10}=0. \tag{61}$$

Computing the determinant of **J** = 0 when it is in the form of Eq. 59 forms a quadric implicit surface, the *degeneracy quadric*, where positions of **P**, defined in the local coordinate frame, cause **J** to be singular. Looking at Eq. 59, each element in the Jacobian is a plane with a normal defined by the second partial in that element, so that the plane in $\mathbf{J}_{00}$ is orthogonal to $\mathbf{S_{uu}}$, the plane in $\mathbf{J}_{11}$ is orthogonal to $\mathbf{S_{vv}}$, and the planes for both $\mathbf{J}_{01}$ and $\mathbf{J}_{10}$ are orthogonal to $\mathbf{S_{uv}}$. These planes are the dominant visual cues in some forms of the quadric surfaces.

For example, Figure 27 shows a surface with an estimated closest point near the center of the patch. The degeneracy quadric is shown as a point sampled implicit. The

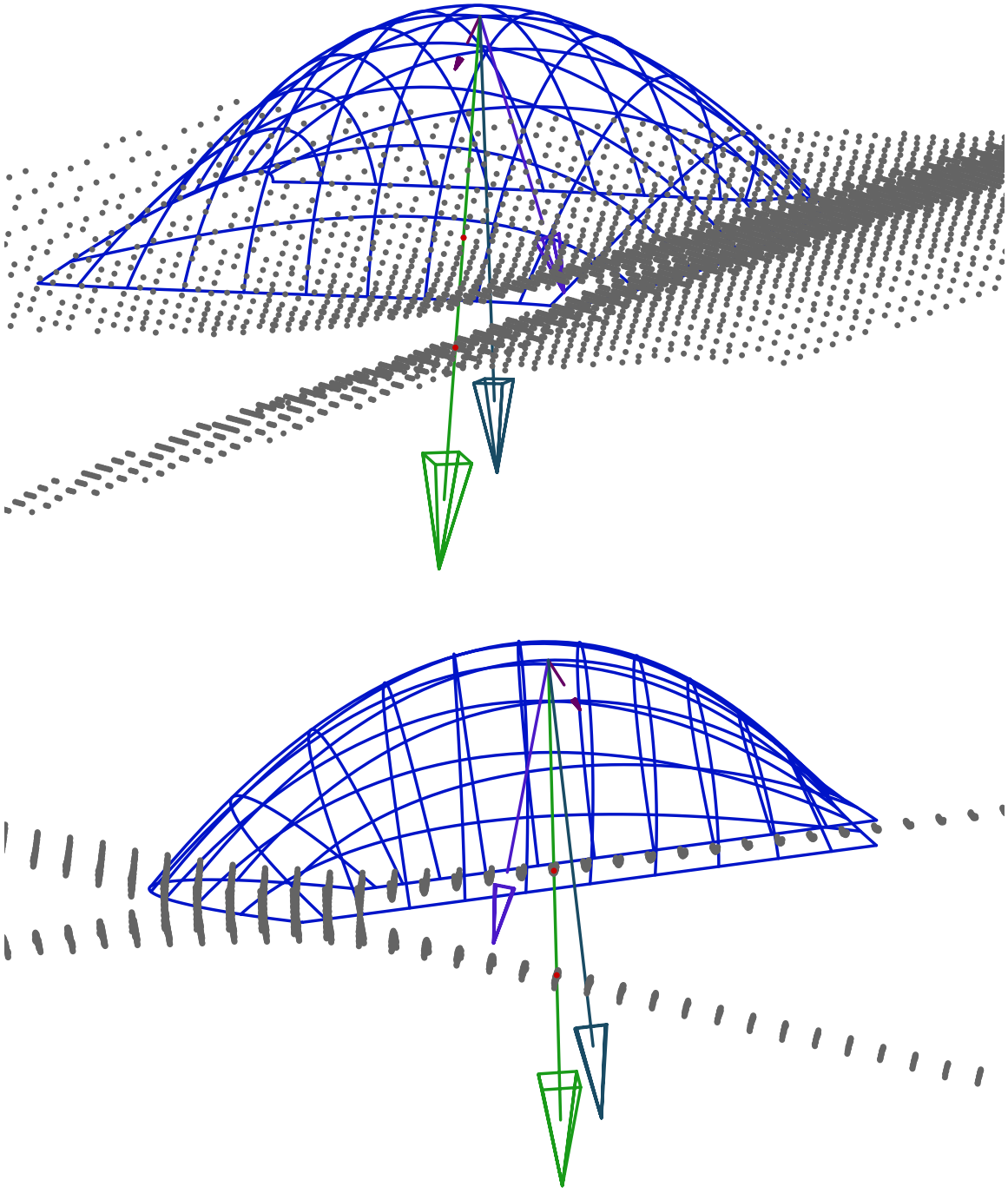


Figure 27: The shape of the degeneracy quadric derives from the direction and magnitude of the surface partial derivatives.

second part of Figure 27 shows the same situation, but with the model rotated to show how the shape of the quadric is largely determined by $\mathbf{S}_{\mathbf{uu}}$ and $\mathbf{S}_{\mathbf{vv}}$, which are much larger in magnitude than $\mathbf{S}_{\mathbf{uv}}$. The implicit hyperboloid has portions perpendicular to $\mathbf{S}_{\mathbf{uu}}$ and $\mathbf{S}_{\mathbf{vv}}$. When $\mathbf{S}_{\mathbf{uv}}$ grows in magnitude, these cues are no longer so easily discernable. More special forms of the quadric will be discussed later in this chapter.

Degeneracy Along the Normal

While the shape of the degeneracy quadric can be quite complex, we are most interested in finding minimum distance solution given a reasonable starting point, such as tracking the closest point on a surface while moving a haptic interface. This situation is when *x* and *y* in Eq. 58 are small.

So, setting $x = 0$ and $y = 0$, the determinant of **J** equals zero when

$$\det\left(\begin{bmatrix} z\mathbf{N}\cdot\mathbf{S}_{\mathbf{uu}}+\mathbf{S}_{\mathbf{u}}{}^{2} & z\mathbf{N}\cdot\mathbf{S}_{\mathbf{uv}}+\mathbf{S}_{\mathbf{u}}\cdot\mathbf{S}_{\mathbf{v}} \\ z\mathbf{N}\cdot\mathbf{S}_{\mathbf{uv}}+\mathbf{S}_{\mathbf{u}}\cdot\mathbf{S}_{\mathbf{v}} & z\mathbf{N}\cdot\mathbf{S}_{\mathbf{vv}}+\mathbf{S}_{\mathbf{v}}{}^{2} \end{bmatrix}\right) = 0 \qquad \textbf{(62)}$$

The first and second fundamental forms of a surface[38] are

$$\begin{aligned} G &= \begin{bmatrix} \mathbf{S}_{\mathbf{u}}\cdot\mathbf{S}_{\mathbf{u}} & \mathbf{S}_{\mathbf{u}}\cdot\mathbf{S}_{\mathbf{v}} \\ \mathbf{S}_{\mathbf{u}}\cdot\mathbf{S}_{\mathbf{v}} & \mathbf{S}_{\mathbf{v}}\cdot\mathbf{S}_{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \\ L &= \begin{bmatrix} \mathbf{S}_{\mathbf{uu}}\cdot\mathbf{N} & \mathbf{S}_{\mathbf{uv}}\cdot\mathbf{N} \\ \mathbf{S}_{\mathbf{uv}}\cdot\mathbf{N} & \mathbf{S}_{\mathbf{vv}}\cdot\mathbf{N} \end{bmatrix} = \begin{bmatrix} L & M \\ M & N \end{bmatrix} \end{aligned}. \qquad \textbf{(63)}$$

Thus, in terms of the first and second fundamental forms, Eq. 62 is

$$\det\left(\begin{bmatrix} zL+E & zM+F \\ zM+F & zN+G \end{bmatrix}\right) = 0\,, \qquad \textbf{(64)}$$

and the determinant equals zero at the roots of

$$z^2\left(\frac{LN-M^2}{EG-F^2}\right)+z\left(\frac{LG+EN-2MF}{EG-F^2}\right)+1=0. \tag{65}$$

These roots are more easily interpreted by rewriting the quadratic in terms of the principal curvatures at $\mathbf{S}(u,v)$. The principal curvatures of a surface, κ_1 and κ_2, define the maximum and minimum curvatures of curves through a point on a surface. They are defined in terms of the fundamental forms of a surface[42] such that

$$\begin{aligned}\kappa_1\kappa_2 &= \left(\frac{LN-M^2}{EG-F^2}\right)\\ \kappa_1+\kappa_2 &= \left(\frac{LG+EN-2MF}{EG-F^2}\right)\end{aligned} \tag{66}$$

These definitions offer a simple substitution for rewriting Eq. 62 as

$$z^2\kappa_1\kappa_2+z(\kappa_1+\kappa_2)+1=0\,, \tag{67}$$

and the roots are just

$$z=-\frac{1}{\kappa_1}\ and\ z=-\frac{1}{\kappa_2} \tag{68}$$

which shows that Jacobian becomes singular and Newton's method fails to converge when the query point $\mathbf{P}$ is a distance along the normal equal to one of the principal radii of curvature of the estimated closest point on the surface. Figure 28 shows a circle for each principal curvature, oriented in the corresponding principal direction. The exaggerated dots are the centers of the circles, and the centers lie on the degeneracy quadric, as predicted in Eq. 68.

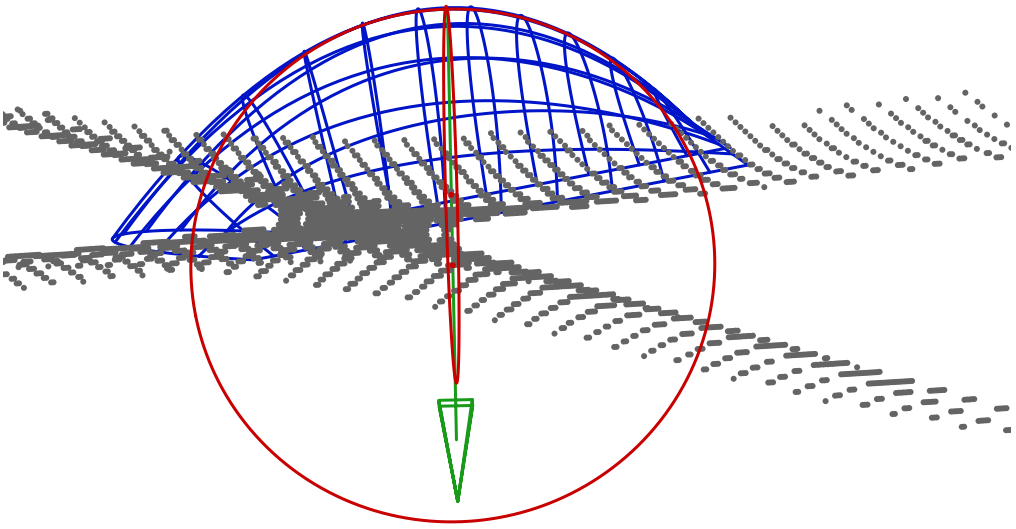


Figure 28: The surface normal intersects the degeneracy quadric at the radius of curvature for each the principal curvatures.

Additional Examples of the Degeneracy Quadric

There are forms of $\mathbf{S}(u,v)$ that produce simpler forms of the degeneracy quadric. Figure 29 shows a surface patch with $\mathbf{S}_{\mathbf{uu}}$ and $\mathbf{S}_{\mathbf{vv}}$ in the same direction, and $\mathbf{S}_{\mathbf{uv}}$ with zero magnitude. The quadric collapses to two parallel lines, each a principal radius of curvature along the normal from the point on the surface. This particular patch was defined using a warp operator, which displaces control points along a common normal direction weighted by distance from the warp center. Surfaces of revolution also have second partials that lead to simplified forms of the degeneracy quadric such as the cylindrical ellipsoid in Figure 30.

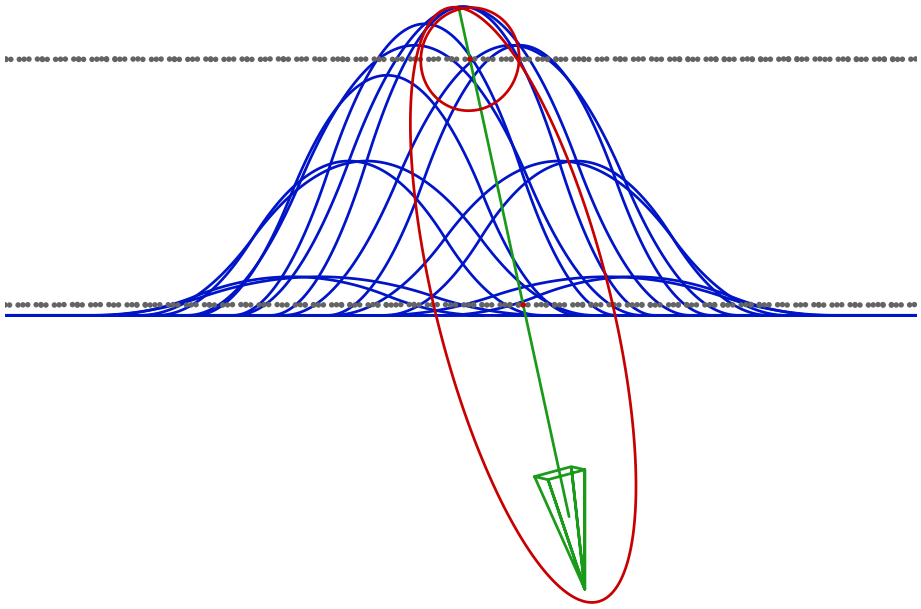


Figure 29: The degeneracy quadric forms two parallel planes perpendicular with the second partials.

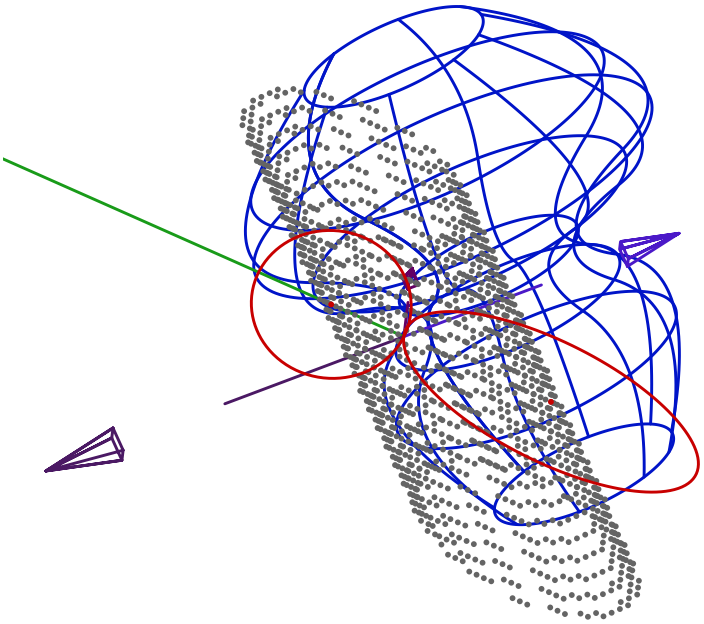


Figure 30: A cylindrical ellipsoid degeneracy quadric forms from a point on a surface of revolution.

Discussion

This chapter shows how to use multidimensional Newton's method to find local minima in distance between a point and a surface. As in the curve case, regions of degeneracy can cause Newton's method to fail, but the degeneracy conditions are even more complex in the surface case. In addition, simply avoiding a degeneracy does not mean that Newton's method will converge. Therefore, while Newton's method plays a role where fast local methods are essential, this result motivates the development of robust geometric approaches to solve point to surface minimum distance queries.

CHAPTER 8

GLOBAL SEARCH FOR POINT-SURFACE DISTANCE EXTREMA USING NORMAL CONES

The normal cone approach for point-curve distance extrema extends naturally to point-surface distance queries. In the curve case, a cone bounded the range of normals for a curve interval. In the surface case, the same approach applies, except to surface patches instead of curve intervals.

