CURVE AND SURFACE FRAMING FOR SCIENTIFIC VISUALIZATION AND DOMAIN DEPENDENT NAVIGATION

by

Hui Ma

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Department of Computer Science

Indiana University

February 3, 1996

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

Dr. Andrew J. Hanson (Principal Advisor)

Dr. Dennis B. Gannon

Dr. Randell Bramley

Dr. David Wise

Dr. Jiping Sha

February 3, 1996

To Lijuan and my mother

Acknowledgments

This dissertation would not be possible without the continuous support and encouragement of my advisor Andrew Hanson. His enthusiasm, broad range of interests and dedication to excellence have had an immense impact on me. He cares for his students and he is the best person I have ever met to discuss questions with. I have learned more things from him than he will ever know.

Dennis Gannon also has had a profound influence on me. He not only introduced me to parallel and distributed computing, but his leadership and his enthusiasm for keeping up with the latest developments have inspired me and others around him to set high personal goals and to go after them.

Randall Bramley introduced me to scientific computing. I enjoyed talking with him about graduate student life both within computer science departments and mathematics departments. He also gave me good advice on how to prepare for both the oral exam and the final defense.

I would like to thank David Wise for being on my committee and voicing his opinions about this research from a different point of view.

Jiping Sha is always there when I have mathematical questions. His clear explanation of mathematics and good judgment about new ideas helped enormously during

this research.

I would like to thank Chuck Livingston for introducing me to the wonderful world of topology, and Peter Shirley for teaching me computer graphics. I'll benefit from the knowledge I have learned from them for the rest of my life.

Thanks are also due to Marc Najork for a very fruitful summer internship at DEC SRC. I enjoyed working with him and learned a lot from him and the rest of the SRCers.

Suresh Srinivas and Jake Gotwals are my office mates and best friends. I really enjoyed our countless debates on various topics. I learned more from them than from any courses. Jake also helped me with presentations at conferences and the writing of this dissertation.

I would like to thank many graduate students in this department and many friends in the IU Friendship Association of Chinese Students and Scholars for making my student life at IU very enjoyable. Special thanks to Phil Bradford, Chun-Perng Cheah, Chih-Yi Chen, Jun He, Manoj Jain, Tom Loos, Sudhir Rao, Raja Sooriamurthi, Shankar Swamy, Esen Tuna and Jun Yuan.

I would like to thank our fantastic departmental and system staff for making my stay in this department very comfortable. Sincere thanks to Nancy Garrett, Dee Heifner, Rob Henderson, Steve Kinzler, Pam Larson, Nat McKamey and Bruce Shei.

Finally, I would like to thank my wife, Lijuan, for her love and for sharing life with me during the past several years. Those busy years will become wonderful memory for us in the future. Also I would like to thank my Mom for her encouragement and support.

Hui Ma

Curve and Surface Framing for Scientific Visualization
and Domain Dependent Navigation

Curves and surfaces are two of the most fundamental types of objects in computer graphics. Most existing systems use only the 3D positions of the curves and surfaces, and the 3D normal directions of the surfaces, in the visualization process. In this dissertation, we attach moving coordinate frames to curves and surfaces, and explore several applications of these frames in computer graphics and scientific visualization.

Curves in space are difficult to perceive and analyze, especially when they are densely clustered, as is typical in computational fluid dynamics and volume deformation applications. Coordinate frames are useful for exposing the similarities and differences between curves. They are also useful for constructing ribbons, tubes and smooth camera orientations along curves.

In many 3D systems, users interactively move the camera around the objects with a mouse or other device. But all the camera control is done independently of the properties of the objects being viewed, as if the user is flying freely in space. This type of domain-independent navigation is frequently inappropriate in visualization applications and is sometimes quite difficult for the user to control. Another productive approach is to look at domain-specific constraints and thus to create a new class of navigation strategies. Based on attached frames on surfaces, we can constrain the camera gaze direction to be always parallel (or at a fixed angle) to the surface normal. Then users will get a feeling of driving on the object instead of flying through the space. The user's mental model of the environment being visualized can be greatly enhanced by the use of these constraints in the interactive interface.

Many of our research ideas have been implemented in *MeshView*, an interactive system for viewing and manipulating geometric objects. It contains a general purpose C++ library for nD geometry and supports a winged-edge based data structure. Dozens of examples of scientifically interesting surfaces have been constructed and included with the system.

Contents

List of Figures

Chapter 1

Introduction

Curves and surfaces are two of the most fundamental types of objects in computer graphics. In this dissertation we attach moving coordinate frames to curves and surfaces, and explore several applications of such frames in computer graphics and scientific visualization.

Most of today's computer graphics applications use only the 3D positions of the curves and surfaces and the 3D normal directions of the surfaces in the visualization process. By studying the properties of the coordinate frames determined by the geometry of curves and surfaces, and finding numerical algorithms to generate them, we can attach frames to each point of any curve or surface. We will show that one can use these frames to develop original tools to visualize the properties of curves and surfaces, to construct new objects associated with the curves and surfaces, and to control the camera orientation when we navigate through a scene of curves and surfaces.

We will explore three major applications of coordinates frames:

Generating ribbons, tubes and smooth camera orientations along a curve. After we attach coordinate frames to the points of a curve, it is straightforward to construct ribbons, tubes and forward-facing camera orientations along the curve. These will have certain properties related to the framing method we choose. For example, we may want to minimize camera rotation so the user will be less apt to experience motion sickness, or we may want some extra twist which is constrained to return to the initial orientation after the camera travels once around a closed curve.

Visualization of dense sets of curves. Curves in space are difficult to perceive and analyze, especially when they form dense clusters, as is typical in computational fluid dynamics (CFD) and volume deformation applications. Coordinate frames are useful for exposing the differences between curves, even when the curves appear almost identical. Powerful curve analysis tools can thus be built based on these frames.

Domain dependent navigation of curves and surfaces. In many 3D systems, users interactively move the camera around the objects using a mouse, a joy-stick or other device. Users can translate and rotate the camera around the objects, or change various camera parameters, like focal length, near and far clipping planes, field of view, etc. However, all the camera control is done independently of the properties of the objects being viewed. This kind of camera control is based on the assumption that the users are flying freely in space. This assumption of domain-independent navigation is frequently inappropriate in visualization applications and is sometimes quite difficult for the user to control.

Another productive approach is to look at domain-specific constraints and thus to create a new class of navigation strategies. A domain-specific movement may depend

on the particular local or global geometric properties of the object being viewed. We can use frames attached to the surfaces to constrain the camera gaze direction to always be parallel to (or at a fixed angle to) the surface normal. Then the user will get a feeling of driving on the surface instead of flying through the space. The local properties of the surface, e.g., the curvatures, will directly affect the navigation. The user's mental model of the environment being visualized can be greatly enhanced by the use of these constraints in the interactive interface.

1.1 Contributions of the Dissertation

The primary contributions of this dissertation are an investigation of the mathematical foundations of framing methods, and a family of novel ways of exploiting coordinate frames for applications in scientific visualization and computer graphics. A secondary contribution is the *MeshView* visualization system, an interactive system for viewing and manipulating geometric objects, especially 4D objects such as quaternion-valued coordinate frames. Many of our research ideas have been implemented in this system, which contains a general purpose C++ library for nD geometry and supports a new object description format – *WING*. Dozens of examples of scientifically interesting surfaces have been constructed and included with the system. Most of our research has been published in refereed articles (see [34, 35, 36, 37]).

We now look at the primary contributions of this dissertation in more detail.

- An algorithm for constructing parallel transport frames along a curve is presented, as well as a mathematical proof of its validity and an analysis of its rate of convergence. We compare the advantages and disadvantages of parallel

transport frames with classical Frenet frames along curves.

- We discuss the issues involved in constructing ribbons, tubes and smooth camera orientations along a curve using parallel transport frames or Frenet frames.

- While coordinate frames are useful for exposing the similarities and differences between curves, it is awkward to represent frames visually in high-density data because a frame consists of three 3D vectors, or nine components. We present a technique that re-expresses the moving frame as a unit quaternion, a point in 4D. We discuss the issues involved in assigning Frenet quaternion frames and parallel-transport quaternion frames to curves.

- We present several methods for visualizing properties of the four-dimensional quaternion fields. We also examine an interactive technique for simultaneously displaying the original 3D curves and the corresponding 4D quaternion curves, and for visualizing the interaction between these two spaces.

- An interactive method for exploring topological spaces based on the natural local geometry of the space is presented. The method is applicable to any-dimensional manifolds in any-dimensional ambient space. In our approach, a controller is used to choose a direction in which to "walk" a manifold along a local *geodesic* path (see Appendix D for a brief introduction to mathematical theories of tangent spaces and geodesics). The method automatically generates orientation changes that produce a maximal viewable region with each step of the walk. The proposed interaction framework has many natural properties that help the user develop a useful cognitive map of a space. Also, the framework

is well-suited to haptic interfaces that can be incorporated into desktop virtual reality systems.

1.2 Related Work

The classical theory of Frenet frames can be found in many differential geometry textbooks, e.g., [19, 23, 59]. The parallel transport frame was described in Bishop [10].

Shoemake in [56] introduced quaternions into the computer graphics field. He showed how to use quaternions to interpolate (linearly or by using splines) between two or more 3D frames. See also Kim et al [44], Nielson [48], Schlag [52] and Shoemake [57].