While it is possible to compute surface normal cones using symbolic pseudonormals, with non-unit length vectors, we use simpler tests based on both tangent directions. The subdivision test computes bounding cones for each of the tangent directions (Figure 31) and a solution line cone from the query point to a bounding sphere around the patch. Patches are pruned by checking each tangent cone for perpendicularity with the solution line cone. A failure to find potential orthogonality between a vector in the solution line cone and vectors in either of the tangent cones means that surface patch cannot contain a local solution.

Tangent cones are computed by finding differences of control points over all the rows or columns of the control mesh, depending on the tangent direction. Similarly, the patch bounding sphere is computed from the average and extent of the patch control

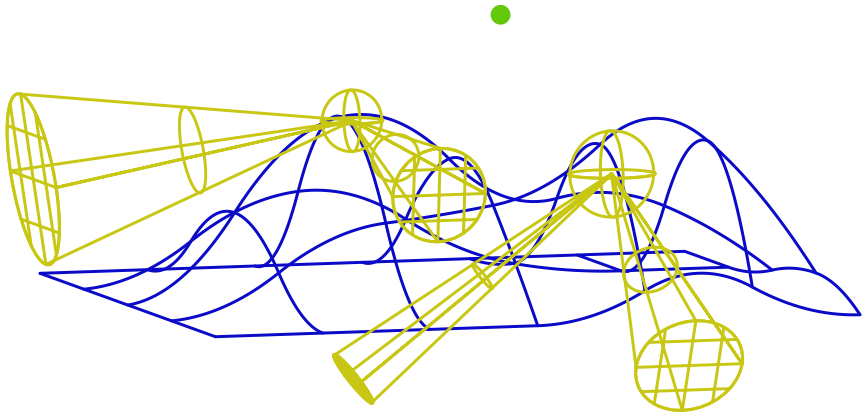


Figure 31: Two surface patches with their sample tangent cones. The spheres bound he spatial extent of the patch and help to form the solution line cone from the query point.

points. For low order Bézier surfaces, the number of rows or column is equal to the parametric order, so the computation is not too costly.

Surface patches that may contain a solution are split into four smaller patches (Figure 32), while patches that fail the normal cone test are removed. Exact closest points are found on patches with both tangent cone spread angles smaller than an epsilon. A combination of nodal mapping[11] followed by Newton's method computes these exact local closest points (Figure 33). Because leaf nodes may not contain a local solution, solutions that leave the leaf node domain need to be discarded, or checked against existing solutions to see if they are redundant. The second approach also provides some additional robustness to Newton's method by providing multiple starting locations for each local minimum distance solution.

When the local solutions form a curve on the surface, this curve is approximated to the level of the convergence epsilon. However, heuristics can detect the large number of child patches being created in this degenerate situation and choose a smaller sample if speed is needed over a full solution.

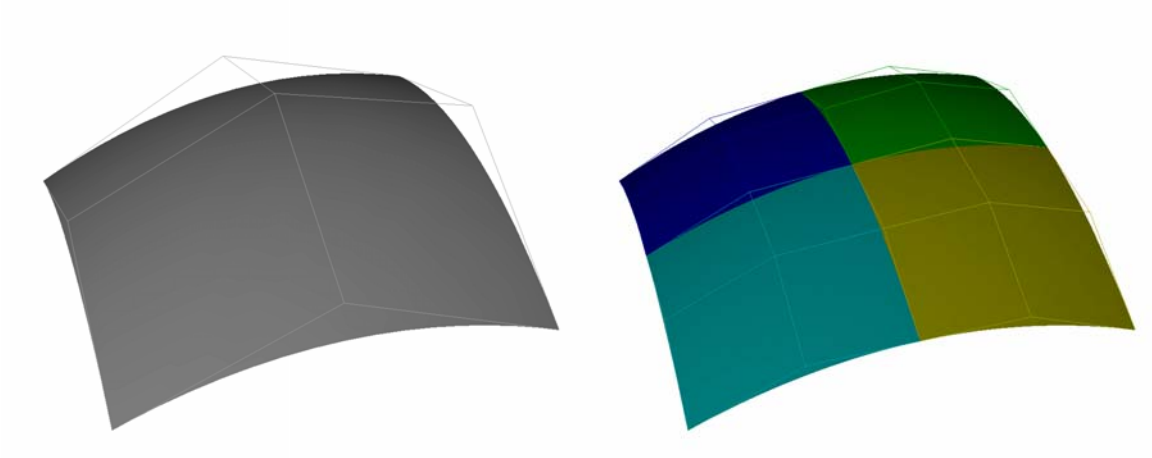


Figure 32: Refining a patch splits it into four independent surfaces. The new control meshes more tightly bound the surface.

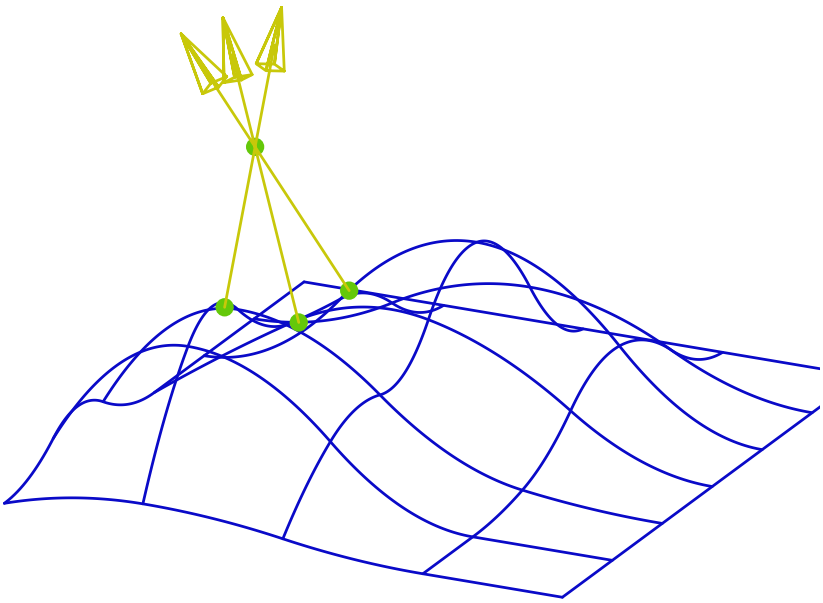


Figure 33: All local minimum distance solutions are found using the normal cone approach. Nodal mapping and Newton's method improve the final solution.

Experiments indicate that the subdivision hierarchy manages to stay fairly sparse during typical distance queries, which shows the subdivision and pruning test is effective. Figure 34 shows the number of patches checked at each iteration of the algorithm for two different query points, one yielding five solutions and the other three. The chart does show that for the tested queries, little pruning occurs during the first few levels of subdivision, and that further on, the algorithm creates and destroys a lot of patches that do not yield a solution. Fortunately, since the subdivision process always splits patches in half, a tested child patch is identical from one query to the next, so surface patches and their associated cones could be stored for use in following queries. However, retaining all surface patches can be costly in terms of computer memory.

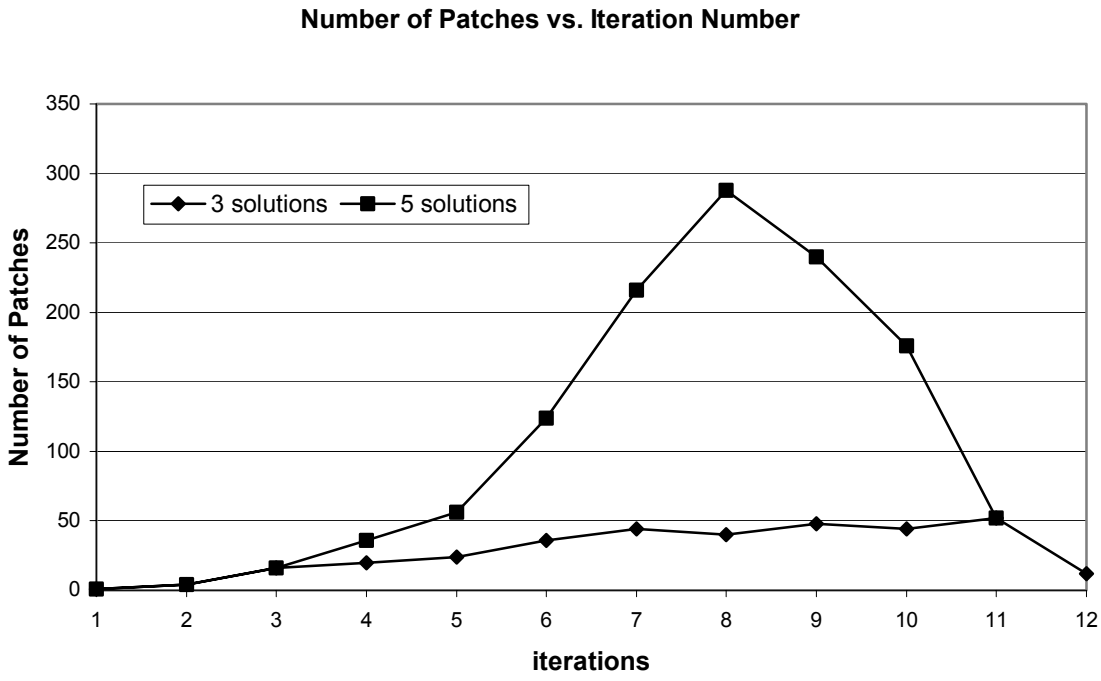


Figure 34: The number of active surface patches during each iteration varies with search complexity. The query that returned five local minima had to create and search many more patches than the query that returned three.

Coherency with the Dynamic Subdivision Tree

Instead, we adopt the approach of retaining portions of the subdivision tree over multiple queries, but removing portions that are no longer currently used. Patches are stored as computed in a tree data structure, a *dynamic subdivision tree*, with each node having four children. Patches (and their children) are deleted only when a query tests a patch's tangent cones and determines there is no local solution in that patch. This has the effect of retaining patches during sets of temporally and spatially coherent queries. The patch may need to be recomputed later, but this approach balances memory usage with computational efficiency.

The dynamic subdivision tree approach dramatically improves computational speed when query points are closely spaced. The dynamic subdivision tree was tested by creating a sample path above a surface, moving along the path in small increments, and performing a distance query at each step (Figure 35). Without coherence, 1000 distance queries took 24 seconds; with coherence, the same test took 4 seconds, 6 times faster.

Discussion

Normal cones, in conjunction with dynamic subdivision trees, provide robust and efficient computation of all local distance minima between a point and a surface. In the next chapter, we show how the developed distance techniques can be applied to haptic rendering of NURBS surfaces.

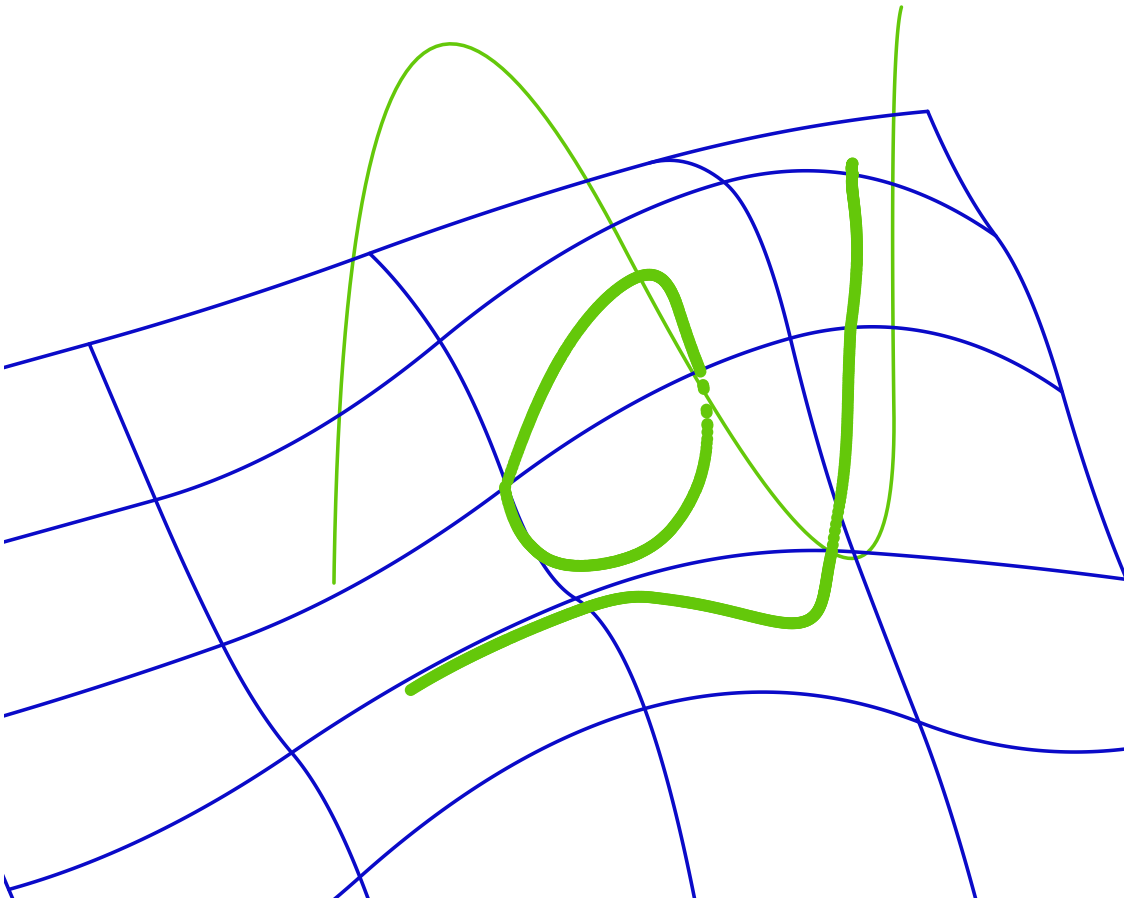


Figure 35: The query point moves along the test path (thin line). The local closest points on the surface are shown as densely placed points, and include regions where three local minima were found. This type of test is sped using a dynamic subdivision tree.

CHAPTER 9

HAPTIC RENDERING OF NURBS SURFACES

Haptic rendering requires update rates around 1000 Hertz to maintain stability and sensations of contact between a point and a surface. These rates cannot be met without careful design of the overall system so that virtual environment complexity can be handled gracefully. We use a three-phase approach to focus computational resources where they are needed during each time step of the haptic rendering:

1. Remove surfaces too distant to be interacted with using spherical bounds.
2. Use normal cones to find local minima on the remaining surfaces.
3. Update these local minima using Newton's method as the haptic query point moves, using these local updates to determine penetration into the surface.

Finding Local Minima

Local distance minima are needed, rather than a global distance to the surface, because the local minima provide more information about possible future contacts. Imagine the haptic query point is above the surface, but in a valley between two hills. The haptic query point could enter the surface at the floor of the valley, or along either of the two hills. If these features are small, there might not be enough time for a global distance computation to reinitialize Newton's method after a discontinuous jump in closest point

location. A normal cone search for all local minima tracks all these potential future contacts (Figure 36).

Local Update

Newton's method updates the local minima found with the normal cone computation. Only those normal cone minima that start above the surface are tracked, else additional extrema from the far side of the model could be included. Newton's method updates these local closest points at haptic rates. The normal cone computation continually feeds the local update so that discontinuous jumps are correctly updated.

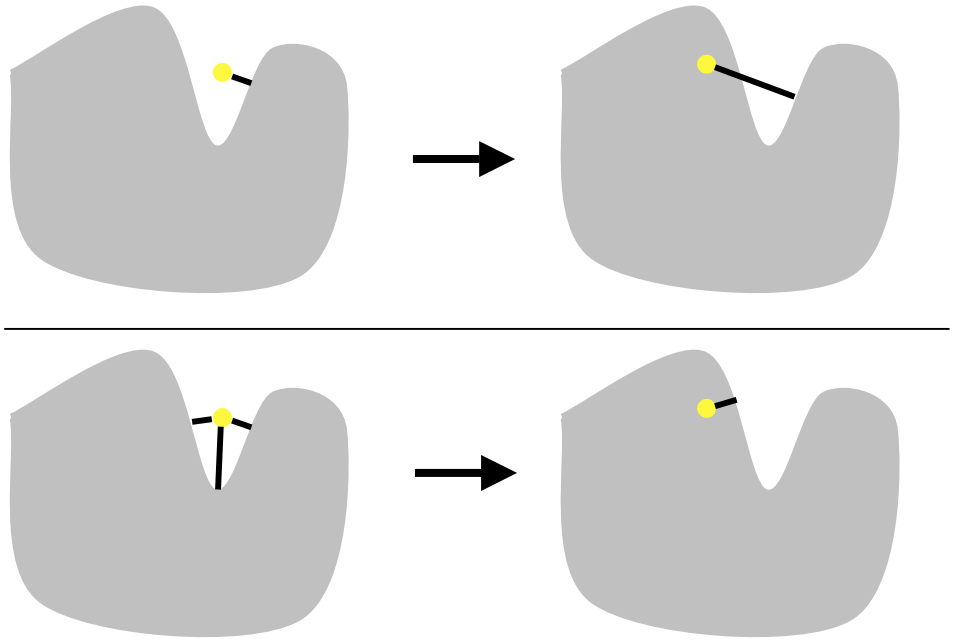


Figure 36: All local minima are needed to initialize local tracking. (top) A global minimum is not updated in time to jump the local tracking to the new side. (bottom) Normal cone computation finds all local minima, all of which are tracked, so the penetration is detected.

However, if the haptic query point enters the surface, as determined with a test with the surface normal, then only that local minimum updates and the normal cone computations are ignored until the haptic query point leaves the surface.

Newton's method is fairly stable in this situation because the normal cone computation provides an accurate initial point when outside the surface, and inside the surface, the haptic interaction forces prevent the haptic query point from penetrating far into the model, where degeneracies are more likely. If Newton's method fails to converge, then the local closest point is updated using a linear approximation from [11].

Results

Figure 37 shows the results of a simulated haptic path entering a surface. When the patch first starts, there are multiple local distance minima, and the normal cone algorithm identifies these regions. Newton's method provides fast updates to these minima. As the path nears the surface, only one local minimum remains, and again, the normal cone computation updates the set of minima for the local tracking to update. When the path enters the surface, the local closest point is maintained only by Newton's method. Upon exit, the normal cone algorithm again periodically updates the local update.

For this test case, the combined normal cone and Newton method algorithm was able to maintain a 2000Hz update rate, well in the realm for haptic rendering. This technique provides a robust means of quickly tracking local distance minima on a surface.

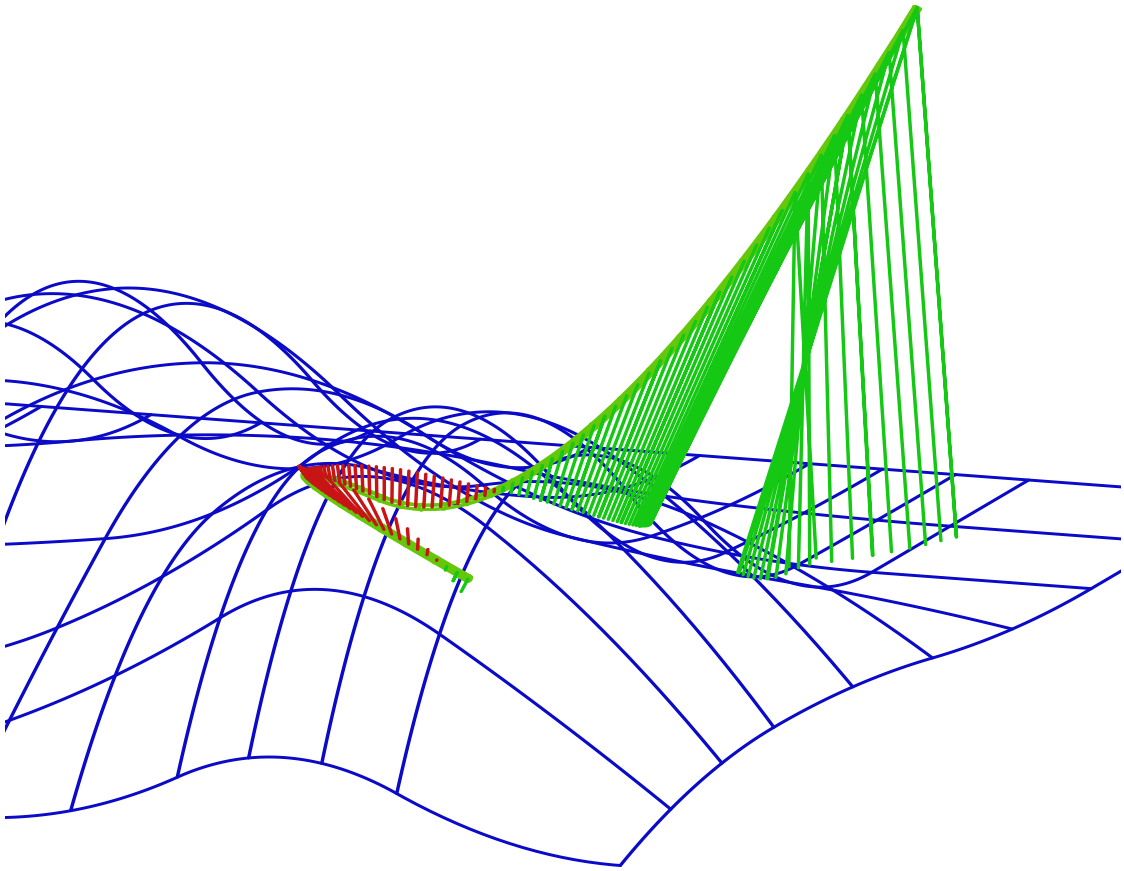


Figure 37: A simulated haptic path enters a surface. Above the surface, multiple minima are tracked, whereas only one is tracked while below. Forces are generated only while inside. The normal cone computation updates the set of local minima for the local update to maintain at haptic rates.

CHAPTER 10

GLOBAL SEARCH FOR LOCAL DISTANCE MINIMA BETWEEN POLYGONAL MODELS

Distance queries to and between polygonal models have most frequently relied upon upper and lower bound distance algorithms to hierarchically prune away model regions[20][43][14][44]. In Chapters 7-8, bounds on normal direction allowed a global search for local minima to parametric surfaces. This chapter applies that paradigm to polygonal models, extending the computation of local solutions from sculptured models to faceted ones. In support of this computation, we develop a data structure, the spatialized normal cone hierarchy (SNCH) that hierarchically encapsulates the range of normals and the position in space for portions of a triangulated surface.

Many computer graphics applications depend upon the surface normal. Data structures to encapsulate sets of these surface normals have accelerated backface culling[45], lighting[46], model simplification[47], and silhouette extraction[48]. These problems can be defined in terms of a point, such as a viewpoint or point light source, and a single model. The SNCH data structure applies to problems involving two models, and is thus suitable for general minimum distance computations.

The Spatialized Normal Cone Hierarchy

For polygonal models, with their fixed resolution, it is feasible to precompute a hierarchy of normal cone bounds, rather than the dynamic computation used in the earlier chapters. This hierarchy is based on a simple data structure consisting of a normal cone, represented by a cone axis vector and a cone semiangle; and a Euclidean bounding volume, in this case a sphere, represented by a center and a radius (Figure 38). The bounding volume associates the normals with a particular region of model space, or spatializes the normal cone. Each node of the data structure also contains pointers to two child nodes and a pointer to an underlying triangle if it is a leaf node.

Constructing the Hierarchy

This data structure encapsulates a triangular mesh structure of vertices, edges, and triangles. The connectivity of the triangles is not important for the operation of the minimum distance algorithm, except for the construction of appropriate normals for each of the primitives.

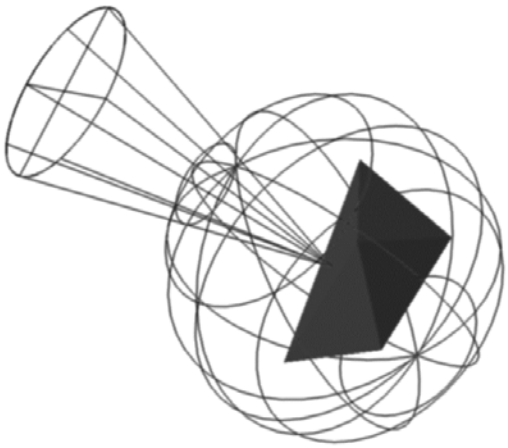


Figure 38: The spatialized normal cone encompasses the range of normals and bounds the geometry.

The first step in the construction process constructs a spatial bounding volume hierarchy. The publicly available PQP code (http://www.cs.unc.edu/~geom/SSV/) recursively subdivides the model into a Euclidean bounding volume hierarchy. A bounding sphere fits the geometry at each node for use in the normal cone.

The second step computes a normal cone for each node of the Euclidean bounding volume. We compute the cone axis by averaging the triangle, edge and vertex normals contained in that node. The cone semiangle is the maximum angle between the cone axis and the node geometry normals. These steps are repeated down the hierarchy until the normal cone data structure is complete.

Minimum Distance Computations with a SNCH

Although this data structure was developed for model-model distance computations, we start by describing the solution to the minimum distance between a query point **P** and a polygonal model. All the surface features, such as faces, edges, or vertices, on the model containing a local minimum distance solution have an associated vector collinear with the vector from **P** to the closest point on the feature (Figure 39).

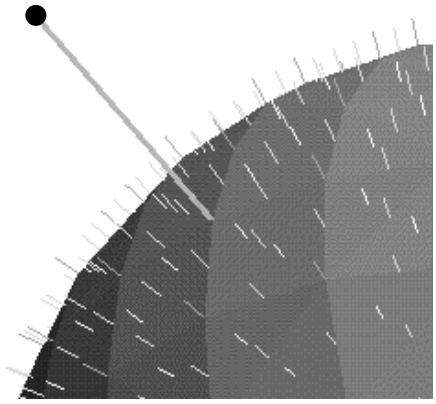


Figure 39: The line between the query point and the feature is in line with the normal at the feature.

The global minimum distance is the length of the shortest such vector to a feature on the model. However, it is not necessary to select only the shortest line as the solution. Instead, all the vectors that satisfy the criteria can be used. These are the local minimum distance solutions between **P** and the model (Figure 40).

While a linear algorithm could test each triangle for a potential solution, this would be slow for large models. Instead, the SNCH is used to hierarchically test portions of the model, quickly excluding most of it from consideration.

The range of possible vectors from the query point **P** to the geometry primitives contained within a node of the SNCH forms a potentially complicated shape. Instead of computing this form, the range of vectors from **P** to the bounding sphere at a node conservatively bounds the range of vectors to the contained geometry. This set of vectors forms a solution line cone between **P** and the bounding sphere (see Figure 41).

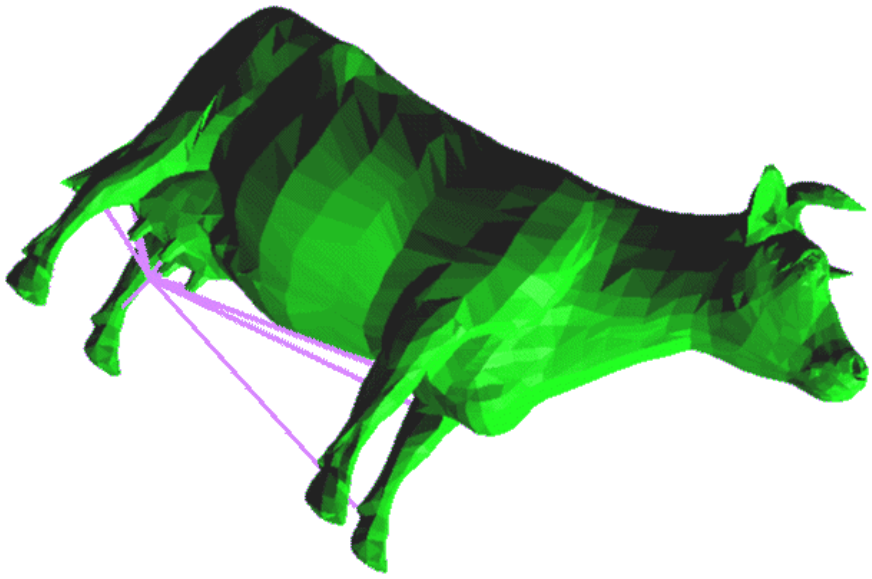


Figure 40: All the solutions satisfying the collinearity approach. The local minimum distance lines extend to each leg, the udder, and tail.

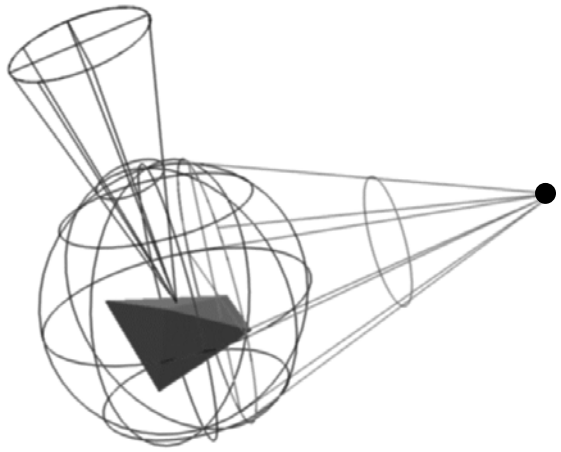


Figure 41: A node of the SNCH contains geometry bounded by a sphere. The range of vectors from a query point to that sphere forms the solution line cone, which bounds the range of vectors to the contained geometry.

If the solution line cone contains a vector that is collinear with a vector in the node's normal cone, then there may be a local minimum distance solution from the query point to the contained geometry. This possibility is simply tested by expanding the normal cone by the solution line cone semiangle, and then seeing if **P** is contained within the expanded normal cone. Nodes that pass this test are subdivided and recursively tested until the leaf level, where an exact test is applied. Nodes that fail are pruned, as neither they nor their children can contain a local minimum distance solution.

Leaf Tests for Local Minimum Distance

In a global minimum leaf test, the minimum distance between the query point and the leaf triangle is computed and compared to previous leaf distances. To test for a local minimum distance, the leaf test also computes the closest point on the leaf triangle to the

query point. Rather then comparing to a global minimum, the geometric feature associated with this closest point must have a normal that is collinear with the minimum distance vector. The closest point may lie on a triangle face, edge, or vertex, each with a characteristic range of normals. If the closest point lies on the triangle face, the leaf test must only compare the triangle normal with the minimum distance vector. If the closest point lies on a triangle edge, the minimum distance vector must be compared against the span of normals from the triangle face to the edge normal.

When the closest point matches up with a triangle vertex, the minimum distance vector must be compared against the span of normals for the vertex. Each vertex normal covers an area on the Gauss sphere defined by the surrounding triangle face normals (Figure 42). This area is divided into disjoint regions to associate with each surrounding triangle for computational efficiency.

We divide the normal range for a vertex among each surrounding triangle to prevent redundant solutions. A possible region would be to take the span from the triangle normal to the vertex normal and then half of each edge span. However, that

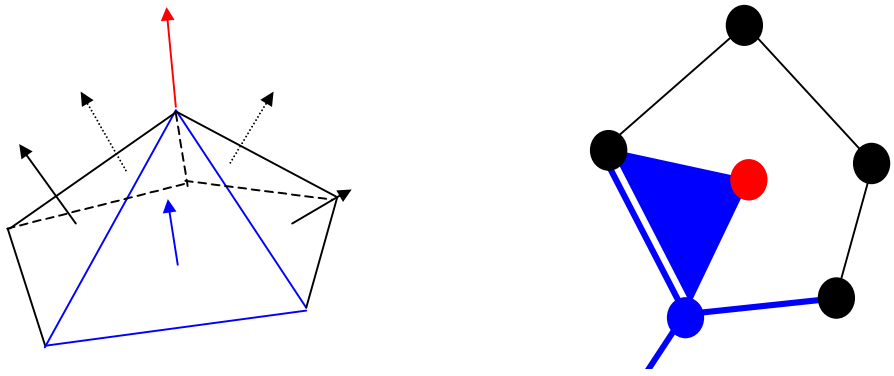


Figure 42: A. The vertex normal is surrounded by triangles. B. Each triangle normal maps to a point on the Gauss sphere and each edge maps to an arc.

produces a quadrilateral on the sphere that may cross itself in complex ways. Instead, the triangular span from the triangle normal to the vertex normal to the left neighbor's face normal (Figure 42.B.) is associated with each triangle vertex. Given this association, it is easy to determine if the minimum distance vector falls within the triangular range of normals.