Orientation spaces and their relationship to quaternions are described in Altmann [2]. An interesting approach to the visualization of the properties of quaternions was given by Hart, Francis, and Kauffman [40]. Systematic approaches for representing clusters of orientations in 3D spaces of angles have been suggested, for example, by Alpern et al. [1]. Gray in [17, 23] exploits the curvature and torsion scalar fields on a curve for visualization purposes.

Hanson and his students have done extensive work in visualization of high dimensional spaces, especially in 4D, see [26, 27, 29, 30, 31, 32, 33]. Several papers (Banchoff[3], Banchoff[4], Banks[5]) show pictures of computer generated 4D surfaces. In [6], Banks introduces some refinements of previous techniques on nD diffuse and specular illumination. Geomview ([49]) has several external modules that do high dimensional visualization. The *4Dview* module reads 4D objects and every time the

user moves the mouse, it creates a 3D object and sends it into Geomview. It has some unique features, including 4D perspective projection and 4D slicing. But its interactive 4D control is slow and unwieldy. Another external module of Geomview is *Maniview* ([24]). Maniview can display 3-manifolds using discrete groups. Users see mirror-images of the fundamental domain of the 3-manifold. The *NDView* module of Geomview provides yet another approach, with displays consisting of selected 3D subspaces of an nD object.

Several algorithms for finding minimal distances on polyhedral surfaces have been described ([47, 53, 55, 65]). Bryson in [14] presents a system for visualizing geodesics in gravitational spacetime that allows user control of the displayed paths in a virtual reality environment.

Regge in [51] extends many concepts in smooth Riemannian geometry to a flat polyhedral framework. This formulation of geometry is closely related to the methods we use to navigate tessellated geometric models in computer graphics. See Appendix E for a brief introduction to the *Regge Calculus*.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows.

In Chapter 2, we introduce the differential geometry of 3D curves. We compare the mathematical properties of the Frenet frame and the parallel transport frame. A numerical algorithm for calculating parallel transport frames is presented. We describe applications to the generation of ribbons, tubes and smooth camera orientations along a curve.

In Chapter 3, we present a technique that re-expresses a frame as a unit quaternion, which reduces the number of components from nine with six constraints to four with one constraint. We study the theories of both the quaternion Frenet frame and the quaternion parallel transport frame. We describe several flexible approaches for interacting with and exploiting the properties of the resulting four-dimensional quaternion fields. As examples, we examine a torus knot, a spherical volume deformation known as the Dirac string trick ([40]), a twisted volume used in topology to construct knots, and streamlines of 3D vector flow fields.

In Chapter 4, as an example of the general domain dependent navigation methodology, we propose an interactive method for exploring topological spaces based on the natural local geometry of the space. We present an algorithm to walk along a curve, a surface or a higher dimensional object. The algorithm automatically generates orientation changes that produce a maximal viewable region with each step of the walk.

In Chapter 5, we present an overview of our MeshView visualization system. We describe the features of MeshView, the 3D/4D rolling ball interface, a C++ library of nD geometry and a new winged-edge based data structure – WING.

Finally, Chapter 6 presents a summary along with directions for future research.

Chapter 2

Curve Framing

In this chapter, we attack the problem of associating moving coordinate frames to three-dimensional space curves in ways that are well-understood mathematically and that have optimal behavior for certain classes of computer graphics applications. These frames can be used for creating ribbons, tubes, and smoothly varying camera orientations that are controlled by the curve geometry itself. They also form the foundation for visualizing densely clustered 3D curves using quaternions, a topic for the next chapter.

Classical differential geometry typically treats moving frames using the Frenet frame formalism because of its close association with a curve's curvature and torsion, which are coordinate-system independent [19, 23, 59]. The Frenet frame, unfortunately, has the property that it is undefined when the curve is even momentarily straight (has vanishing curvature), and exhibits wild swings in orientation around points where the osculating plane's normal has major changes in direction. We propose an alternative approach, the parallel-transport frame method (see Bishop [10]),

which has a mathematically sound foundation and more appropriate behavior for computer graphics. We compare the properties of alternative framing methods and point out when the parallel-transport approach has unique advantages. In cases where the Frenet frame has desirable properties, a hybrid strategy is also feasible.

Typical computer graphics applications of the parallel-transport frame include the generation of ribbons and tubes from 3D space curves, and the generation of forward-facing camera orientations given an appropriate initial camera path. If the curve is coarsely refined, but is smooth enough to generate appropriate frame control points from the parallel-transport frame algorithm, the resulting frames can be used as control points for any desired degree of smooth spline interpolations using the methods of Shoemake [56], Schlag [52], Kim et al [44] and Barr et al [7]. Rotating camera orientations relative to a stable forward-facing frame can be added by various techniques such as that of Shoemake [57].

In Section 2.1, we introduce the basic mathematics of coordinate frames on space curves, emphasizing the parallel-transport frame; the properties of the Frenet and parallel-transport frames are compared. Our algorithms are given in Section 2.2, which also describes how coordinate frames can be applied to the generation of ribbons, tubes, and camera frames. Appendices A, B and C contain proofs and derivations of useful formulas.

2.1 The Differential Geometry of Space Curves

Our first goal is to define moving coordinate frames that are attached to a curve in 3D space. We will assume that the curves are defined in practice by a discrete sequence of

points connected by straight line segments; thus numerical derivatives can be defined at each point.

2.1.1 Frenet Frames

The *Frenet frame* (see, e.g., [19, 23]) is defined as follows: If $\vec{\mathbf{x}}(t)$ [1] is any thrice-differentiable space curve with non-vanishing second derivative, its tangent, binormal, and normal vectors at a point on the curve are given by

$$\begin{aligned} \hat{\mathbf{T}}(t) &= \frac{\vec{\mathbf{x}}'(t)}{\|\vec{\mathbf{x}}'(t)\|} \\ \hat{\mathbf{B}}(t) &= \frac{\vec{\mathbf{x}}'(t) \times \vec{\mathbf{x}}''(t)}{\|\vec{\mathbf{x}}'(t) \times \vec{\mathbf{x}}''(t)\|} \\ \hat{\mathbf{N}}(t) &= \hat{\mathbf{B}}(t) \times \hat{\mathbf{T}}(t) \ . \end{aligned} \tag{2.1}$$

This standard frame configuration is illustrated in Figure 2.1. When the second derivative vanishes on some interval, the Frenet frame is temporarily undefined. Attempts to work around this problem involve various heuristics [54].

The Frenet frame obeys the following differential equation in the parameter t (which is the origin of the requirement for one more order of differentiability beyond the second derivative):

$$\begin{bmatrix} \hat{\mathbf{T}}'(t) \\ \hat{\mathbf{N}}'(t) \\ \hat{\mathbf{B}}'(t) \end{bmatrix} = v(t) \begin{bmatrix} 0 & \kappa(t) & 0 \\ -\kappa(t) & 0 & \tau(t) \\ 0 & -\tau(t) & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}}(t) \\ \hat{\mathbf{N}}(t) \\ \hat{\mathbf{B}}(t) \end{bmatrix} \tag{2.2}$$

[1] In this dissertation, $\vec{\mathbf{x}}$ indicates that $\vec{\mathbf{x}}$ is a vector while $\hat{\mathbf{x}}$ indicates that $\hat{\mathbf{x}}$ is a unit vector.

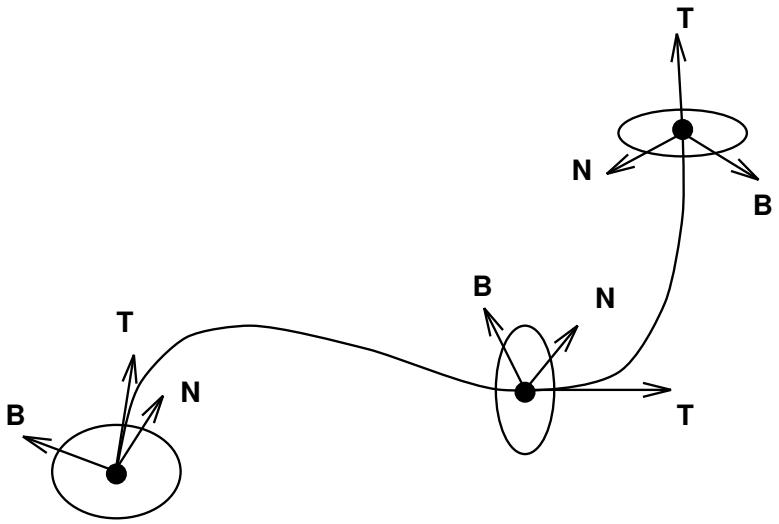


Figure 2.1: The triad of orthogonal axes forming the Frenet frame for a curve with non-vanishing curvature.

where $v(t) = \|\vec{\mathbf{x}}'(t)\|$ is the scalar magnitude of the curve derivative (often re-parameterized to be unity, so that t becomes the arclength s), $\kappa(t)$ is the scalar curvature, and $\tau(t)$ is the torsion. These quantities can in principle be calculated in terms of the parameterized or numerical local values of $\vec{\mathbf{x}}(t)$ and its first three derivatives as follows:

$$\begin{aligned} \kappa(t) &= \frac{\|\vec{\mathbf{x}}'(t) \times \vec{\mathbf{x}}''(t)\|}{\|\vec{\mathbf{x}}'(t))\|^3} \\ \tau(t) &= \frac{(\vec{\mathbf{x}}'(t) \times \vec{\mathbf{x}}''(t)) \cdot \vec{\mathbf{x}}'''(t)}{\|\vec{\mathbf{x}}'(t) \times \vec{\mathbf{x}}''(t)\|^2} \ . \end{aligned}$$

Given a non-vanishing curvature and a torsion as smooth functions of t, one can theoretically integrate the system of equations to find the unique numerical values of the corresponding space curve $\vec{\mathbf{x}}(t)$ (up to a rigid motion).