Local Minimum Distance for Two Models

This approach for local minimum distance between a query point and model extends to problems involving two models. Essentially, when there are two models, we must account for the range of possible solution lines between two nodes, one on each model. The range of solution lines now spans between the two bounding spheres, one from each node, and forms a solution line cone for two models (Figure 43).

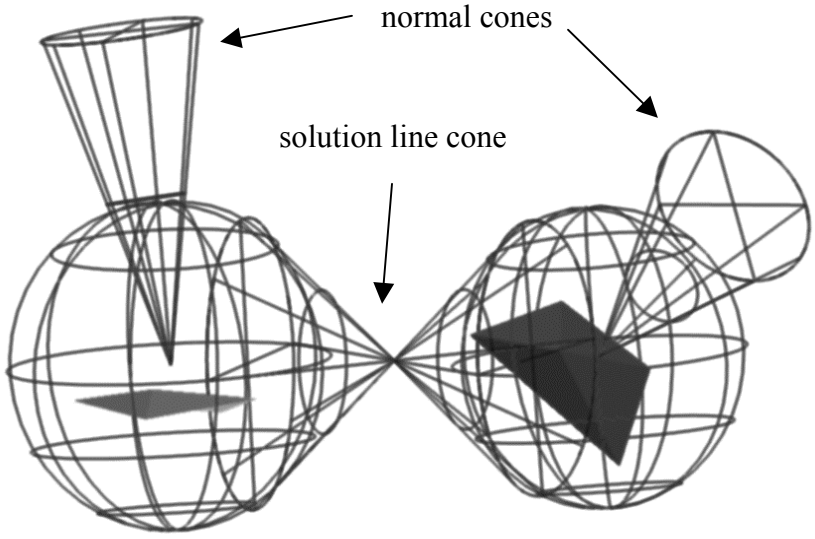


Figure 43: The solution line cone now spans between the two bounding spheres – encompassing all possible solution lines between the models.

The pruning test for a pair of nodes checks if the normal cones face each other, contain at least one pair of collinear normals, and that this pair is also collinear with a line in the solution line cone. This extension allows pruning of each model's normal cone tree down to leaf nodes that may meet the local minimum distance requirement. The leaf test finds the minimum distance between the two leaf triangles, and checks the minimum distance line against the ranges of normals from the closest point features as in the leaf test for point-model queries.

Results

First, we tested the query point to model method with a variety of models. The models rotated randomly while the query point stayed fixed in space above the moving model. The local minimum distances (including the global minimum distance) from a point in space to a model (Table 2) are computed in sub-millisecond time on average. This makes the presented approach appropriate for a number of tasks, including haptic rendering of polygonal models and volumetric conversion.

Table 2: The local minimum distance from a point in space to a model is fast for a range of model types.

	Sphere	Holes3	Small bunny	Large bunny
# triangles	32,700	11,800	10,800	69,500
Time (ms)	0.06	0.05	0.1	0.2

To test this method applied to the distance between two models, we compared the speed of the local minimum distance method with the global minimum distance method from the PQP package. The normal cone method gives faster results on fairly smooth surfaces. This suggests it may be a good choice for certain classes of models, such as those derived from subdivision surfaces. It was slower on surfaces with a lot of fine detail, such as the bunny. The small bumps on the bunny translate into a wide range of normals in a small area, as well as producing numerous local minima. The normal cone method runs at a competitive speed (Table 3) compared with the global PQP method, but returns all the local minima as well (Figure 44).

Table 3: Timing results for finding the distance between models.

	sphere-torus	**torus-cow**	**torus-bunny**
# Triangles 1	8192	4096	4096
# Triangles 2	4096	5804	69451
PQP (secs)	0.007	0.006	0.0057
Normal cone (secs)	0.004	0.008	0.0059

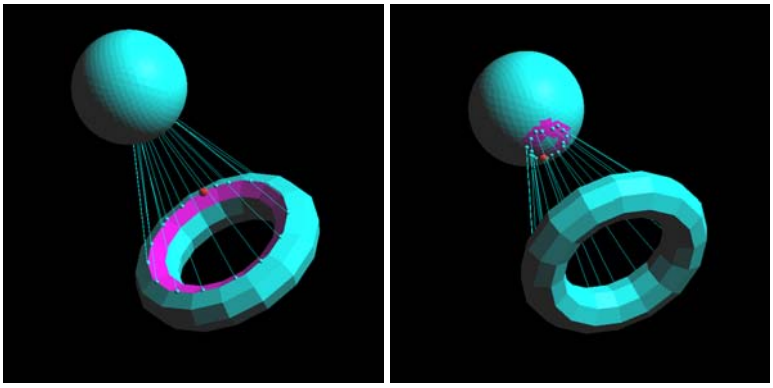


Figure 44: The local minimum distances between a torus and sphere.

Discussion

In Figure 45, the normal cone method finds potentially interesting points on the legs, head and body of the cow compared to the one point provided by the global minimum. Of course, the shortest of these minima can also be chosen as the global minimum, if that measure is needed. In the next chapter, we explore how these local minimum distances provide useful guidance for haptic algorithms.

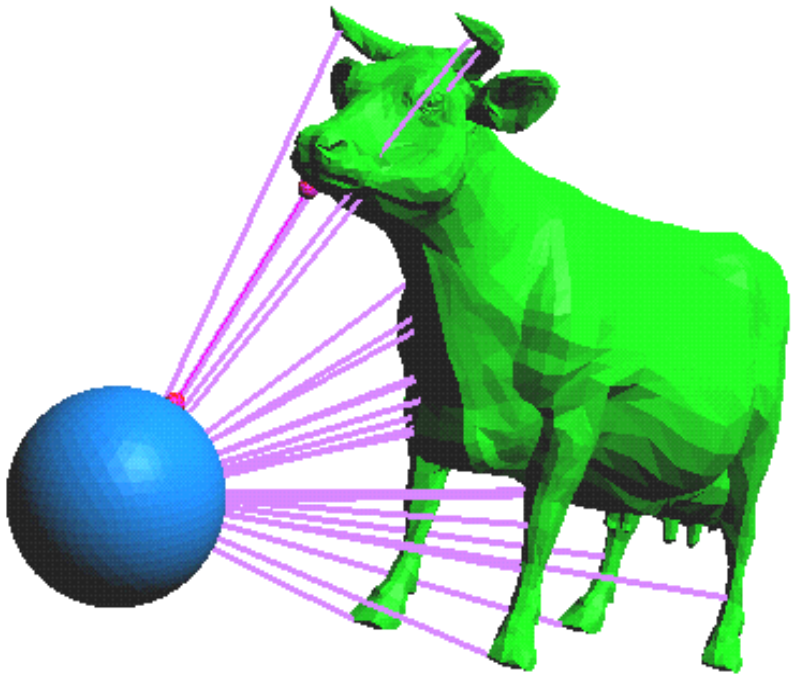


Figure 45: The local minimum distance returns more information about the distance between models than the global minimum distance.

CHAPTER 11

SIX DEGREE-OF-FREEDOM HAPTIC RENDERING OF COMPLEX POLYGONAL MODELS

Spatialized normal cone hierarchies find the *local minimum distances* (LMDs) between polygonal models. The work in this section adapts the SNCH computation to a haptic rendering system for polygonal models by focusing computational resources on the portions of the models that are nearly in contact.

Rather than finding forces that move models apart once they have collided, this haptic rendering algorithm prevents collisions by applying preventive forces as models approach each other (Figure 46). This technique is appropriate for representing interactions between models since allowing models to penetrate each other violates a real-world constraint we are interested in maintaining.

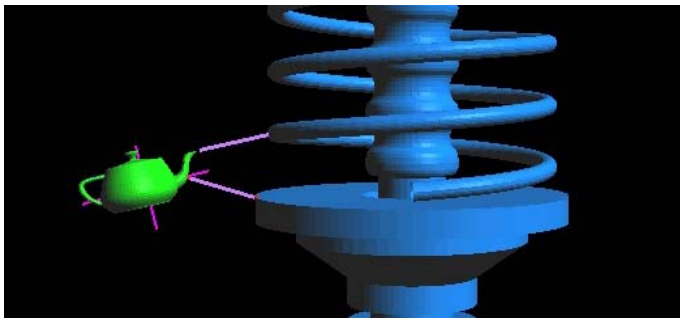


Figure 46: The SNCH approach finds local minimum distances between models and applies repulsive forces to prevent contact.

Some distinguishing characteristics of a haptic rendering environment using the SNCH approach are as follows:

- polygonal models of arbitrary shape can be used in the virtual environment.
- elements of the scene can be moved or added and deleted without requiring substantial preprocessing.
- environments with large number of triangles can be used, increasing the accuracy of simulated model interactions.

System Overview

The haptic system is based on a Sensable 6-DOF PHaNTOM haptic interface (Figure 47). The computations run on a dual processor Pentium 4 2.4 GHz Linux computer with a gigabyte of RAM and a GeForce 4 Ti 4400 graphics card. The application is multithreaded, with the haptic force computation thread running at a high priority to ensure fast update rates.

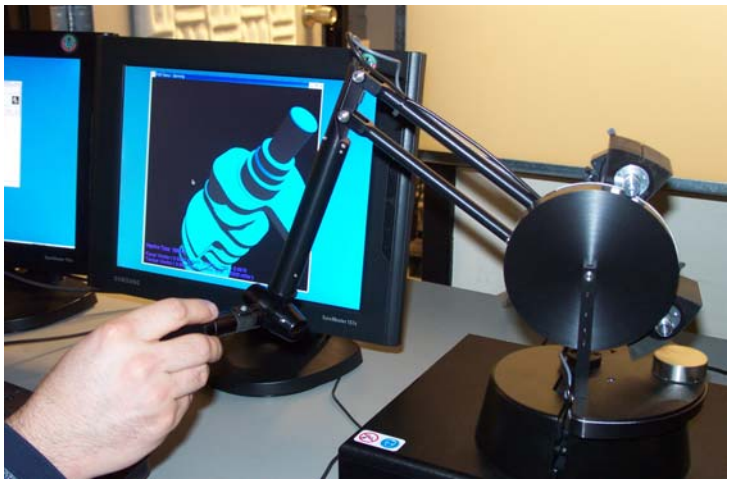


Figure 47: The 6-DOF Phantom used in a virtual prototyping session.

Approach

We adopt the approach of the Boeing voxel sampling virtual prototyping system[49], which prevents models from colliding rather than moving them apart once interpenetration has occurred. This has several advantages:

- accuracy of virtual prototyping is maintained since real-world constraints are maintained,
- lower rates than the typical kilohertz haptic rate are acceptable, since we are not attempting to create the impulsive forces of hard contact,
- minimum distances are faster to compute than penetration depths, allowing haptic rendering of more accurate models.

Local Minimum Distances

Global minimum distances may be rapidly computed between polygonal models using a number of algorithms. However, if these techniques were used in a haptic rendering system, the global minimum would generate only a single penalty force at a time. This force could rapidly change direction, creating haptic instabilities. Alternatively, one could easily imagine modifying a distance computation to return all pairs that are within a certain distance, rather than just the global minimum. However, this could potentially create large numbers of penalty forces, which would swamp the haptic computation (Figure 48).

We argue that an appropriate solution is to compute the local minimum distances between models. Imagine two models that have just collided. This collision can be represented at a single point on each surface (even for manifold contacts, a single point encapsulates that area of contact). If the models move apart, this pair of points tracks the

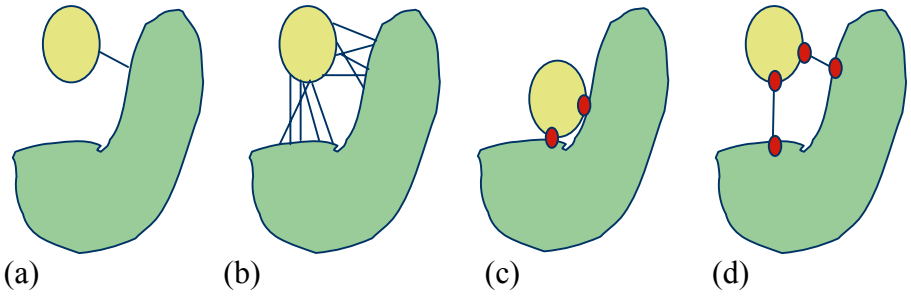


Figure 48: Different distance possibilities for force computation. (a) The global minimum distance. (b) All pairs within a distance. (c) Contact points between two colliding models. (d) Local minimum distances.

local minimum distance and represents the potential future contact between entire sections of these two models. Additional pairs of contact points for those sections are redundant predictors of future contacts for those regions, thus the local minimum distance pairs are adequate. This formulation keeps a manageable number of contacts for the haptic computation, yet is complete enough to safely predict all potential contacts.

Modifying the LMD Computation

We use a modified LMD computation based on the spatialized normal cone hierarchies to quickly determine all the potential areas of contact. The main modification is to introduce a cutoff distance that prunes pairs of nodes that are further apart than this distance. This is appropriate for haptic rendering, where we are interested in computing penalty forces only for models in proximity (Figure 49).

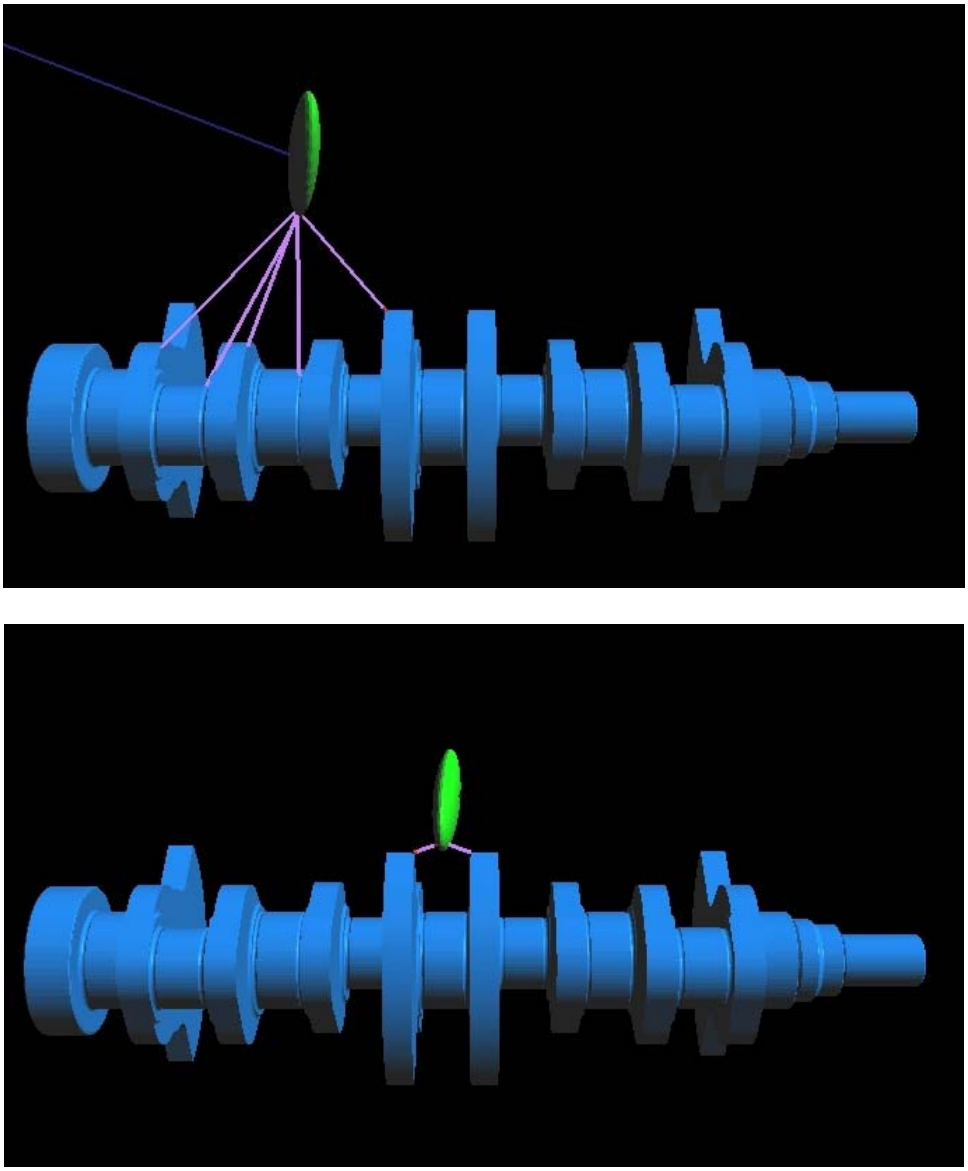


Figure 49: Finding all LMDs will create forces between far apart portions of the models, as shown in the top image. Using a small cutoff distance in the lower image removes unnecessary LMDs from consideration and controls the onset of forces.