2.1.2 Parallel Vector Fields

Before going on to introduce parallel-transport frames as an alternative to the Frenet frame, let us spend a moment looking at parallel vector fields on curves in general; such vector fields are typically constructed by parallel transporting vectors along a curve.

For a given space curve $\vec{\mathbf{x}}(s)$ parameterized by arclength s, a vector field $\vec{\mathbf{V}}(s)$ is said to be *normal* if it is everywhere perpendicular to the curve's tangent $\hat{\mathbf{T}}(s) = \vec{\mathbf{x}}'(s)$. A normal vector field $\vec{\mathbf{V}}(s)$ is said to be *parallel* to the curve $\vec{\mathbf{x}}(s)$ if its derivative is tangential along the curve; that is, $\vec{\mathbf{V}}'(s) \parallel \hat{\mathbf{T}}(s)$. Such a vector field turns only as much as is necessary for it to remain normal.

More generally, an arbitrary vector field $\vec{\mathbf{V}}$ along a curve $\vec{\mathbf{x}}$ is said to be *parallel* if its normal component is parallel and its tangential component is a constant multiple of the unit tangent field of $\vec{\mathbf{x}}$.

The curve $\vec{\mathbf{y}} = \vec{\mathbf{x}} + \vec{\mathbf{V}}$ is called a *parallel curve* of $\vec{\mathbf{x}}$, if $\vec{\mathbf{V}}$ is parallel to $\vec{\mathbf{x}}$.

Properties of Parallel Vector Fields. A curve $\vec{\mathbf{x}}$, a parallel normal vector field $\vec{\mathbf{V}}$, and the corresponding parallel curve $\vec{\mathbf{y}}$ have the following key properties:

(a) $\vec{\mathbf{V}}$ has constant length.

(b) $\vec{\mathbf{V}}$ is perpendicular to both $\vec{\mathbf{x}}'$ and $\vec{\mathbf{y}}'$.

(c) $\vec{\mathbf{V}}$ is locally a segment of minimum length between the two curves if $\|\vec{\mathbf{V}}\|$ is sufficiently small.

(d) For an arbitrary normal vector $\vec{\mathbf{V}}_0$ at a point $\vec{\mathbf{x}}(s_0)$, there exists a *unique* parallel field $\vec{\mathbf{V}}(s)$ on $\vec{\mathbf{x}}(s)$ such that $\vec{\mathbf{V}}(s_0) = \vec{\mathbf{V}}_0$. A numerical algorithm will be

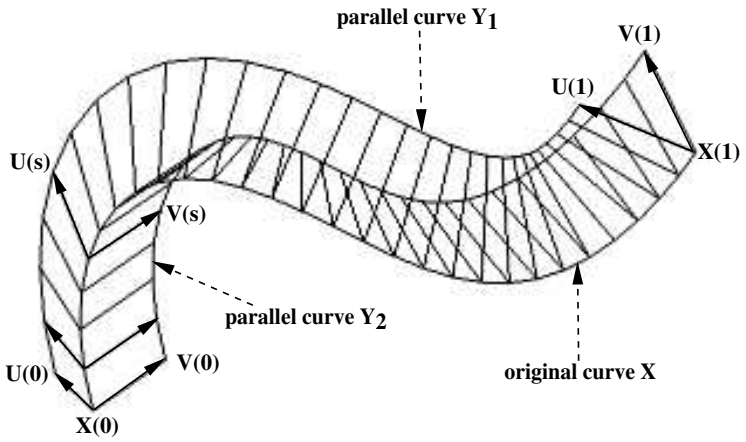


Figure 2.2: Properties of parallel vector fields.

presented in Figure 2.7.

(e) If two arbitrary normal vectors $\vec{\mathbf{V}}_0$ and $\vec{\mathbf{U}}_0$ generate parallel fields $\vec{\mathbf{V}}(s)$ and $\vec{\mathbf{U}}(s)$ respectively, the angle between $\vec{\mathbf{V}}(s)$ and $\vec{\mathbf{U}}(s)$ is constant along the curve. That is $\vec{\mathbf{V}}(s) \cdot \vec{\mathbf{U}}(s) = \vec{\mathbf{V}}_0 \cdot \vec{\mathbf{U}}_0$ for all s.

These properties are illustrated in Figure 2.2.

2.1.3 Parallel-Transport Frames

The *parallel-transport (PT) frame* is an alternative approach to defining a moving frame that is well-defined even when the curve has vanishing second derivative. Because of the property (e) of parallel vector fields, we can parallel transport an orthonormal frame along a curve simply by parallel transporting each component of the frame.

The PT frame is based on the observation that, while $\hat{\mathbf{T}}(t)$ for a given curve

model is unique, we may choose any convenient arbitrary basis $(\hat{\mathbf{N}}_1(t), \hat{\mathbf{N}}_2(t))$ for the remainder of the frame, so long as it is in the normal plane perpendicular to $\hat{\mathbf{T}}(t)$ at each point. If the derivatives of $(\hat{\mathbf{N}}_1(t), \hat{\mathbf{N}}_2(t))$ depend only on $\hat{\mathbf{T}}(t)$ and not each other, we can make $\hat{\mathbf{N}}_1(t)$ and $\hat{\mathbf{N}}_2(t)$ vary smoothly throughout the path regardless of the curvature. We therefore have the alternative frame equations

$$\begin{bmatrix} \hat{\mathbf{T}}' \\ \hat{\mathbf{N}}_1' \\ \hat{\mathbf{N}}_2' \end{bmatrix} = v \begin{bmatrix} 0 & k_1 & k_2 \\ -k_1 & 0 & 0 \\ -k_2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}} \\ \hat{\mathbf{N}}_1 \\ \hat{\mathbf{N}}_2 \end{bmatrix} . \tag{2.3}$$

One can show (see, e.g, Bishop [10]) that

$$\begin{aligned} \kappa(t) &= \left((k_1)^2 + (k_2)^2\right)^{1/2} \\ \theta(t) &= \arctan\left(\frac{k_2}{k_1}\right) \\ \tau(t) &= -\frac{d\theta(t)}{dt} , \end{aligned}$$

so that k_1 and k_2 effectively correspond to a Cartesian coordinate system for the polar coordinates κ, θ with $\theta = -\int \tau(t)\, dt$. The orientation of the PT frame includes the arbitrary choice of integration constant θ_0, which disappears from τ (and hence from the Frenet frame) due to the differentiation.

As with the Frenet equations, we can begin with a pair of functions $(k_1(t), k_2(t))$ and an initial frame, and then integrate any alternate form of the frame equations to find the curve $\vec{\mathbf{x}}(t)$ up to a rigid motion.

These equations also give an abstract construction for a PT frame. In Section 2.2, we will present a much simpler numerical method. In Appendix C, we give a

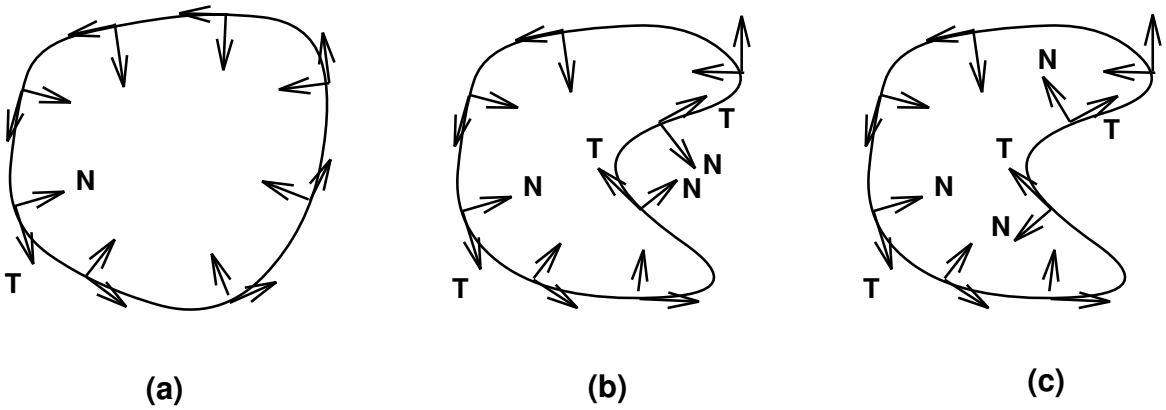


Figure 2.3: Comparing the Frenet and PT frames on plane curves. (a) Convex curve; both frames are identical, with the third component of the frame pointing out of the paper. (b) The Frenet frame on a non-convex curve (with inflection points) reverses the direction of the normal fields at each inflection point, so the direction of $\hat{\mathbf{B}}$ is into the paper on the indented portion of the curve. (c) A PT frame on the same non-convex curve maintains continuity in the direction of the third component of the frame throughout the curve.

proof showing that the frame field generated by our algorithm correctly approximates the PT frame of an underlying smooth curve as the curve segment length approaches zero.

2.1.4 Comparison of Frenet and Parallel-Transport Frames

The contrast between the properties of the Frenet frame and the PT frame is best seen by looking at some examples.

Figure 2.3a shows the frames for a convex plane curve; the Frenet and PT frames are identical. However, as soon as the curve has inflection points in the plane, as shown in Figure 2.3b, one sees that the Frenet frame's normal components instantly switch sign at each inflection point, while the PT frame has no such discontinuities; if the curvature remains zero along a straight line segment, the Frenet frame provides no

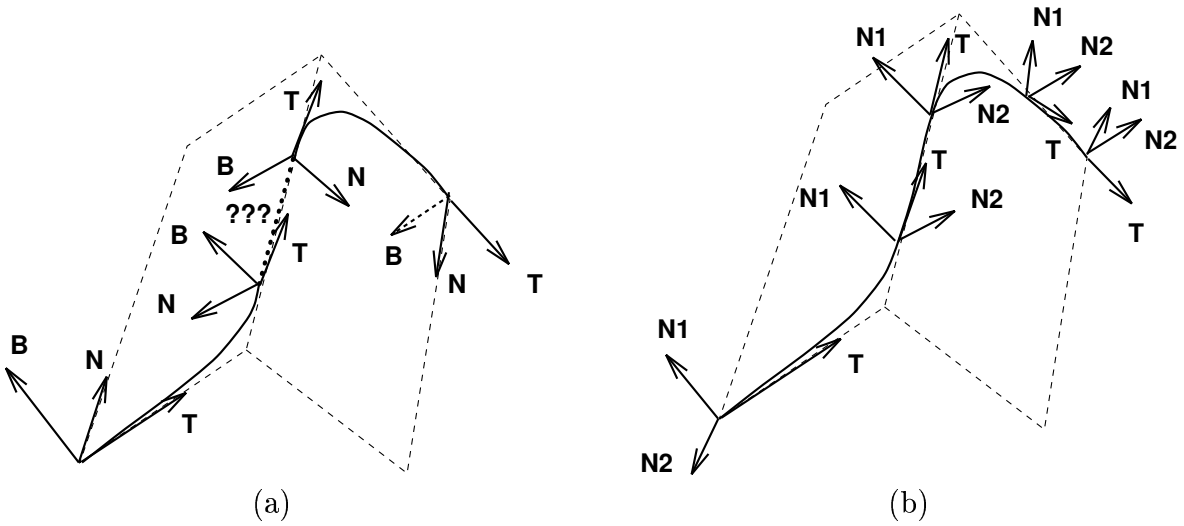
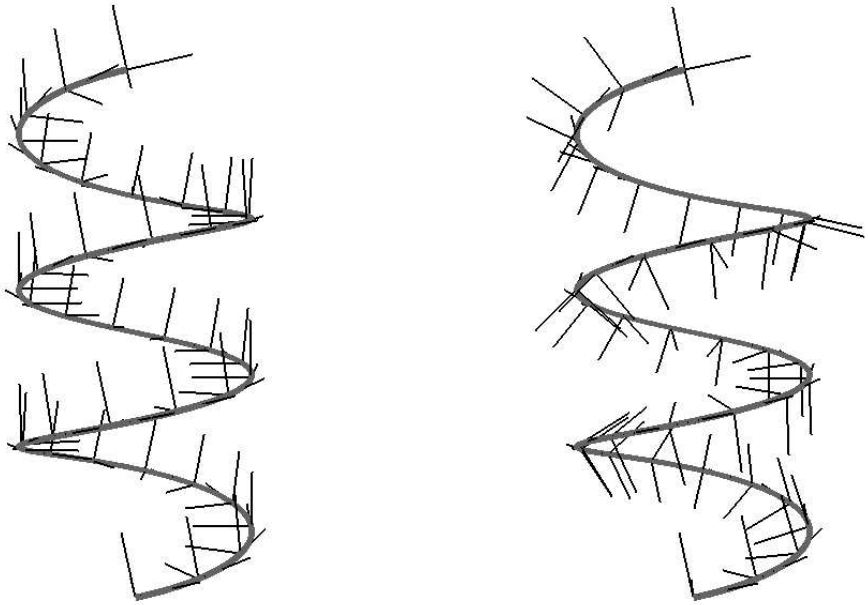


Figure 2.4: Comparing the Frenet and PT frames on a "roof-top." (a) The Frenet frame becomes undefined on the straight line at the peak, then changes abruptly as the curve descends the right side. (b) The PT frame is smooth throughout. (Note: the initial orientation of the normal plane is arbitrary.)

prescription for defining a smooth transition from the frame coming into the straight segment and the (possibly radically different) frame leaving the straight segment.

Next, we look at a non-planar curve drawn on a "roof-top," which exhibits momentarily vanishing curvature and a radical change in the normal to the osculating circle; again, as illustrated in Figures 2.4 a,b, the PT frame is well-behaved and the Frenet frame is not smooth enough to be used as the basis for a ribbon or tube construction.

Finally, we look at a cylindrical helix, which has the property that the torsion is a constant along the whole curve. In this case the Frenet frame returns to the same orientation each time the helix passes through the same axial line on the tube, as shown in Figure 2.5a. This behavior is in fact desirable in some applications. However, at the same time, it hides an essential property: non-zero torsion means the



(a) (b)

Figure 2.5: (a) The Frenet frame of a 3D helix, which has constant torsion; the frame is identical after each turn. (b) The PT frame on a helix, showing how the torsion produces a constant angular velocity or "spin" about the moving tangent vector. The total amount of spin experienced after each turn depends on the pitch of the helix, which determines the torsion.

PT frame is rotating with a constant angular velocity that washes out the torsion in such a way as to *reduce* the total change in the frame orientation at the end of one circuit! Viewed in terms of the quaternion picture of rotations (see Chapter 3), the path traveled in the space of unit quaternions will be shorter. Figure 2.5b illustrates the way in which the PT frame changes by a constant rotation in the normal plane with each cycle around the cylinder.

We note that, since we can create a closed curve by attaching a planar curve to two points of the helix tangent to a single plane, it is possible for closed curves to have PT frames that do not match up after one full circuit of the curve; we will discuss a

simple correction procedure for this situation below (see Section 2.2.3).

2.2 Algorithms and Implementation Issues

In this section, we present our parallel transport algorithm and compare it with another algorithm (see Section 2.2.2). We will also discuss implementation issues dealing with closed curves, spinning, non-uniform tubing and camera animations.

2.2.1 The Parallel Transport Algorithm

In practice, we never have smooth curves in numerical applications, but only piecewise linear curves that are presumed to be approximations to differentiable curves. We will need to compute the tangents to a curve given by the set of points $\{\vec{\mathbf{x}}_i\}$. Without making any assumptions about the shape of the curve approximated by the points, the best we can do is to compute the tangent using a formula involving neighboring points such as

$$\hat{\mathbf{T}}_i = \frac{\vec{\mathbf{x}}_{i+1} - \vec{\mathbf{x}}_{i-1}}{\left\|\vec{\mathbf{x}}_{i+1} - \vec{\mathbf{x}}_{i-1}\right\|} \ .$$

If we are willing to make some assumptions, such as taking any three neighboring points to represent the arc of a circle, we may compute the tangent to that circle at the middle point; this *osculating circle* [59], whose center lies on the intersection line of the planes perpendicular to each chord is discussed in detail in Appendix B.

If the curve is locally straight, i.e., $\vec{\mathbf{x}}''(t) = 0$ or $\hat{\mathbf{T}}_{i+1} = \hat{\mathbf{T}}_i$, then there is no locally-determinable coordinate frame component in the plane normal to $\hat{\mathbf{T}}$; a non-

INPUT: (1) A list of unit tangent vectors $\{\hat{\mathbf{T}}_i\}$, $i = 0, \ldots, N$;
(2) An initial normal vector $\vec{\mathbf{V}}_0$, $\vec{\mathbf{V}}_0 \perp \hat{\mathbf{T}}_0$.

OUTPUT: A list of parallel-transported normal vectors $\{\vec{\mathbf{V}}_i\}$, $i = 1, \ldots, N$, $\vec{\mathbf{V}}_i \perp \hat{\mathbf{T}}_i$.

ALGORITHM:

```
for i ← 0 to N − 1 step 1
    B⃗ ← T̂_i × T̂_{i+1};
    if ‖B⃗‖ = 0 then
        V⃗_{i+1} ← V⃗_i;
    else
        B̂ ← B⃗/‖B⃗‖;
        θ ← arccos(T̂_i · T̂_{i+1});      // 0 ≤ θ ≤ π
        V⃗_{i+1} ← R(B̂, θ) * V⃗_i;
        // Rotate by θ about B̂ (see Appendix A for R)
    end if
end for
```

Figure 2.6: Parallel-transport algorithm

local definition must be used to decide on the remainder of the frame once $\hat{\mathbf{T}}$ is determined, and this is what the PT algorithm provides.

The basic steps of the PT algorithm for piecewise linear curves can be formulated as in Figure 2.6 and the diagram in Figure 2.7. Notice that the input to the algorithm is a list of tangent vectors. The tangent vectors can be computed either numerically as mentioned above, or analytically if the mathematical model for the curve is known.

Remarks.

- If $\hat{\mathbf{T}}_i$ is nearly parallel to $\hat{\mathbf{T}}_{i+1}$, then θ is close to zero, so the rotation matrix $\mathbf{R}$ is close to the identity, making the value of $\hat{\mathbf{B}}$ irrelevant.