Forces and Torques

At each time step in the haptic rendering loop, the haptic rendering algorithm computes the LMDs that are closer than the cutoff distance between the model that is controlled by the haptic interface and the rest of the models in the scene. Each LMD is considered a virtual spring with a rest length equal to the cutoff distance. Each spring is attached to the models by the pairs of points that form the LMD. The force applied to the moving model is then

$$\begin{aligned} \vec{f}_i &= (cutoffDist - \|LMD_i\|)LMD_i \\ \vec{F} &= \sum_i -k\vec{f}_i \end{aligned} \quad \textbf{(69)}$$

and the torque is

$$\vec{\tau} = \sum_i \vec{p}_i \times \vec{f}_i \quad \textbf{(70)}$$

where $\vec{p}_i$ is the distance from the center of mass to the tracking point on the model for the local minimum distance.

The center of mass and the first-order moments are approximated by the geometric extent of a PQP generated, oriented swept sphere bounding box surrounding and approximating the shape of the model. Values that are more precise could be easily used when available.

The repulsive forces between models begin at zero at the cutoff distance, so LMDs that are created and destroyed as sections of the two models approach the cutoff distance only modify the total force and torque a small amount. Furthermore, since we

are not attempting to render the forces of hard contact, only guiding the placement of models, the springs can be fairly soft, smoothing the haptic rendering.

Preprocessing

The LMD computations require precomputing a spatialized normal cone hierarchy for each polygonal model in the virtual prototyping environment. However, models in the scene can be added and deleted, or moved around interactively, without needing further precomputation. The preprocessing step takes a few seconds for models of several thousand triangles.

Results

We have tested our algorithms on a variety of models. The 6-DOF force feedback allows the model controlled by the haptic device to slide around the objects in the stationary scene, providing good intuition for the user (Figure 50). We were able to explore concave portions of the stationary model, with repulsive forces keeping us from all the potential contact areas (Figure 51).

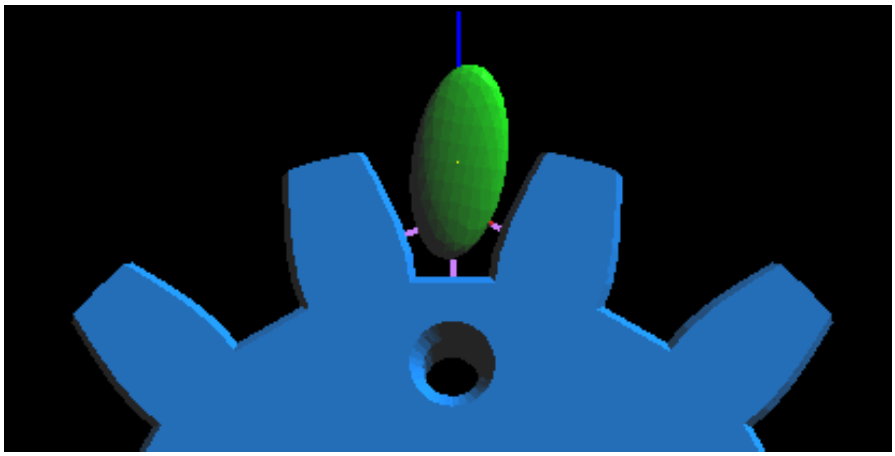


Figure 50: Users are able to feel translation forces and torques generated by interacting arbitrary models.

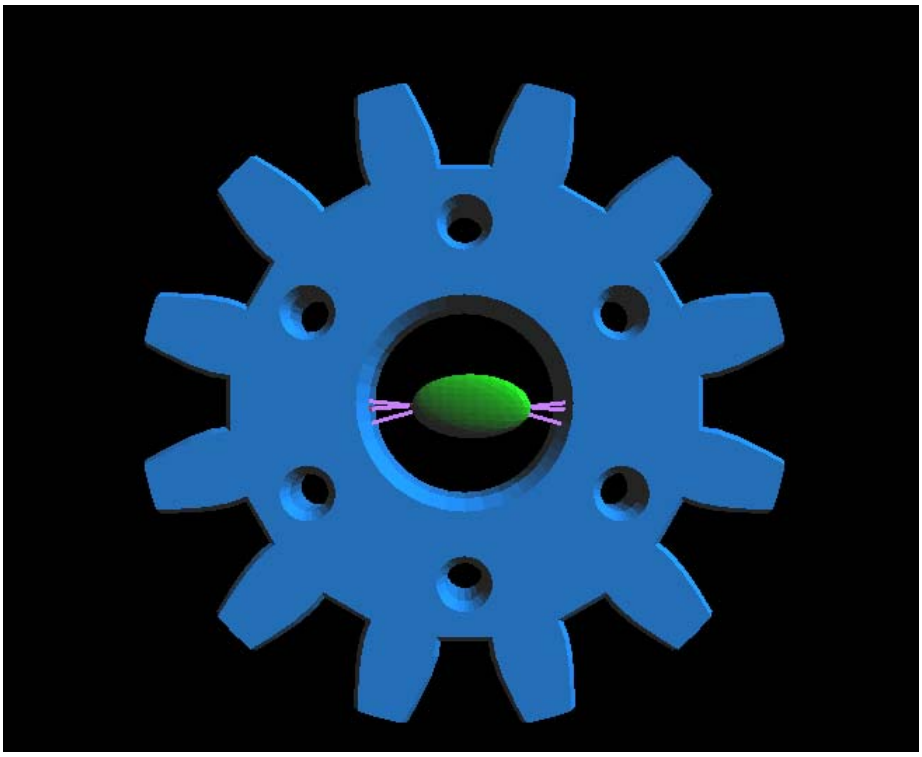


Figure 51: The technique handles concave regions of models.

It is difficult to give a chart with timings for the rendering of these models, since the computation time varies with the cutoff distance, the number of LMDs found, and the relative configuration of the models. Instead, Figure 52-Figure 54 show a variety of sample interactions with the polygon counts, timings, and number of LMDs. Typically, haptic rates in the hundreds of Hertz are achieved between models with hundreds and thousands of polygons.

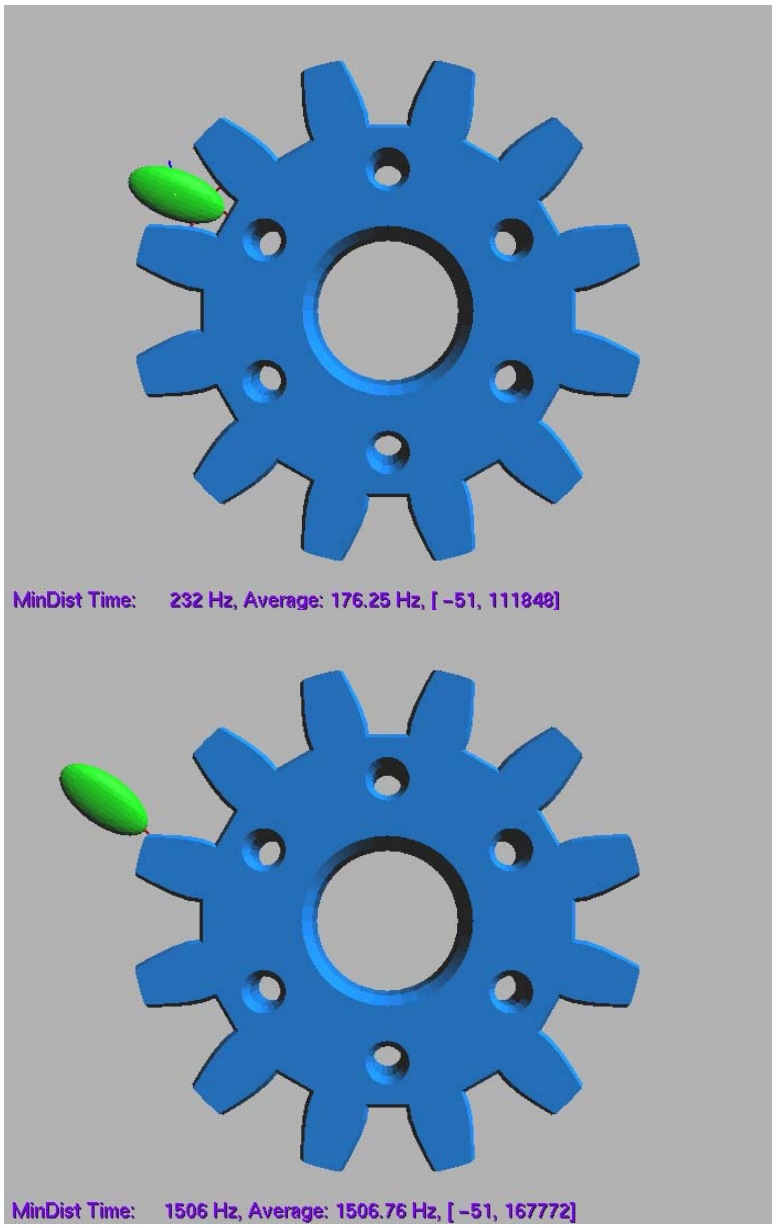


Figure 52: A disc model interacting with a gear

Image	Model (# tris)	Scene (# tris)	# of LMDs	Rate (Hz)
a	Disc (512)	Gear (6302)	3	176
b	Disc (512)	Gear (6302)	1	1506

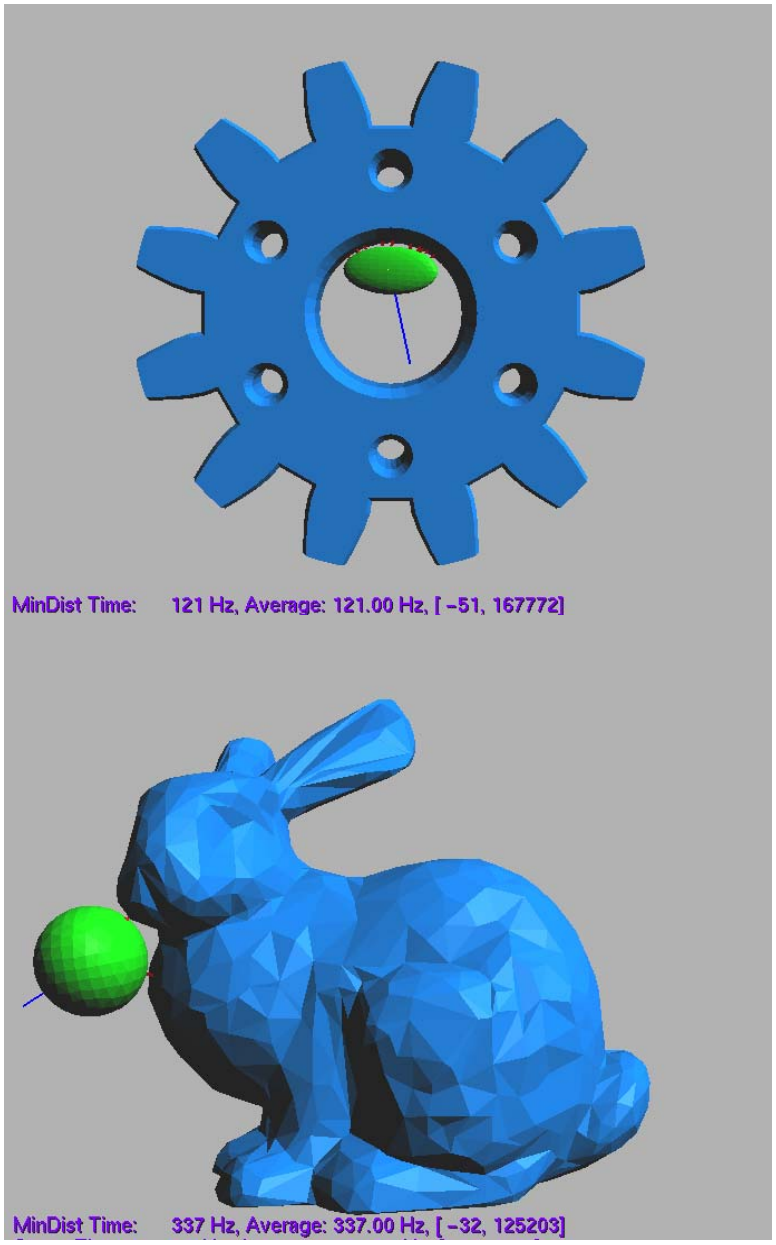


Figure 53: Mechanical and organic models can be used.

Image	Model (# tris)	Scene (# tris)	# of LMDs	Rate (Hz)
c	Disc (512)	Gear (6302)	10	121
d	Sphere (128)	Bunny (2204)	2	337

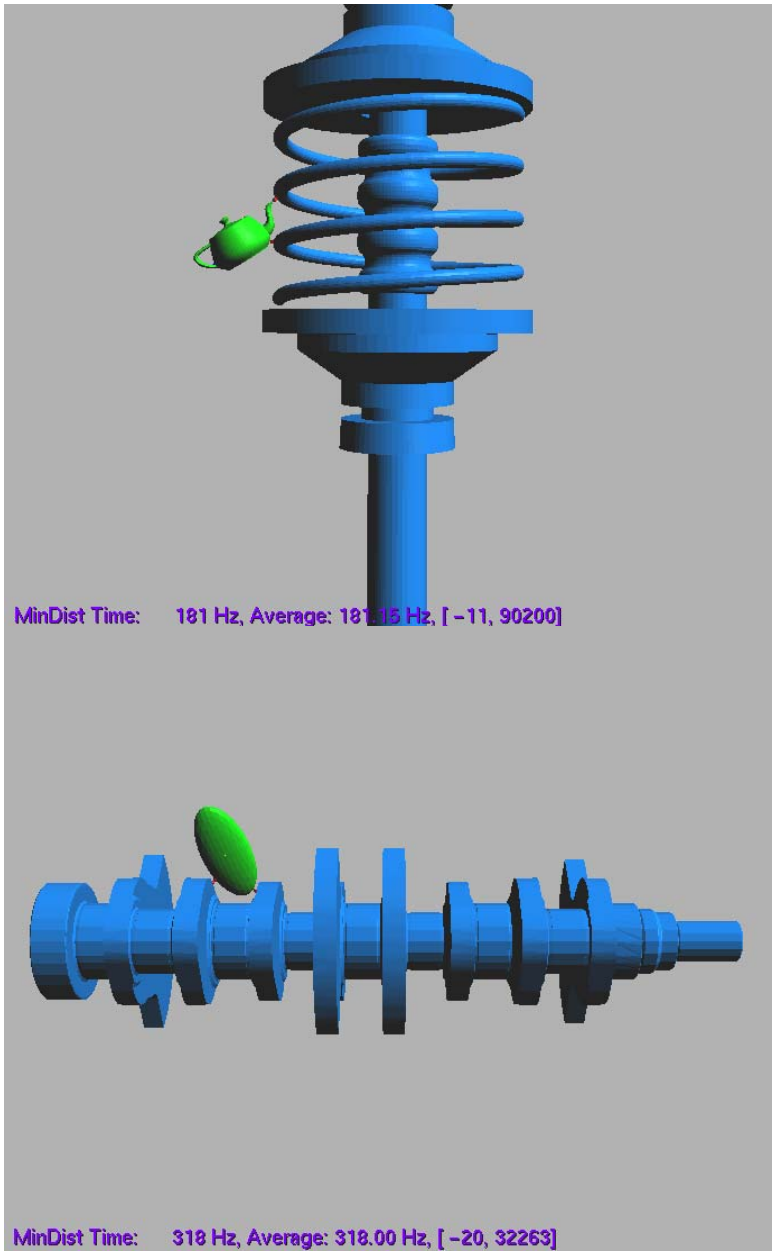


Figure 54: Larger models scale well in this system.

Image	Model (# tris)	Scene (# tris)	# of LMDs	Rate (Hz)
e	Teapot (5648)	Spring (23578)	2	181
f	Disc (512)	Crankshaft (12802)	2	318

CHAPTER 12

SIX DOF HAPTIC RENDERING WITH LOCAL DESCENT

In the last chapter, we computed *local minimum distances* (LMDs) between polygonal models using *spatialized normal cone hierarchies* (SNCHs). In this chapter, this global search for LMDs is augmented with a local gradient search for maintaining LMDs during haptic rate interactions. This multistage approach provides much faster computation of the LMDs and allows haptic rendering of environments with much more complicated models.

Approach

The augmented approach adds a local search to speed intermediate time steps between global updates. The haptic rendering algorithm first computes all LMDs within a cut-off distance using the global SNCH search. These LMDs are fed to the local update thread, which performs local gradient descent on the LMDs given new positions of the models. The updated LMDs are used to compute forces and torques repelling the models.

The local update works as fast as it can on the LMDs it knows about. Concurrently, the global search computes new LMDs. When it finishes a time step, it notifies the local search that new LMDs are available. The local search then updates these new LMDs to the current model positions and continues local updates.

Local Search

A pair of points, one on each model, forms each LMD. After a model moves, the local search algorithm looks at the neighborhood around each LMD point and computes the distances between all the triangles in one neighborhood and all the triangles in the other model's neighborhood. If any of these triangle pairs are closer than the current LMD, then the search continues with those triangles' neighborhoods until the minimum distance converges. The points that form this new minimum distance are the updated LMD.

If the last LMD point was on the face of a triangle, then the local neighborhood is defined to be the triangle plus the three triangles bordering its edges. If the last point was on an edge, only the two triangles that share that edge are part of the local neighborhood. When the last point was at a vertex, all triangles that share that vertex are searched for a new LMD (Figure 55).

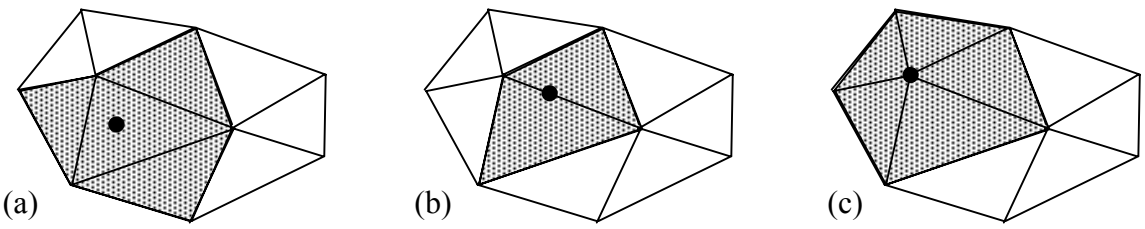


Figure 55: The three possible neighborhoods to be checked depend on the minimum distance point from the last time step. In (a), the minimum distance point was on a triangle face, and it and all its adjoining triangles form the neighborhood. When the point from the previous step was on an edge, as in (b), the triangles that share that edge form the neighborhood. The final case (c) is for a vertex, where all triangles that share that vertex form the neighborhood.

Computational Efficiency

The number of neighborhoods that must be checked varies with the model resolution and the movement of the models. For models with reasonably formed triangles, the number of triangles searched on one model grows roughly as the $\sqrt{n}$, where *n* is the number of triangles. Figure 56 shows the resolution quadrupling and the number of triangles crossed growing by a little more than a factor of two. In addition, for haptic rendering running near 1000 Hz, the number of triangles crossed is usually small.

Preprocessing

The local search routine uses local topological connectivity between the triangles of the models. This information can be derived from models consisting of just triangle lists. Most of the models used in our examples went through a one-time conversion from triangle list data structures to vertex-edge-triangle lists with neighbor information.

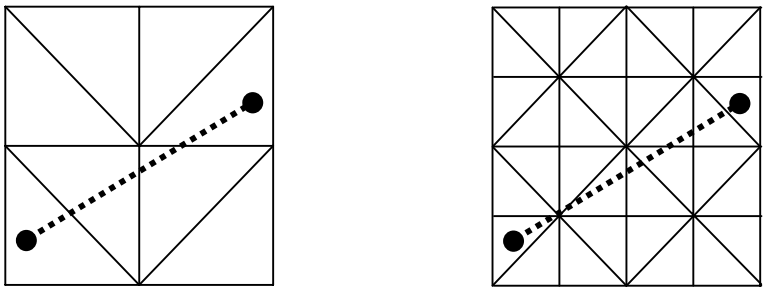


Figure 56: The number of triangles crossed grows roughly by $\sqrt{n}$, where n is the number of triangles in the model. (a) This model has eight triangles and four are crossed. (b) There are four times as many triangles in this model, but roughly twice as many are crossed.

System Architecture

This type of application would be difficult to write as a single thread of computation. We use three threads: a global search thread, a local update thread, and a graphics thread. This architecture allows us to restrict the computational load of the graphics and global threads, and let the local update run as fast as possible. On a two-processor system, this translates into the local update getting one processor to itself and the other threads sharing the other processor.

Results

The local search algorithm computes updated forces and torques at kilohertz rates. When model complexity grows, the global search tends to slow down, but the local update speed is mostly dependent on the number of LMDs, not the complexity of the model.

We instrumented the local update thread to record the time to compute the local update, the number of triangle pairs searched during the local update, and the time for the global search to compute the LMDs. Figure 57-Figure 59 show these results for a variety of model-model interactions. For all these examples, the top graph represents the local update time, the middle graph the number of triangle-pairs searched during the local update, and the bottom graph the time for the global LMD computation to update. The local update and searched triangles graphs do not cover the full extent of the global search graph since the data were stored in a circular buffer and the fast updates of the first two graphs filled the available space.

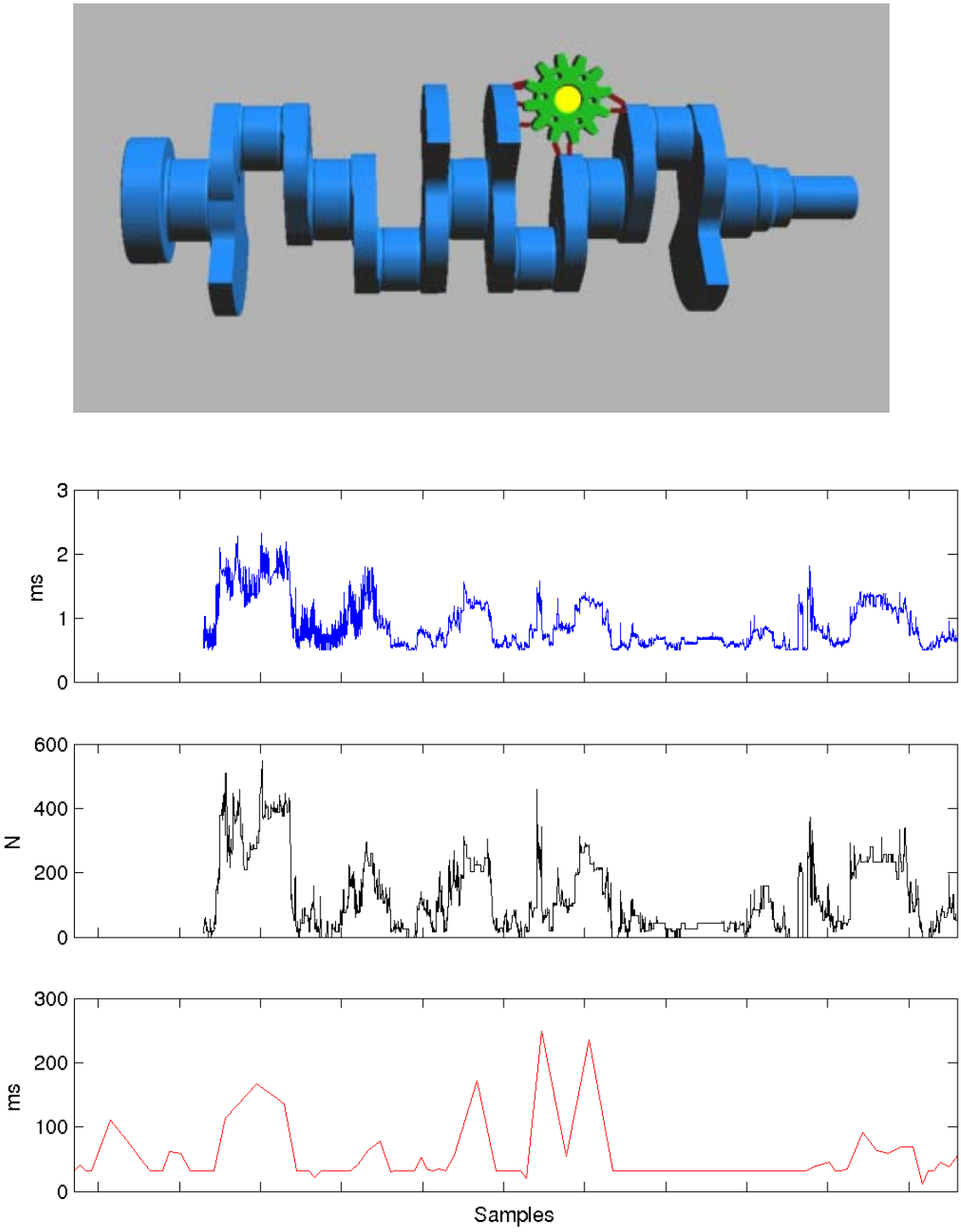
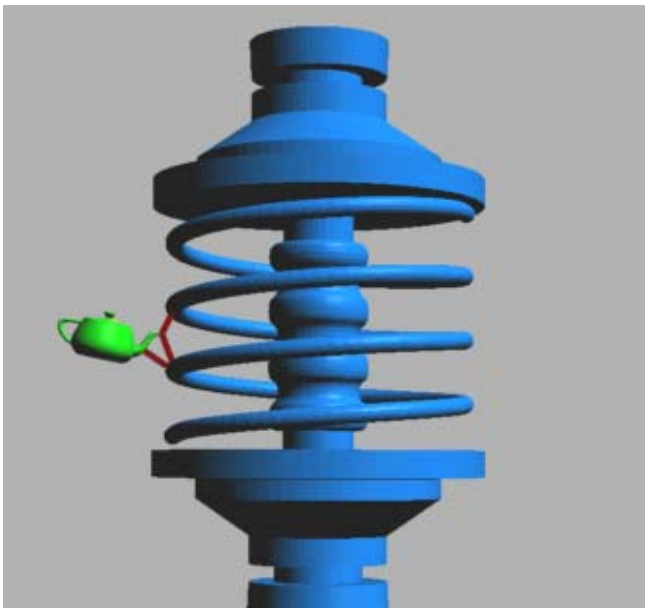


Figure 57: The crankshaft model has 45,000 triangles and the gear has 6,300 triangles. The local update time correlates well with the number of triangles searched. The bottom graph shows the computation time for the global search to find the LMDs. Without the local update, haptic interaction would have been unstable and slow.



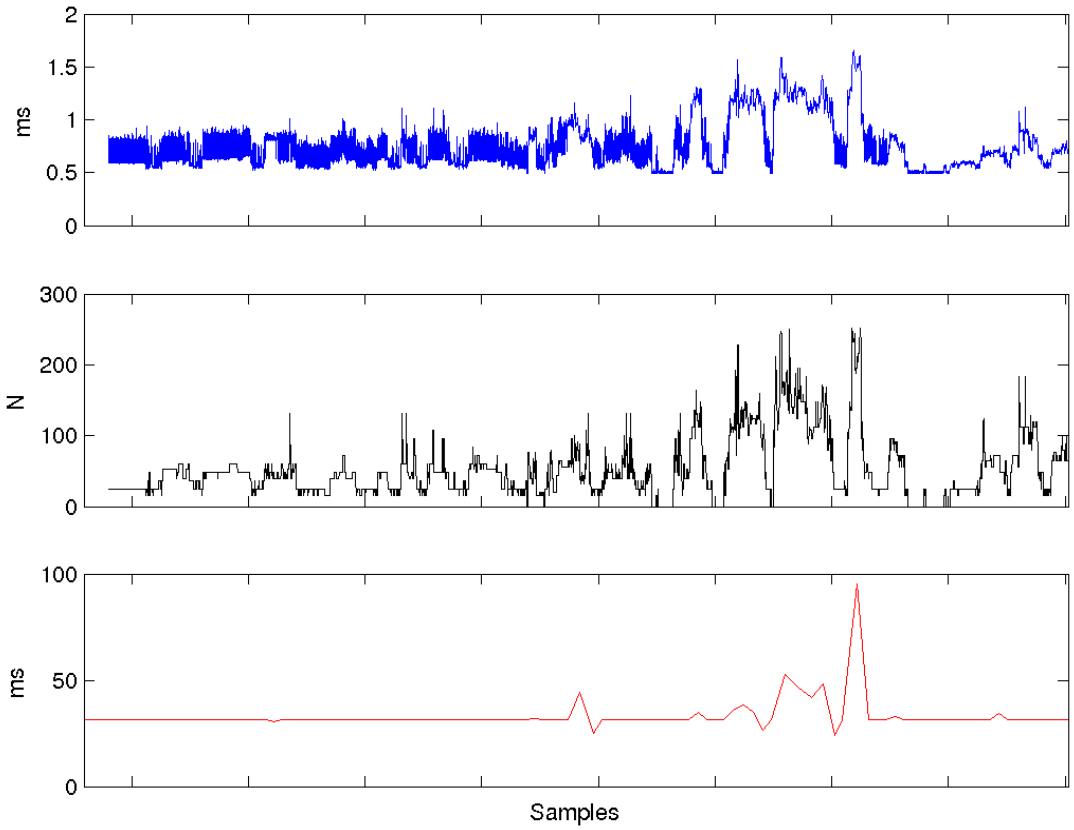


Figure 58: The spring model has 23,500 triangles and the teapot has 5,600 triangles. Even though the models are high resolution, typically there were only a few LMDs to track, and the local update was able to maintain a high update rate.