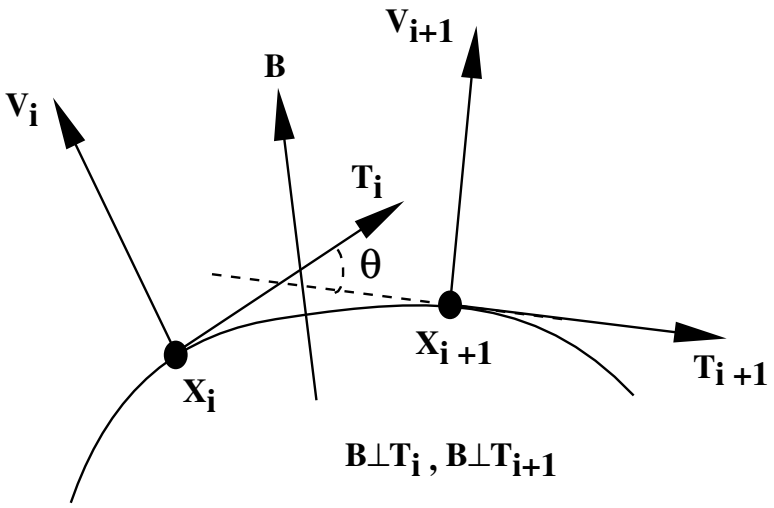


Figure 2.7: Diagram illustrating the geometric quantities used in the parallel transport algorithm.

- Since all actions are rotations, the length of any transported vector is preserved automatically. In fact, the input of the algorithm is not restricted to normal vectors; the method correctly parallel transports any vector. What happens, in effect, is that the normal component of the vector is parallel transported, while the tangential component is repeatedly rotated to coincide in direction with the current tangent vector.

- Given an initial frame with one vector in the direction of the curve tangent and two normal vectors, we can thus construct the PT frame field by applying the algorithm to each of the normal components separately. Orthonormality is automatically preserved due to property (e) in Section 2.1.2. Alternatively, since the third component of the triad is dependent on the other two, we may parallel transport only one component and compute the second by taking the cross-product of the first with the tangent vector.

- The long chain of matrix multiplications may incur some numerical error. This error can be reduced by representing the initial frame as a quaternion (see, e.g., [56]). Since each rotation $\mathbf{R}_i$ can be expressed directly in terms of a quaternion using the same parameters $q = (\cos\frac{\theta}{2}, \hat{\mathbf{B}}\sin\frac{\theta}{2})$, one can carry out the numerical computation of the frame change over the entire curve by quaternion multiplication instead of 3×3 matrix multiplication.

- One can show that as the tessellation of the curve becomes finer and finer, the resulting vectors $\{\vec{\mathbf{V}}_i\}$ approximate the smooth parallel vector field defined in Section 2.1.2. See Appendix C.

2.2.2 The Projection Method

A common method for the generation of tubes is what might be called the "projection method." In this approach, one takes the current unit normal vector $\hat{\mathbf{V}}_i$, the current unit tangent vector $\hat{\mathbf{T}}_i$, and the next unit tangent vector $\hat{\mathbf{T}}_{i+1}$, and computes $\hat{\mathbf{V}}_{i+1}$ as follows:

$$\hat{\mathbf{V}}_{i+1} = \frac{\hat{\mathbf{V}}_i - \hat{\mathbf{T}}_{i+1}(\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_{i+1})}{\|\hat{\mathbf{V}}_i - \hat{\mathbf{T}}_{i+1}(\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_{i+1})\|} \ . \tag{2.4}$$

Compared with the parallel transport method, the projection method has several drawbacks:

- It requires renormalization at every step, while parallel transport explicitly preserves the length of the vector.

- If $\hat{\mathbf{T}}_i \perp \hat{\mathbf{T}}_{i+1}$ and $\hat{\mathbf{V}}_i = \hat{\mathbf{T}}_{i+1}$, we get a zero vector.

- Because the value of $\hat{\mathbf{V}}_{i+1}$ doesn't change if we replace $\hat{\mathbf{T}}_{i+1}$ with $-\hat{\mathbf{T}}_{i+1}$ in

Eq. 2.4, it behaves undesirably when the angle θ between $\hat{\mathbf{T}}_i$ and $\hat{\mathbf{T}}_{i+1}$ is $> \pi/2$. It gives the same result as if $\theta = \pi - \theta$. The parallel transport algorithm can naturally handle any $\theta \in [0, \pi]$.

- One can prove that as the tessellation of a smooth curve becomes finer and finer, the results of the projection method also approach the smooth parallel vector field. But our parallel transport method (Figure 2.6) converges faster. See Appendix C.3.

Therefore, we are going to use the parallel transport algorithm throughout this dissertation.

2.2.3 Closed Curves and Spinning

For closed curves, we let $\vec{\mathbf{x}}_{N+1} = \vec{\mathbf{x}}_0$. The Frenet frame, which is defined only by local curve properties, returns to its initial value. In contrast, the parallel-transport frame will in general *not* return to its initial orientation. The angular difference α between the initial and final frames is determined by the torsion,

$$\alpha = -\oint \tau(s)\, ds \mod (2\pi) \tag{2.5}$$

where s is the arclength.

We can *heuristically* create a continuous frame that is aligned after one trip around the closed curve by adding an additional "spin" around the tangent direction at each vertex. For example, if the curve is described by a set of points $\{\vec{\mathbf{x}}_i\}_{i=0}^{N+1}$, where

$\vec{\mathbf{x}}_{N+1} = \vec{\mathbf{x}}_0$, and the partial curve lengths are

$$L_i = \sum_{j=1}^{i} \|\vec{\mathbf{x}}_j - \vec{\mathbf{x}}_{j-1}\|, \qquad 1 \leq i \leq (N+1) \ ,$$

then for $i = 1, \ldots, N$ we can choose

$$\begin{aligned} \alpha_i &= \alpha \frac{L_i}{L_{N+1}} \\ \vec{\overline{\mathbf{V}}}_i &= \mathbf{R}(\hat{\mathbf{T}}_i, \alpha_i) * \vec{\mathbf{V}}_i \ . \end{aligned}$$

Examples are shown in Color Plate G.1.

2.2.4 Sweeping Tubes and Ribbons

To generate a ribbon or tube, we can use the parallel transport method to create a complete structure by sweeping an initial cross-section composed of vectors $\{\hat{\mathbf{V}}_0^k\}; k = 0, \ldots, K$ through a curve. A thickness function $\{r_i^k\}; i = 0, \ldots, N,\ k = 0, \ldots, K$ can also be applied as

$$\vec{\overline{\mathbf{V}}}_i^k = r_i^k \hat{\mathbf{V}}_i^k \ .$$

Examples are shown in Color Plate G.2.

Smoothing corners. Creating tessellations of ribbons and tubes is complicated by the fact that the outside corners can be handled easily without introducing self-intersections, while the inside corners may have colliding normal line segments if the lengths of the normal segments are longer than the radii of the osculating circles (see Appendix B). For the purposes of this dissertation, we assume that the burden is on

the user to supply a curve that is sufficiently detailed and sufficiently smooth, as well as a tube or ribbon cross-section that does not unreasonably cause self-intersections when swept along the curve.

2.2.5 Parallel Transport Camera Frames

In many (though not all) applications of camera animation, it is desirable to have the camera gaze direction pointing forward along a space curve throughout the motion. Typical examples would include a flight looking through the front window of an airplane cockpit, riding on a roller coaster, or sliding down a bannister. Other applications require the camera gaze direction to remain in some fixed skew orientation relative to the camera path, e.g., a passenger looking out an airplane window. All such applications are easily accommodated using the parallel transport mechanism applied to an initial camera orientation. An example is shown in Color Plate G.3. In this figure, we started with a closed curve, the trefoil knot, and a parallel transport frame that did not return to its initial value after one circuit; we then used the method described in Section 2.2.3 to adjust the frame field. The resulting frame is distinct from the Frenet frame for this curve.

Smooth Orientation Changes. If the discrete points on the camera path are not closely spaced, using the discrete rotations of the basic parallel transport algorithm will result in unacceptably jerky camera motion. This is easily corrected by reinterpreting the camera frame fields at each curve vertex as *key frames* for a quaternion interpolation [7, 52, 56]; then the camera orientations will move smoothly throughout the curve, passing through or near the vertex frame fields, depending on the particular

spline chosen.

In addition, certain classes of camera rolls and spins can be handled naturally as well. The simplest such case is the one in which the additional motions take place with respect to the PT frame of the curve. More complex motions can be handled by a variety of interpolation methods (see, e.g., [57]).

2.3 Conclusion

In this chapter, we have introduced a coherent method for generating smoothly moving frames based on the geometry of a space curve. Our principal new tool is the parallel-transport frame, supplemented when appropriate by the classical Frenet frame. The tools we have introduced provide a number of mathematically well-defined options for producing ribbons, tubes, and camera orientation sequences automatically from the intrinsic geometry of a given space curve.

Possible topics for future investigation include the treatment of parallel transport on higher dimensional manifolds such as surfaces and volumes, parallel transport along curves restricted to such manifolds (that is, curves in curved ambient spaces), and a variety of problems having to do with creating mathematically satisfactory smoothed corners for tubes and ribbons derived from piecewise linear curves.

In Chapter 3, we will study the problem of visualizing dense sets of curves. Since the Frenet and PT frames represent the intrinsic properties of the curves, they are very useful for exposing the similarities and differences of the curves.

Chapter 3

Curve Visualization Using Quaternions

Curves in space are difficult to perceive and analyze, especially when they form dense sets as in typical 3D flow and volume deformation applications. In the previous chapter, we discussed methods that expose essential properties of space curves by attaching an appropriate moving coordinate frame to each point. This chapter presents techniques that re-express that moving frame as a unit quaternion, and support user interaction with the resulting quaternion field. The original curves in 3-space are associated with piecewise continuous 4-vector quaternion fields, which map into new curves lying in the unit 3-sphere in 4-space. Since 4-space clusters of curves with similar moving frames occur independently of the curves' original proximity in 3-space, a powerful analysis tool results. We discuss two separate moving-frame formalisms, the Frenet frame (introduced in Section 2.1.1) and the parallel-transport (PT) frame (introduced in Section 2.1.3). We describe several flexible approaches for interacting

with and exploiting the properties of the four-dimensional quaternion fields.