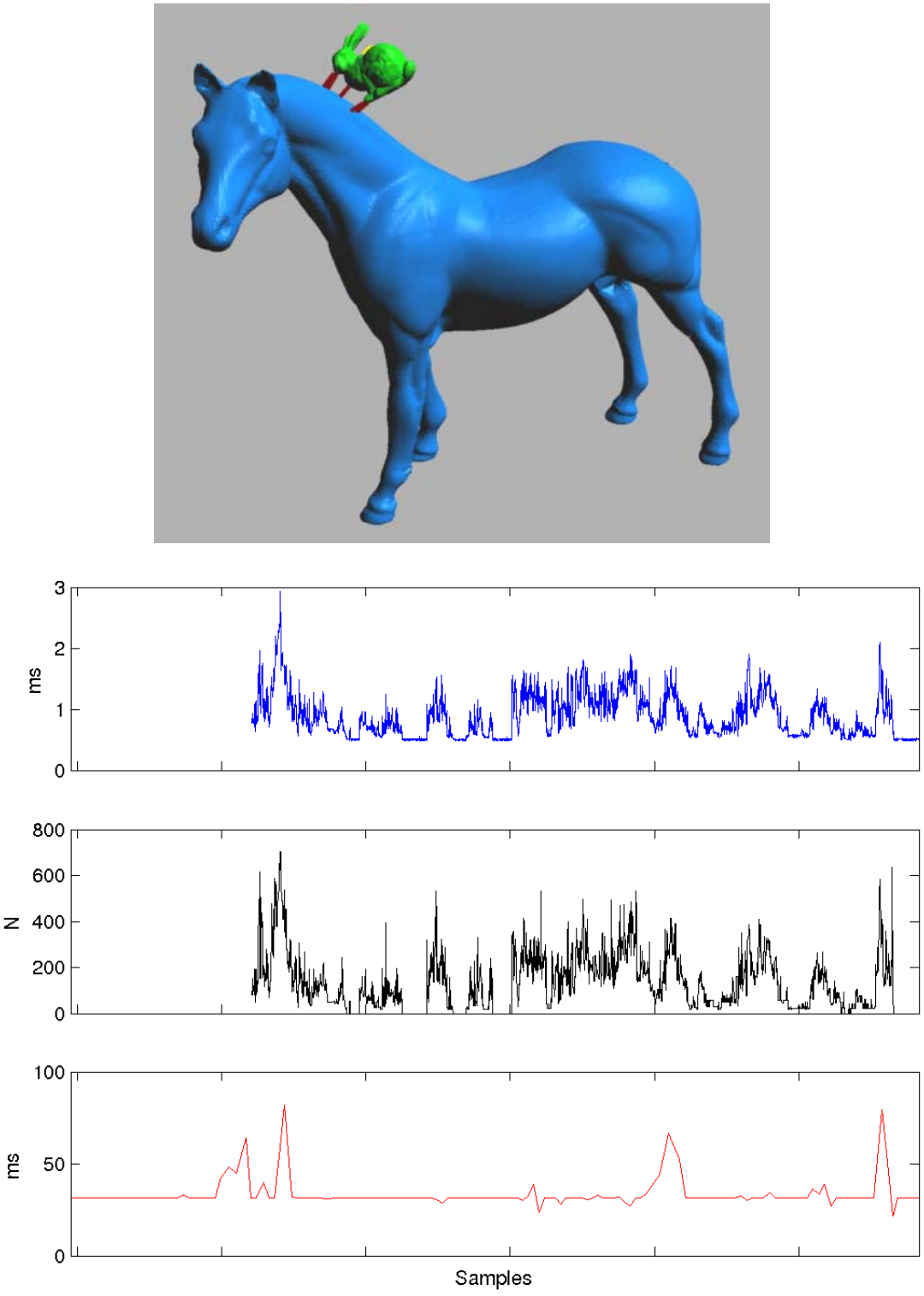


Figure 59: The horse model has 97,000 triangles and the bunny model has 2,200 triangles. The finely detailed surfaces can produce nearly redundant local minima. In this example, the algorithm still updates the LMDs and forces at around 1 kilohertz.

The forces of interaction feel smooth. Figure 60 shows the magnitude of the translational forces during haptic interaction between the horse model and the bunny. The large-scale bumps are from moving the bunny model around the horse and bumping against it. Smoother responses are possible when continuously pressing the two models together. Even in this example, there is not much high-frequency force response, as would be expected if the control system became unstable.

Discussion

The use of local updates dramatically improved the performance of the system over just using the SNCH global search. The haptic rendering system can use models that are an order of magnitude larger at update rates an order of magnitude faster than with just using global search for LMDs. The addition of local LMD updates makes the force computation much less dependent on the configuration of the two models. The resulting algorithm is a high performance haptic rendering system suitable for augmenting virtual reality interactions.

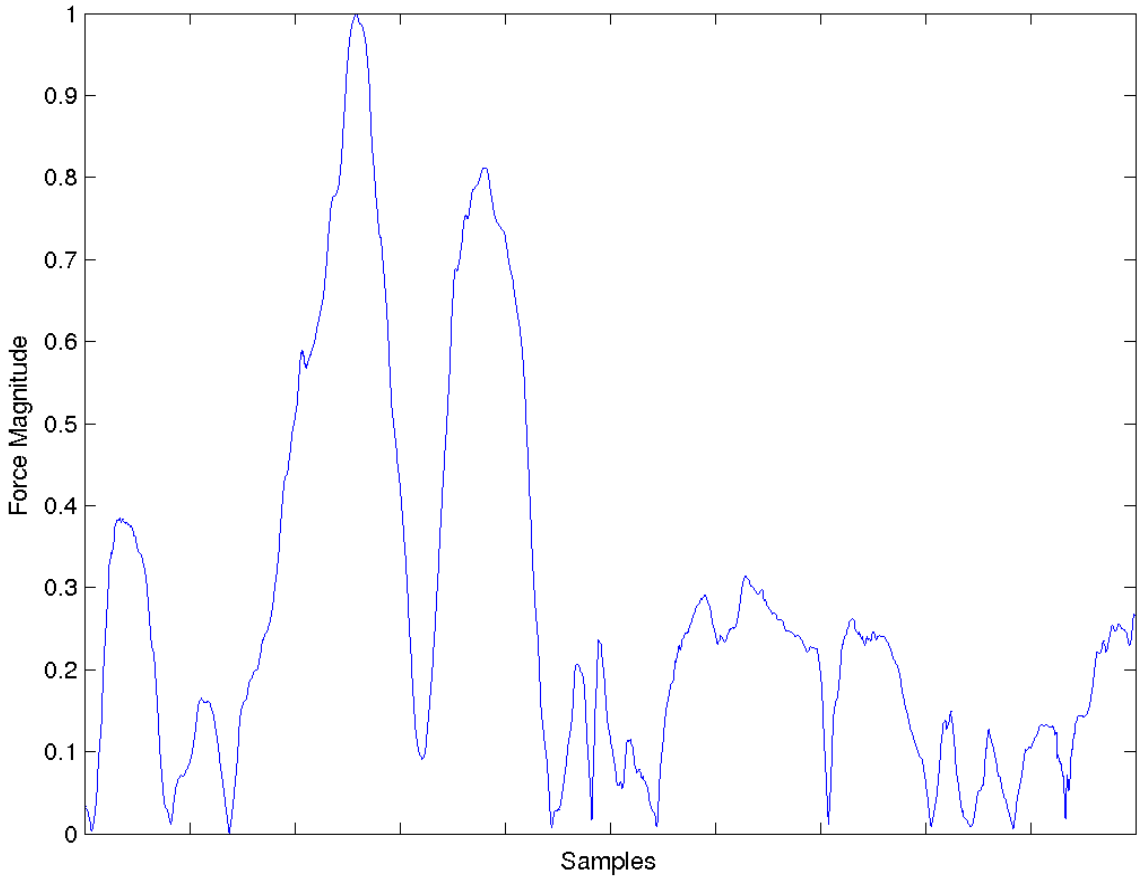


Figure 60: This graph shows the magnitude of the translation forces during the horse-bunny interactions.

CHAPTER 13

A VIRTUAL PROTOTYPING APPLICATION

The last three chapters have developed techniques for computing local minimum distances between polygonal models and a multithreaded system applying these techniques to haptic rendering of complex scenes. In this chapter, we apply these ideas to a virtual prototyping system for polygonal models.

While mechanical model design increasingly relies upon computer-aided design (CAD) and sophisticated simulation programs, physical prototypes still play an important role in design evaluation. Since physical prototypes are expensive to build, and may take significant time to manufacture, virtual prototyping environments attempt to replace as much functionality of the physical prototypes as possible with a virtual prototype.

Accessibility is a design evaluation task that is difficult to simulate on a computer. Two main reasons preclude easy automatic simulation:

1. computation of a collision-free path for complex models is difficult and time-consuming,
2. modeling human manipulation capabilities is difficult.

We propose a haptic system for virtual prototyping that allows human guidance and intuition in developing a collision-free path between virtual models. This type of

system provides the intuitive usability of a physical prototype, yet retains the low-cost and time advantages of a computer model.

Virtual Prototyping Background

Virtual prototyping of accessibility tasks is closely related to the area of path planning. The main distinction is that in virtual prototyping, there is some assumption of human involvement, whereas path planning is usually more of an automatic technique.

Path planning methods fall into two main categories, global methods and local methods. Global methods try to sample the configuration space of the model and the environment, and then connect together collision-free instances into a collision-free path[50][51][53]. Local methods use local repulsion techniques to avoid collisions, while being drawn towards a distant goal[3]. However, these local methods can converge to local minima and never reach the goal. Our haptics system is similar to the local approach, but human guidance pushes models past local minima.

Haptics has been proposed as a virtual prototyping interface in prior work. Hollerbach et al. [54] computed fast penetration depths between a point and a spline model to create a sensation of contact with the model, as did researchers at Ford Motor Company in [35]. Nelson developed haptic rendering of moving linkages using a three-DOF haptic interface[55].

McNeely used a 6-DOF device to manipulate a point-sampled model with a large-scale voxelized environment[49]. The environment is static; however, this approach guarantees a worst-case computation time, important for reliable haptic rendering. They report that they were able to use haptics to find collision-free paths in complex environments for which global path planning algorithms failed.

Virtual Prototyping Approach

Since we compute LMD's while the moving model is still some distance from the environment models, haptic forces are used to guide the moving model away from collision with the environment. The onset distance for forces is adjustable, so the user can decide how much clearance between models is desired during testing. In general, the LMD's tend to approximate the local distance field between the models, and the forces tend to push the moving model towards the medial axis between the models. Since the medial axis is the surface of maximum clearance between models, these forces tend to guide the moving model towards the safest path (Figure 61).

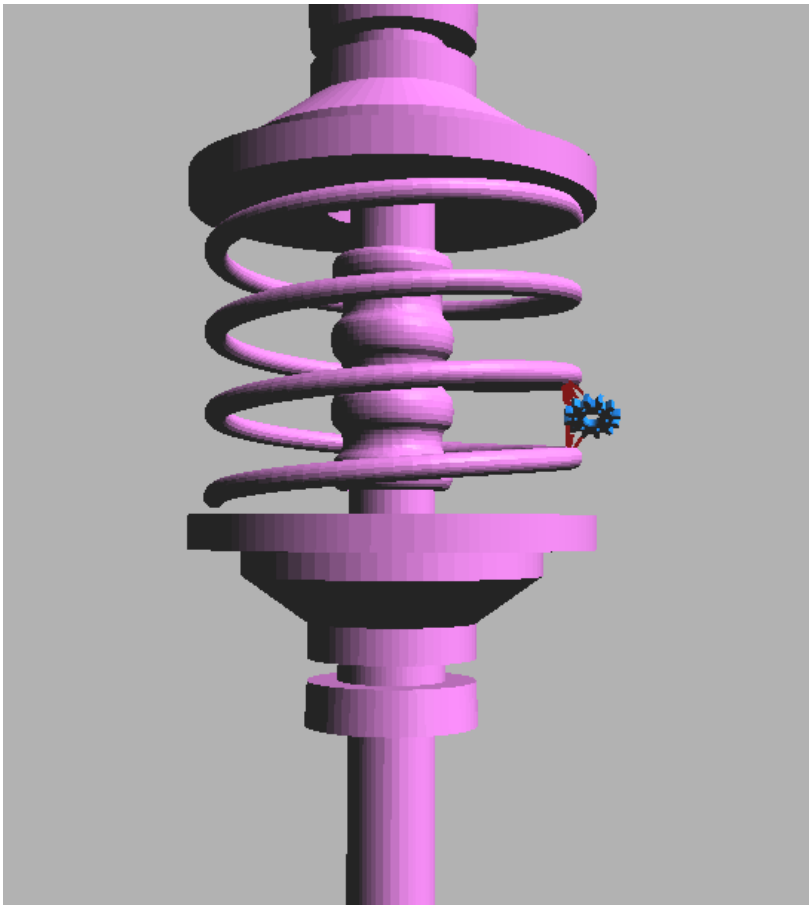


Figure 61: The LMD's provide guidance in regions of limited clearance.

While the test object is being moved by the haptic interface, its position and orientation are stored in a buffer. This buffer allows the motion of the test object to be played back for review, analysis, or further modification.

If the moving model is forced to penetrate an environment model by the user, the simulation is no longer valid. A collision state is detected and the simulation is rolled back, using the stored positions and orientations in the buffer, until the model state is valid. The simulation can then resume, and the user can try new approaches for finding a collision-free path. This means that the path stored by our virtual prototyping program is always valid, and if the moving model can reach its goal, the problem has been solved.

A collision state is detected by finding a LMD less than some specified parameter between the moving model and the environment. Changing the collision state distance allows the user to adjust for tessellation error in the models or for other desired constraints in the clearance of the collision-free path.

Interface

The main interface is, of course, the 6-DOF haptic interface. After loading models into the environment, the position of the currently selected model is controlled by the user moving the haptic interface. The selected model is changed with keyboard commands, so any model in the environment is freely movable by the haptic interface.

Keyboard commands also control the recording of the collision-free path, stopping of recording, and visualization of the path in playback mode.

The current set of LMD's is displayed as lines between the two models. They help provide feedback cues to the relative positions of the two models in the absence of stereo viewing.

Results

We tested our virtual prototyping system with a variety of models. In the tests, we threaded one moving model around and inside the environment model. In all the cases, we were able to intuitively find a collision-free path to accomplish the goal.

Gear-Spring Part

In the first test, we used a gear model with 6,300 triangles and a spring part model with 23,500 triangles. The goal was to have the gear enter the spring, traverse down the body, and then exit the spring. There was limited clearance between the gear and the spring and spring body, so without haptic feedback this would have been a difficult task (Figure 62).

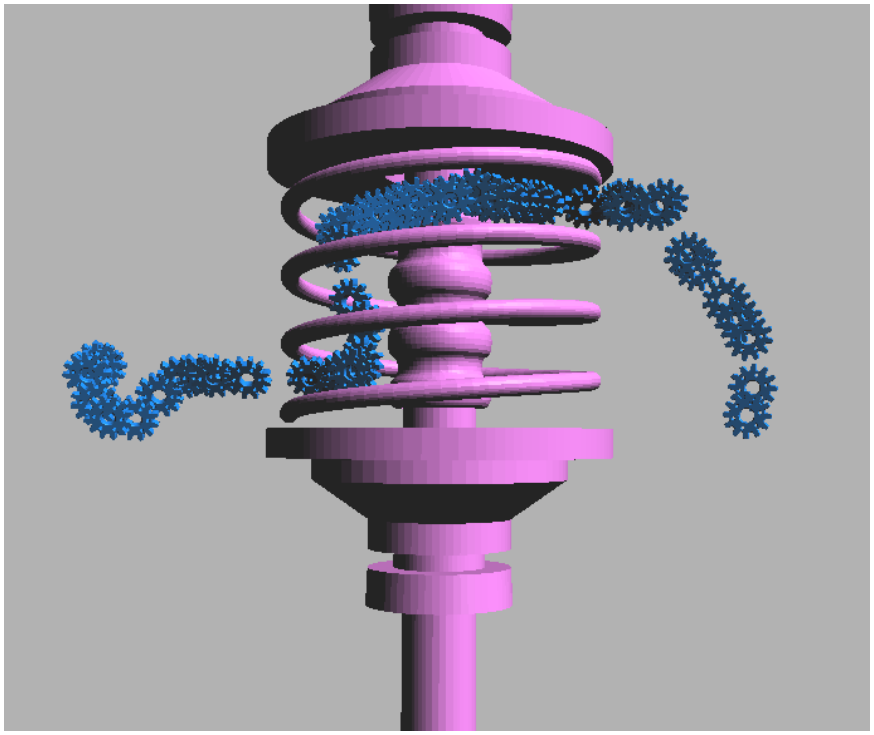


Figure 62: The gear had to rotate to slip between the coil and center axis of the environment model.

Crank-Holes-Teapot

In this example, we used a crank model with 45,000 triangles, a three-holed block with almost 12,000 triangles, and a teapot model with 5,600 triangles. We used the haptic interface to position the block and the teapot in such a way that there was not a clear path from one hole to the next. The goal in this test was to thread the crank model through all three holes while avoiding the teapot.

The haptic interface provided enough cues to the user to find a path out of the middle hole and to tilt around the teapot, even though that portion of the path was occluded by the teapot during the test. (Figure 63 shows the view during the test and a tilted view after the test for the path visualization.)

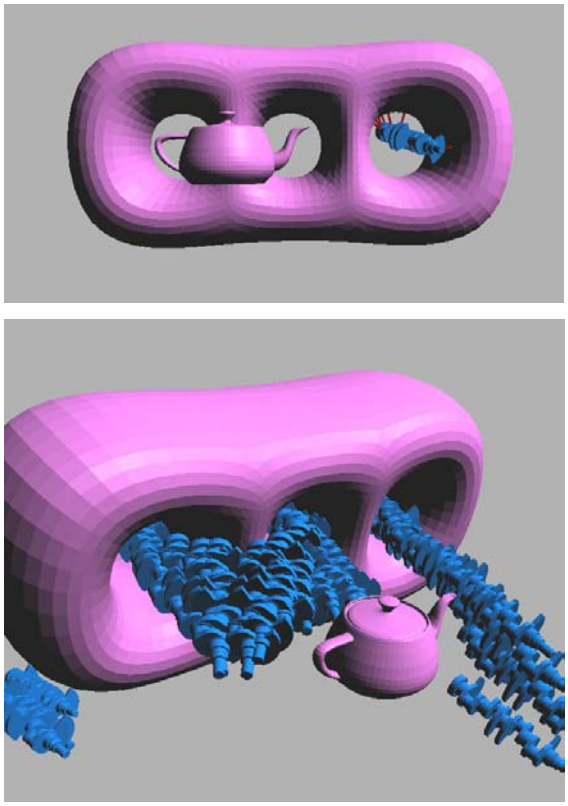


Figure 63: Haptics guides the crank model through the holes while avoiding the teapot model.

Helicopter-Rocker

The final test used a helicopter model with 113,000 triangles and a rocker model with 40,000 triangles. For this test, we wanted to pass the rocker through the open window of the helicopter, around the interior, and back out (Figure 64). The haptic rendering system was able to provide useful feedback during this test, creating a collision-free path under user guidance.

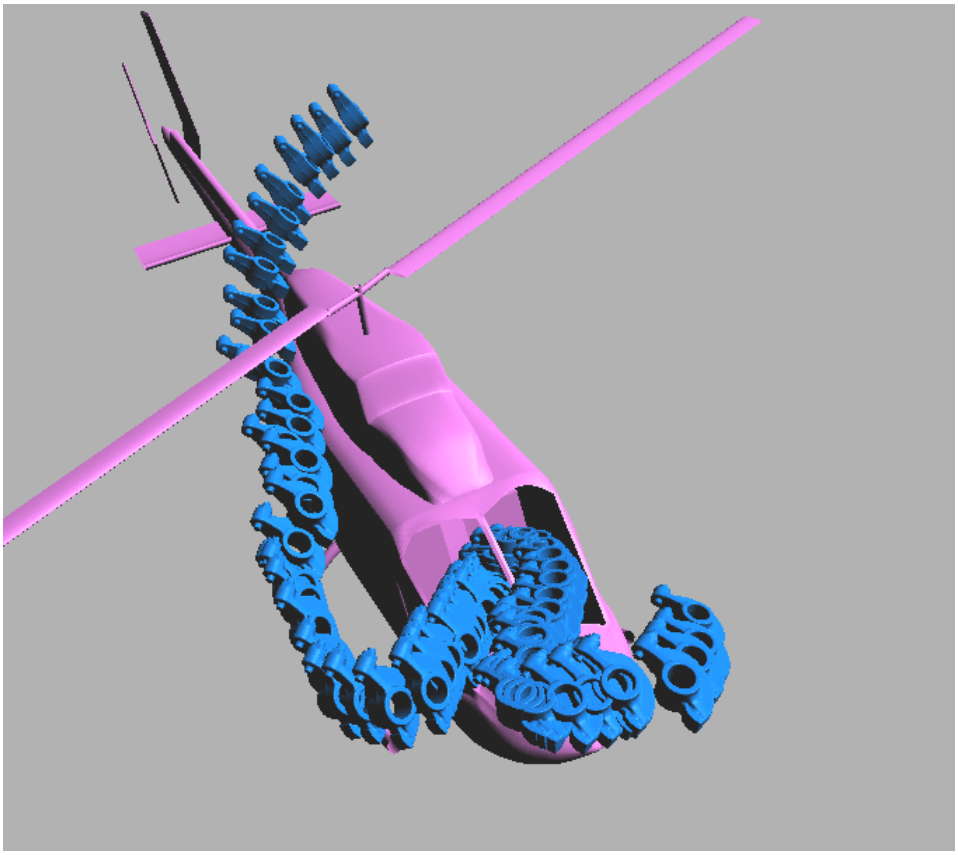


Figure 64: This test involved models with over 150,000 combined triangles.

Discussion and Conclusion

The presented system advances the state-of-the-art in haptically enhanced virtual prototyping systems by allowing virtual prototyping on general, freely positioned, polygonal models. In addition, our mechanism for rolling back model collision states to a collision-free position and orientation simplifies motion planning by always storing a safe path. The use of adjustable distances for both force onset and collision distance enhances the capabilities of the system by simulating different clearance constraints during the task.

The tests demonstrated the system on a variety of model types and sizes. While the chosen models were not solving an actual real-world problem, the model shapes, resolutions, and task types are representative of the kinds of problems the system can solve.

CHAPTER 14

CONCLUSION

In this dissertation, a number of algorithms are derived for computing the minimum distance between computer models. In particular, one technique using normal cones for solving for a collinearity condition between models has broad application to polygonal and to parametric models. This style of computation seems particularly well suited for haptic applications, where stringent demands on computation rates make predictive methods a necessity. The set of local minima returned by normal cone methods provide greater knowledge of potential future interactions than global techniques, and thus can be used to initialize fast, local methods that can quickly react to interactions between models. The haptic applications developed using these techniques work on polygonal and smooth models, and at new levels of model complexity compared to previous techniques. These haptic applications provide the building blocks for virtual prototyping applications involving models of realistic complexity.

Future Directions

An area of haptic rendering not addressed herein is rendering of deformable models. This topic is important for medical simulation, where a person is touching or cutting models of soft tissue. The normal cone approach for sculptured surfaces should

work well in this context, since unlike most methods, it does not require a lengthy preprocessing of a static model to work.

Normal cones, even with demonstrated successes when applied to distance and haptics problems, are still in the early stages of development. In global distance computations, spherical bounding hierarchies were a standard technique for some time[14], but the introduction of bounding volumes with variable aspect ratios, such as swept sphere volumes[17], dramatically improved computation speed. A similar development should be possible with normal cones, by bounding vector spreads and geometries with more general volumes that still possess fast tests for orthogonality or collinearity.

Additionally, since distance finding is a basic operation on virtual models, the techniques developed here have broad application to problems in simulation, animation, and modeling. Even more broadly, the normal cone approach can be seen as quickly and robustly solving a system of constraints, in this case for collinearity between models' surface normals. However, other kinds of constraints should be solvable in a similar style, so the normal cone approach should prove useful in other applications solving constraints based on surface properties.

REFERENCES

[1] D. Baraff, "Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation," *Computer Graphics*, vol. 24, no. 4, pp.19-28, Aug. 1990.

[2] J.E. Bobrow, "Optimal Robot Path Planning Using the Minimum-time Criterion," *IEEE Journal of Robotics and Automation*, vol. 4, no. 4, pp. 443-450, Aug. 1988.

[3] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90-98, Spring 1986.

[4] S. Quinlan, "The Real-Time Modification of Collision-Free Paths," Ph.D. dissertation, Dept. of Mechanical Engineering, Stanford University, 1994.

[5] D. Dobkin and D. Kirkpatrick, "Determining the Separation of Preprocessed Polyhedra - A Unified Approach," *Proc. 17th International Colloq. Automata Lang. Program*, pp.400-413, 1990.

[6] H. Edelsbrunner, "Computing the Extreme Distances Between Two Convex Polygons," *Journal of Algorithms*, vol. 6, no. 2, pp. 213-224, June 1985.

[7] Y. Adachi, "Touch and Trace on the Freeform Surface of Virtual Object," *Proc. IEEE Virtual Reality Annual Intl. Symp.*, pp. 162-168, 1993.

[8] F. Brooks, M. Ouh-Young, J.J. Batter, and P.J. Kilpatrick, "Project GROPE -- Haptic Displays for Scientific Visualization," *Computer Graphics*, vol. 24, no. 4, pp. 177-185, Aug. 1990.

[9] W.R. Mark, S.C. Randolph, M. Finch, J.M. Van Verth, and R.M. Taylor III, "Adding Force Feedback to Graphics Systems: Issues and Solutions," *Proc. SIGGRAPH' 96*, pp. 447-452, 1996.

[10] D.C. Ruspini, K. Kolarov, and O. Khatib, "The Haptic Display of Complex Graphical Environments," *Proc. SIGGRAPH'97*, pp. 345-352, 1997.

[11] T.V. Thompson II, D.E. Johnson, and E.C. Cohen, "Direct Haptic Rendering Of Sculptured Models," *Proc. 1997 Symposium on Interactive 3D Graphics*, pp. 167-176, 1997.

[12] S. Ehmann, and M. Lin, “Accurate and Fast Proximity Queries Between Polyhedra Using Surface Decomposition,” *Proc. Eurographics*, pp. 500-510, 2001.

[13] M. Ponamgi, D. Manocha, and M. Lin, “Incremental Algorithms for Collision Detection Between Solid Models,” *Proc. of ACM/SIGGRAPH Symposium on Solid Modeling*, pp. 293-304, 1995.

[14] S. Quinlan, "Efficient Distance Computation between Non-Convex Objects," *IEEE Int. Conference on Robotics and Automation*, pp. 3324-3329, 1994.

[15] M. Mortenson, *Geometric Modeling*, pp. 305-317, New York: John Wiley & Sons, 1985.

[16] J. Snyder, “Interval Analysis for Computer Graphics,” *Computer Graphics*, vol. 26, no. 2, pp.121-130, July 1992.

[17] M. Lin, “Efficient Collision Detection For Animation and Robotics,” Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, Univ. of California, Berkeley, 1993.

[18] E. Gilbert, D. Johnson, and S. Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of Robotics and Automation*, pp. 193-203, April 1988.

[19] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, “Fast Distance Queries Using Swept-Sphere Volumes,” *Proc. IEEE International Conference on Robotics and Automation*, pp. 24-28, 2000.

[20] S. Gottschalk, M.Lin, and D. Manocha, “OBBTree: A Hierarchical Structure for Rapid Interference Detection,” *Computer Graphics Proceedings, Annual Conference Series*, pp.171-180, 1996.

[21] P. Schneider, *Graphics Gems I*, p. 607, San Diego: Academic Press, 1990.

[22] M. Lin and D. Manocha, “Fast Interference Detection Between Geometric Models,” *The Visual Computer*, vol. 11, no. 10, pp. 542-561, Nov. 1995.

[23] J. Snyder, A.R. Woodbury, K. Fleischer, B. Currin, and A.H Barr, “Interval Methods For Multi-Point Collisions Between Time-Dependent Curved Surfaces,” *Computer Graphics*, vol. 27, no. 2, pp. 321-334, Aug. 1993.

[24] J. Snyder, "An Interactive Tool for Placing Curved Surfaces Without Interpenetration," *Proceedings of Computer Graphics*, pp. 209-218, 1995.

[25] Y. Adachi, T. Kumano, and K. Ogino, “Intermediate Representation For Stiff Virtual Objects,” *Proc. Virtual Reality Annual Intl. Symposium,* pp. 203-210, 1995.

[26] H. Iwata, and H. Noma, "Volume Haptization," *Proc. IEEE Symp. Research Frontiers in Virtual Reality*, pp. 16-23, 1993.

[27] C.B. Zilles, and J.K. Salisbury, "A Constraint-based God-object Method for Haptic Display," *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pp. 146-151, 1995.

[28] P. Stewart, Y. Chen, and P. Buttolo, "Direct Integration of Haptic User Interface in CAD Systems," *Proc. ASME Dynamic Systems and Control Division*, pp. 93-99, 1997.

[29] A. Gregory, M. Lin, S. Gottschalk, and R. Taylor, "H-COLLIDE: A Framework for Fast and Accurate Collision Detection for Haptic Interaction," *Proc. IEEE Virtual Reality '99*, pp. 38-45, 1999.

[30] D. Baraff, "Interactive Simulation of Solid Rigid Bodies," *IEEE Computer Graphics and Applications*, vol. 15, no. 3, pp. 63-75, May 1995.

[31] A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin, and D. Manocha, "Six Degree-of-Freedom Haptic Display of Polygonal Models," *Proc. IEEE Visualization*, pp. 139-146, 2000.

[32] Y. Kim, M. Otaduy, M. Lin, and D. Manocha, "Six Degree-of Freedom Haptic Display Using Localized Contact Computations," *Tenth Symposium on Haptic Interfaces For Virtual Environment and Teleoperator Systems*, pp. 209-216, 2002.

[33] M. Otaduy, and M. Lin, "Sensation Preserving Simplification for Haptic Rendering," *Proceedings of ACM SIGGRAPH 2003*, pp. 543-553, 2003.

[34] L. Piegl and W. Tiller, *The NURBS Book*, p. 230, Berlin: Springer-Verlag, 1995.

[35] P. Stewart, Y. Chen, and P. Buttolo, "CAD Data Representations For Haptic Virtual Prototyping," *Proceedings of DETC'97*, pp. 249-255, 1997.

[36] S. Smale, "Newton's Method Estimates from Data at One Point," *The Merging of Disciplines: New Directions in Pure, Applied, and Computational Mathematics,* R. E. Ewing, K. I. Gross, and C. F. Martin, eds., pp. 185-196, New York: Springer-Verlag, 1986.

[37] T. McCalla, *Introduction to Numerical Methods and FORTRAN Programming*, New York: John Wiley & Sons, 1967.

[38] E. Cohen, R. Riesenfeld, and G. Elber, *Geometric Modeling with Splines: An Introduction*, Natick, Mass.: A K Peters, 2001.

[39] D.E. Johnson and E. Cohen, "A Framework for Efficient Minimum Distance Computations," *Proc. IEEE Intl. Conf. Robotics & Automation,* pp. 3678-3684, 1998.

[40] G. Elber, "Free Form Surface Analysis using a Hybrid of Symbolic and Numeric Computation," Ph.D. dissertation, Dept. of Computer Science, University of Utah, Salt Lake City, 1992.

[41] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical recipes in C,* Cambridge, U.K.: Cambridge Press, 1997.

[42] M. DoCarmo, *Differential Geometry of Curves and Surfaces*, Prentice-Hall. 1976.

[43] P.M. Hubbard, "Interactive Collision Detection," *Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality*, pp. 24-31, 1993.

[44] J. Tornero, J. Hamlin, and R. Kelley, "Spherical-Object Representation and Fast Distance Computation For Robotic Applications," *Proceedings of the 1991 IEEE Int. Conf. on Robotics and Automation,* pp. 1602- 1608, 1991.

[45] S. Kumar, D. Manocha, W. Garrett, and M. Lin, "Hierarchical Backface Computation," *Proc. of 7th Eurographics Workshop on Rendering*, pp. 231-240, 1996.

[46] L. Shirman. and S. Abi-Ezzi, "The Cone of Normals Technique for Fast Processing of Curved Patches," *EUROGRAPHICS'93*, vol. 12, no. 3., pp. 261-272, Aug. 1993.

[47] D. Luebke and E. Erickson, "View-Dependent Simplification of Arbitrary Polygonal Environments," *Computer Graphics Proceedings, SIGGRAPH 1997*, pp. 199-208, 1997.

[48] P. Sander, X. Gu, S. Gortler, H. Hoppe, and J. Snyder, "Silhouette Clipping," *Computer Graphics Proceedings, SIGGRAPH 2000,* pp. 327-334, 2000.

[49] W. McNeely, K. Puterbaugh, and J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling," *Computer Graphics Proceedings, SIGGRAPH 1999*, pages 401–408, 1999.

[50] J.F. Canny, "The Complexity of Robot Motion Planning," ACM Doctoral Dissertation Award. MIT Press, 1988.

[51] R. Whitaker and D. Breen, "Level-Set Models for the Deformation of Solid Objects," *Proceedings of the 3rd International Workshop on Implicit Surfaces*, pp. 19-35, 1998.

[52] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, May 1996.

[53] M. Foskey, M. Garber, M. Lin, and D. Manocha, "A Voronoi-Based Hybrid Motion Planner," *Proc. IEEE/RSJ International Conf. on Intelligent Robots and Systems*, pp. 89-97, 2001.

[54] J. Hollerbach, "Haptic Interfacing for Virtual Prototyping of Mechanical CAD Designs," *ASME Design for Manufacturing Symposium*, pp. 201-207, 1997.

[55] D. Nelson and E. Cohen, "Optimization-Based Virtual Surface Contact Manipulation at Force Control Rates," *IEEE Virtual Reality 2000*, pp. 37-44, 2000.