Our fundamental idea is that quaternion frame coordinates are useful for exposing the similarities and differences of sets of streamlines. Good analytic and visual measures for revealing similarities of curve shapes are rare. Because of the existence of a uniform distance measure in the quaternion space that we use, orientation similarities in the evolution of flow fields appear automatically in meaningful spatial groups. Identification of these similarities is useful for applications such as finding repeating patterns and related curve shapes, both on single curves and within large collections of curves. Conversely, if a large set of nearly-identical curves contains a small number of significant curves that differ from their neighbors due to subtle changes in their frame orientations, our method will distinguish them.

Orientation spaces and their relationship to quaternions are described in Altmann [2]; an interesting approach to the visualization of the properties of quaternions was given by Hart, Francis, and Kauffman [40]. Systematic approaches for representing clusters of orientations in 3D spaces of angles have been suggested, for example, by Alpern et al. [1]. Among previous approaches to visualizing the geometry of space curves, we note the work of Gray [17, 23], which exploits the curvature and torsion scalar fields on a curve for visualization purposes; this method extends naturally to higher-dimensional manifolds with well-defined local curvature. We will give some examples of the application of curvature and torsion fields for completeness here, but will not pursue this approach in detail.

The rest of this chapter is organized as follows. Section 3.1 introduces the mathematical properties of quaternions and how they are related to 3D coordinate frames. The procedures for assigning smooth quaternions to points on space curves are dis-

cussed in Section 3.2. Some typical examples of streamline data are presented in Section 3.3. Section 3.4 presents various visualization methods for the exploitation of the quaternions. A variety of interactive techniques for exploring quaternion fields are described in Section 3.5. Finally, we conclude this chapter in Section 3.6.

3.1 Theory of Quaternions

It is awkward to represent moving frames visually in high-density data because a frame consists of three 3D vectors, or nine components, yet it has only three independent degrees of freedom. Some approaches to representing these degrees of freedom in a three-dimensional space were suggested by Alpern et al. [1]. We propose instead to systematically exploit the representation of 3D orientation frames in four dimensions using equivalent unit quaternions that correspond, in turn, to points on the three-sphere (see, e.g., [56]). A collection of oriented frames such as those of a crystal lattice can thus be represented by mapping their orientations to a point set in the 4D quaternion space. The moving frame of a 3D space curve can be transformed into a path in quaternion space corresponding pointwise to the 3D space curve.

The quaternion representation of rotations re-expressing a moving frame of a 3D space curve is an elegant unit four-vector field over the curve; the resulting quaternion frames can be displayed as curves in their own right, or can be used in combination with other methods to enrich the display of each 3D curve, e.g., by assigning a coded display color representing a quaternion component.

3.1.1 Properties of Quaternions

A quaternion is a four-vector $q = (q_0, q_1, q_2, q_3) = (q_0, \vec{q})$ characterized by the following properties:

Unit Norm The inner product of two quaternions is defined as

$$q \cdot p = q_0 p_0 + q_1 p_1 + q_2 p_2 + q_3 p_3 \ ,$$

so the components of a unit quaternion obey the constraint

$$q \cdot q = (q_0)^2 + (q_1)^2 + (q_2)^2 + (q_3)^2 = 1 \ ,$$

and therefore lie on S^3, the three-sphere, which is embedded in four-dimensional Euclidean space R^4. In fact, all unit quaternions form a Lie group H, which is isomorphic to the 2×2 complex matrix group SU(2).

Multiplication rule The quaternion product of two quaternions q and p, which is written as $q * p$, takes the form

$$\begin{bmatrix} [q*p]_0 \\ [q*p]_1 \\ [q*p]_2 \\ [q*p]_3 \end{bmatrix} = \begin{bmatrix} q_0 p_0 - q_1 p_1 - q_2 p_2 - q_3 p_3 \\ q_0 p_1 + q_1 p_0 + q_2 p_3 - q_3 p_2 \\ q_0 p_2 + q_2 p_0 + q_3 p_1 - q_1 p_3 \\ q_0 p_3 + q_3 p_0 + q_1 p_2 - q_2 p_1 \end{bmatrix} .$$

This rule is isomorphic to multiplication in the group SU(2), the double covering of the ordinary 3D rotation group SO(3). If two quaternions a and b are transformed by

multiplying them by the same quaternion q, their inner product $a \cdot b$ transforms as

$$(q * a) \cdot (q * b) = (a \cdot b)(q \cdot q)$$

and so is invariant if q is a unit quaternion.

Mapping to 3D rotations Every possible 3D rotation R (a 3×3 orthogonal matrix) can be constructed from either of two related quaternions, $q = (q_0, q_1, q_2, q_3)$ or $-q = (-q_0, -q_1, -q_2, -q_3)$, using the quadratic relationship

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_3q_1 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_3q_1 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.1}$$

When we substitute $q = (\cos\frac{\theta}{2}, \hat{\mathbf{n}} \sin\frac{\theta}{2})$ into Eq. (3.1), where $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$ is a unit 3-vector lying on the 2-sphere S^2, $R(\hat{\mathbf{n}}, \theta)$ becomes the standard matrix for a rotation by θ in the plane perpendicular to $\hat{\mathbf{n}}$ (see Appendix A). The quadratic form ensures that the two distinct unit quaternions q and $-q$ in S^3 correspond to the *same* SO(3) rotation.

Mapping from 3D rotations Conversely, given any 3×3 rotation matrix

$$R = \begin{bmatrix} R_{00} & R_{01} & R_{02} \\ R_{10} & R_{11} & R_{12} \\ R_{20} & R_{21} & R_{22} \end{bmatrix},$$

$w^2 = (1 + R_{00} + R_{11} + R_{22})/4$
if $w^2 > 0$ then
    $q_0 = \sqrt{w^2}$
    $q_1 = (R_{21} - R_{12})/(4q_0)$
    $q_2 = (R_{02} - R_{20})/(4q_0)$
    $q_3 = (R_{10} - R_{01})/(4q_0)$
else
    $q_0 = 0$
    $x^2 = -(R_{11} + R_{22})/2$
    if $x^2 > 0$ then
        $q_1 = \sqrt{x^2}$
        $q_2 = R_{10}/(2q_1)$
        $q_3 = R_{20}/(2q_1)$
    else
        $q_1 = 0$
        $y^2 = (1 + R_{11})/2$
        if $y^2 > 0$ then
            $q_2 = \sqrt{y^2}$
            $q_3 = R_{21}/(2q_2)$
        else
            $q_2 = 0$
            $q_3 = 1$
        end
    end
end

Figure 3.1: Mapping from 3D rotation matrix R to unit quaternion q

we can use Eq. (3.1) to find one of the two corresponding quaternions $q = (q_0, q_1, q_2, q_3)$ (the other one is $-q$). An algorithm to do this is outlined in Figure 3.1.

3.1.2 Quaternion Frenet Frames

All 3D coordinate frames can be expressed in the form of quaternions using Eq. (3.1). If we assume the columns of Eq. (3.1) are the vectors $(\hat{\mathbf{T}}, \hat{\mathbf{N}}, \hat{\mathbf{B}})$, respectively, one can

show from Eq. (2.2) that $[q'(t)]$ takes the form (see [28])

$$\begin{bmatrix} q_0' \\ q_1' \\ q_2' \\ q_3' \end{bmatrix} = \frac{v}{2} \begin{bmatrix} 0 & -\tau & 0 & -\kappa \\ \tau & 0 & \kappa & 0 \\ 0 & -\kappa & 0 & \tau \\ \kappa & 0 & -\tau & 0 \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} . \tag{3.2}$$

This equation has the following key properties:

- The matrix on the right hand side is antisymmetric, so that $q(t) \cdot q'(t) = 0$ by construction. Thus all unit quaternions remain unit quaternions as they evolve by this equation.
- The number of equations has been reduced from nine coupled equations with six orthonormality constraints to four coupled equations incorporating a single constraint that keeps the solution vector confined to the 3-sphere.

We verify that the matrices

$$A = \begin{bmatrix} q_0 & q_1 & -q_2 & -q_3 \\ q_3 & q_2 & q_1 & q_0 \\ -q_2 & q_3 & -q_0 & q_1 \end{bmatrix}$$

$$B = \begin{bmatrix} -q_3 & q_2 & q_1 & -q_0 \\ q_0 & -q_1 & q_2 & -q_3 \\ q_1 & q_0 & q_3 & q_2 \end{bmatrix}$$

$$C = \begin{bmatrix} q_2 & q_3 & q_0 & q_1 \\ -q_1 & -q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}$$

explicitly reproduce Eq. (2.2),

$$\begin{aligned} 2 \cdot [A] \cdot [q'] = \hat{\mathbf{T}}' &= v\kappa\hat{\mathbf{N}} \\ 2 \cdot [B] \cdot [q'] = \hat{\mathbf{N}}' &= -v\kappa\hat{\mathbf{T}} + v\tau\hat{\mathbf{B}} \\ 2 \cdot [C] \cdot [q'] = \hat{\mathbf{B}}' &= -v\tau\hat{\mathbf{N}} \ , \end{aligned}$$

where we have applied Eq. (3.2) to get the right-hand terms.

Just as the Frenet equations may be integrated to generate a unique moving frame with its space curve for non-vanishing $\kappa(t)$, we may integrate the much simpler quaternion equations (3.2).

3.1.3 Quaternion Parallel-Transport Frames

Similarly, a parallel-transport frame system given by Eq. (2.3) with $(\hat{\mathbf{N}}_1, \hat{\mathbf{T}}, \hat{\mathbf{N}}_2)$ (in that order) corresponding to the columns of Eq. (3.1) is completely equivalent to the following parallel-transport quaternion frame equation for $[q'(t)]$:

$$\begin{bmatrix} q'_0 \\ q'_1 \\ q'_2 \\ q'_3 \end{bmatrix} = \frac{v}{2} \begin{bmatrix} 0 & -k_2 & 0 & k_1 \\ k_2 & 0 & -k_1 & 0 \\ 0 & k_1 & 0 & k_2 \\ -k_1 & 0 & -k_2 & 0 \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{3.3}$$

where antisymmetry again guarantees that the quaternions remain constrained to the unit 3-sphere. The correspondence to Eq. (2.3) is verified as follows:

$$\begin{aligned} 2 \cdot [B] \cdot [q'] = \hat{\mathbf{T}}' &= vk_1\hat{\mathbf{N}}_1 + vk_2\hat{\mathbf{N}}_2 \\ 2 \cdot [A] \cdot [q'] = \hat{\mathbf{N}}_1' &= -vk_1\hat{\mathbf{T}} \\ 2 \cdot [C] \cdot [q'] = \hat{\mathbf{N}}_2' &= -vk_2\hat{\mathbf{T}} \ . \end{aligned}$$

3.2 Assigning Smooth Quaternion Frames

Given a particular curve, we are next faced with the task of assigning quaternion values to whatever moving frame sequence we have chosen.

3.2.1 Assigning Quaternions to Frenet Frames

The Frenet frame equations are pathological, for example, when the curve is perfectly straight for some distance or when the curvature vanishes momentarily. Thus, real numerical data for space curves will frequently exhibit behaviors that make the assignment of a smooth Frenet frame difficult, unstable, or impossible. In addition, since any given 3×3 orthogonal matrix corresponds to two quaternions that differ in sign, methods of deriving a quaternion from a Frenet frame are intrinsically ambiguous. Therefore, we prescribe the following procedure for assigning smooth quaternion Frenet frames to points on a space curve:

(a) Select a numerical approach to computing the tangent $\hat{\mathbf{T}}$ at a given curve point $\vec{\mathbf{x}}$; this typically depends on the chosen curve model and the number of points one wishes to sample.

(b) Compute the remaining numerical derivatives at a given point and use those to compute the Frenet frame according to Eq. (2.1). If any critical quantities vanish, tag the frame as undefined (or as needing a heuristic fix).

(c) Check the dot product of the previous binormal $\hat{\mathbf{B}}(t)$ with the current value; if it is near zero, choose a correction procedure to handle this singular point. Among the correction procedures we have considered are (1) simply jump discontinuously to the next frame to indicate the presence of a point with very small curvature; (2) create an interpolating set of points and perform a geodesic interpolation , [56]; or (3) deform the curve slightly before and after the singular point to "ease in" with a gradual rotation of the frame or apply an interpolation heuristic (see, e.g., [54]). Creating a jump in the frame assignment is our default choice, since it does not introduce any new information.

(d) Apply a suitable algorithm such as the one in Figure 3.1 to compute a candidate for the quaternion corresponding to the Frenet frame.

(e) If the 3×3 Frenet frame is smoothly changing, make one last check on the 4D inner product of the quaternion frame with its own previous value: if there is a sign change, choose the opposite sign to keep the quaternion smoothly changing (this will have no effect on the corresponding 3×3 Frenet frame). If this inner product is near zero instead of ± 1, you have detected a radical change in the Frenet frame which should have been noticed in the previous tests.

(f) If the space curves of the data are too coarsely sampled to give the desired smoothness in the quaternion frames, but are still close enough to give consistent qualitative behavior, one may choose to smooth out the intervening frames

using the desired level of recursive slerping [52, 56] to get smoothly splined intermediate quaternion frames.

In Color Plate G.4, we plot an example of a torus knot, a smooth space curve with everywhere nonzero curvature, together with its associated Frenet frames, its quaternion frame values, and the path of its quaternion frame field projected from four-space. Color Plate G.5 plots the same information, but this time for a curve with a discontinuous frame that flips too quickly at a zero-curvature point. This space curve has two planar parts drawn as though on separate pages of a partly-open book and meeting smoothly on the "crack" between pages. We see the obvious jump in the Frenet and quaternion frame graphs at the meeting point; if the two curves are joined by a long straight line, the Frenet frame is ambiguous and is essentially undefined in this segment. Rather than invent an interpolation, we generally prefer to use the parallel transport method described next.

3.2.2 Assigning Quaternions to Parallel Transport Frames

In order to determine the quaternion frames of an individual curve using the parallel transport method, we follow a similar, but distinct, procedure:

(a) Select a numerical approach to assigning a tangent at a given curve point as usual.

(b) Assign an initial reference orientation to each curve in the plane perpendicular to the initial tangent direction. The entire set of frames will be displaced from the origin in quaternion space by the corresponding value of this initial orientation matrix, but the shape of the entire curve will be the same regardless of the

initial choice. This choice is intrinsically ambiguous and application dependent. However, one appealing strategy is to base the initial frame on the first well-defined Frenet frame, and then proceed from there using the parallel-transport frame evolution; this guarantees that identical curves have the same parallel-transport frames.

(c) Compute the angle between successive tangents, and rotate the frame by this angle in the plane of the two tangents to get the next frame value (see the parallel-transport algorithm in Figure 2.6).

(d) If the curve is straight, the algorithm automatically makes no changes.

(e) Compute a candidate quaternion representation for the frame, applying consistency conditions as needed.

Note that the initial reference orientation and all discrete rotations can be represented *directly in terms of quaternions*, and thus quaternion multiplication can be used directly to apply frame rotations. Local consistency is then automatic.

An example is provided in Color Plate G.6, which shows the parallel transport analog of Color Plate G.4 for a torus knot. Color Plate G.7 is the parallel transport analog of the pathological case in Color Plate G.5, but this time the frame is continuous when the curvature vanishes.

3.3 Examples

We next present some typical examples of streamline data represented using the basic geometric properties we have described. Each data set is rendered in the following

alternative modes: (1) as a 3D Euclidean space picture, pseudo-colored by curvature value; (2) as a 3D Euclidean space picture, pseudo-colored by torsion value; (3) as a four-vector quaternion Frenet frame field plotted in the three-sphere; (4) as a four-vector quaternion parallel-transport frame field plotted in the three-sphere.

- **Color Plate G.8.** A complicated set of streamlines derived from twisting a solid elastic Euclidean space as part of the process of tying a topological knot.
- **Color Plate G.9.** The "Dirac string trick" deformation ([40]) of a spherical solid consisting of concentric spheres in Euclidean space.
- **Color Plate G.10.** An AVS-generated streamline data set; the flow is obstructed somewhere in the center, causing sudden jumps of the streamlines in certain regions.

While our focus is specifically on the frames of space curves, we remark that collections of frames of isolated points, frames on stream surfaces [41], and volumetric frame fields could also be represented using a similar mapping into quaternion space. See [38].

3.4 Visualization Methods

Once we have calculated the quaternion frames, the curvature, and the torsion for a point on the curve, we have a family of tensor and scalar quantities that we may exploit to expose the intrinsic properties of a single curve. Furthermore, and probably of greater interest, we also have the ability to make visual comparisons of the similarities and differences among families of neighboring space curves.

The moving frame field of a set of streamlines is potentially a rich source of detailed information about the data. However, the 9-component frame is unsuitable for direct superposition on dense data due to the high clutter resulting when its three orthogonal 3-vectors are displayed; direct use of the frame is only practical at very sparse intervals, which prevents the viewer from grasping important structural details and changes at a glance. Displays based on 3D angular coordinates are potentially useful, but lack metric uniformity [1].

The 4-vector quaternion frame is potentially a more informative and flexible basis for frame visualizations; below, we discuss several alternative approaches to the exploitation of quaternion frames for data consisting of families of smooth curves.

3.4.1 Direct Three-Sphere Plot of Quaternion Frame Fields

We now repeat the crucial observation: *For each 3D space curve, the moving quaternion frames define completely new 4D space curves lying on the unit three-sphere embedded in 4D Euclidean space.*

These curves can have *entirely different geometry* from the original space curve, since distinct points on the curve correspond to distinct orientations. Families of space curves with exactly the same shape will map to the *same* quaternion curve, while curves that fall away from their neighbors will stand out distinctly in the three-sphere plot. Regions of vanishing curvature will show up as discontinuous gaps in the otherwise continuous quaternion Frenet frame field curves, but will be well-behaved in the quaternion parallel transport frame fields. Straight 3D lines will of course map to single points in quaternion space, which may require special attention in the display.

Color Plates G.4d and G.5d present elementary examples of the three-sphere

plot for the Frenet frame, while Color Plates G.6d and G.7d illustrate the parallel-transport frame. Color Plates G.8c,d, G.9c,d and G.10c,d present more realistic examples.

The quaternion frame curves displayed in these plots are 2D projections of two overlaid 3D solid balls corresponding to the "front" and "back" hemispheres of S^3. The 3-sphere is projected from 4D to 3D along the 0-th axis, so the "front" ball has points with $0 \leq q_0 \leq +1$, and the "back" ball has points with $-1 \leq q_0 < 0$. The q_0 values of the frame at each point can be displayed as shades of gray or pseudo-color. In the default view projected along the q_0-axis, points that are projected from 4D to the 3D origin are in fact identity frames, since unit length of q requires $q = (\pm 1, 0, 0, 0)$ at these points. In Color Plate G.11, we show a sequence of views of the same quaternion curves from different 4D viewpoints using parallel projection; Color Plate G.12 shows the additional contrast in structure sizes resulting from a 4D perspective projection.

3.4.2 Scalar Geometric Fields

Gray [17, 23] has advocated the use of curvature and torsion-based color mapping to emphasize the geometric properties of single curves such as the torus knot. Since this information is trivial to obtain simultaneously with the Frenet frame, we also offer the alternative of encoding the curvature and torsion as scalar fields on a volumetric space populated either sparsely or densely with streamlines; examples are shown in Color Plates G.8a,b, G.9a,b and G.10a,b.

3.4.3 Similarity Measures for Quaternion Frames

Quaternion frames carry with them a natural geometry that may be exploited to compute meaningful similarity measures. Rather than use the Euclidean distance in four-dimensional Euclidean space R^4, one may use the magnitude of the four-vector scalar product of unit quaternions

$$d(q,p) = |q \cdot p| = |q_0p_0 + q_1p_1 + q_2p_2 + q_3p_3| \ ,$$

or the corresponding angle,

$$\theta(q,p) = \arccos(d(q,p)) \ ,$$

which is the angular difference between the two 4D unit vectors and a natural measure of great-circle arc-length on S^3. Choosing this as a distance measure results in a quantity that is invariant under 4D rotations, invariant under 3D rotations represented by quaternion multiplication, and is also insensitive to the sign ambiguity in the quaternion representation for a given frame. Thus it may be used as a quantitative measure of the similarity of any two 3D frames. This is a natural way to compare either successive frames on a single streamline or pairs of frames on different streamlines.

3.4.4 Probing Quaternion Frames with 4D Light

We next explore techniques developed in the literature for dealing with 4D objects (see [30], [31] and [33]). When dealing with 4D geometry and lighting, the critical

observation is that 4D light can be used as a probe of geometric structure provided we can find a way (such as thickening curves or surfaces until they become true 3-manifolds) to define a unique 4D normal vector that has a well-defined scalar product with the 4D light; when that objective is achieved, we can interactively employ a moving 4D light and a generalization of the standard illumination equations to produce images that selectively expose new structural details.

Given a quaternion field, we may simply select a 4D unit vector L to represent a "light direction" and employ a standard lighting model such as $I(t) = L \cdot q(t)$ to select individual components of the quaternion fields for display using pseudo-color coding for the intensity.

Color Plate G.13 shows a streamline data set rendered by computing a pseudo-color index at each point using the 4D lighting formula and varying the directions of the 4-vector L.

3.4.5 True 4D Illumination

The quaternion curves in 4D may also be displayed in an entirely different mode by thickening them to form 3-manifolds using the method of Hanson and Heng [31, 33] and replacing $q(t)$ in the 4D lighting formula and its specular analogs by the 4D normal vector for each volume element or vertex. The massive expense of volume rendering the resulting solid tubes comprising the 4D projection to 3D can be avoided by extending the "bear-hair" algorithm to 4D curves [6, 30, 43] and rendering the tubes in the limit of vanishing radius.

3.5 Interactive Interfaces

We next describe a variety of specific interactive techniques that we have examined as tools for exploring quaternion fields.

3.5.1 4D Light Orientation Control

Direct manipulation of 3D orientation using a 2D mouse is typically handled using a rolling ball [25] or virtual sphere [16] method to give the user a feeling of physical control. This philosophy extends well to 4D orientation control (see [18, 29]), giving a practical approach to interacting with the visualization approaches of Sections 3.4.4 and 3.4.5.

A 3D unit vector has only two degrees of freedom, and so is determined by picking a point within a unit circle to determine the direction uniquely up to the sign of its view-direction component. The analogous control system for 4D lighting is based on a similar observation: since the 4D normal vector has only 3 independent degrees of freedom, choosing an *interior point* in a solid sphere determines the vector uniquely up to the sign of its component in the unseen 4th dimension (the "4D view-direction component").

Color Plate G.13 shows an example with a series of snapshots of this interactive interface at work. An additional information display shows the components of the 4D light vector at any particular moment.

3.5.2 4D Viewing and Three-Sphere Projection Control

Actually displaying quaternion field data mapped to the 3-sphere requires us to choose a particular projection from 4D to 3D and a method for displaying the features of the streamlines. In order to expose all possible relevant structures, the user interface must allow the viewer to freely manipulate the 4D projection parameters. This control is easily and inexpensively provided using the 4D rolling ball interface [18, 29]. A special version of our "MeshView" 4D viewing utility (see [46] and Chapter 5) has been adapted to support real-time interaction with quaternion frame structures. Color Plates G.11 and G.12 show snapshots from this interactive interface for 4D rotations using parallel and polar 4D projections, respectively.

The simplest viewing strategy plots wide lines that may be viewed in stereo or using motion parallax. A more expensive viewing strategy requires projecting a line or solid from the 4D quaternion space and reconstructing an ideal tube in real time for each projected streamline. The parallel transport techniques introduced in this paper are in fact extremely relevant to this task, and may be applied to the tubing problem as well (see, e.g., [12, 35]).

3.5.3 3D Rotations of Quaternion Displays

Using the 3D rolling ball interface, we can generate quaternion representations of 3D rotations of the form $q = (\cos\frac{\theta}{2}, \hat{\mathbf{n}}\sin\frac{\theta}{2})$ and transform the entire quaternion display by quaternion multiplication, i.e., by changing each point to $p' = q*p$. This effectively displaces the 3D identity frame in quaternion space from $(1, 0, 0, 0)$ to q. This may be useful when trying to compare curves whose properties differ by a rigid 3D rotation

(a common occurrence in the parallel-transport frame due to the arbitrariness of the initial condition).

Other refinements might include selecting and rotating single streamlines in the quaternion field display to make interactive comparisons with other streamlines differing only by rigid rotations. One might also use automated tools to select rotationally similar structures based on minimizing the 4D scalar product between quaternion field points as a measure of similarity.

3.5.4 Exploiting or Ignoring Double Points

The unique feature of quaternion representations of orientation frames is that they are doubled. If we have a single curve, it does not matter which of the two points in S^3 is chosen as a starting point, since the others follow by continuously integrating small transformations. A collection of points with a uniform orientation as an initial condition similarly will evolve in tandem and normally requires only a single choice to see the pattern.

However, it is possible for a frame to rotate a full 2π radians back to its initial orientation, and be on the opposite side of S^3, or for a collection of streamlines to have a wide range of starting orientations that preclude a locally consistent method for choosing a particular quaternion q over its "neighbor" $-q$. We then have several alternatives:

- Include a reflected copy of every quaternion field in the display. This doubles the data density, but ensures that no two frame fields that are similar will appear diametrically opposite; the metric properties of similar curves will be easy to detect. In addition, 4D rotations will do no damage to the continuity of fields

that are rotated to the outer surface and pass from the northern to the southern hyper-hemisphere. If 4D depth is depicted by a color code, for example, a point that rotates up to the surface of the displayed solid ball will smoothly pass to the surface and then pass back towards the center while its color changes from positive to negative depth coding.

- Keep only one copy, effectively replacing q by $-q$ if it is not in the default viewing hyper-hemisphere. This has the effect that each data point is unique, but that curve frames very near diametrically opposite points on the S^2 surface of the solid ball representing the north hyper-hemisphere will be close in orientation but far away in the projection. In addition, when 4D rotations are applied, curves that reach the S^2 surface of the solid ball will jump to the diametrically opposite surface instead of passing smoothly "around" the edge to the southern hyper-hemisphere.

3.5.5 Reciprocal Similarities and Differences

One of the most interesting properties of the quaternion frame method is the appearance of clusters of similar frame fields in the 3-sphere display. Two reciprocal tools for exploring these properties immediately suggest themselves. In Color Plate G.14, we illustrate the effect of grabbing a cluster of streamlines that are spatially close in 3D space and then highlighting their counterparts in the 4D quaternion field space, thus allowing the separate study of their moving frame properties. This technique distinguishes curves that are similar in 3D space but have drastically different frame characteristics.

Color Plate G.15, in contrast, shows the result of selecting a cluster of curves with similar frame-field properties and then highlighting the original streamlines back in the 3D space display. This method assists in the location of similar curves that could not be easily singled out in the original densely populated spatial display. Future work would include examining a variety of alternative approaches to the design of such tools.

3.6 Conclusion

In this chapter, we have introduced a visualization method for distinguishing characteristic features of streamline-like volume data by assigning to each streamline a quaternion frame field derived from its moving Frenet or parallel-transport frames; curvature and torsion scalar fields may be incorporated as well. The quaternion frame is a four-vector field that is a piecewise smoothly varying map from each original space curve to a new curve in the three-sphere embedded in four-dimensional Euclidean space. This four-vector field can be probed interactively using a variety of techniques, including 4D lighting, 4D view control, and interaction with selected portions of the data in tandem 3D streamline and 4D quaternion field displays.

Chapter 4

Space Walking

In this chapter, we propose an interactive method for exploring topological spaces based on the natural local geometry of the space. Examples of spaces appropriate for this visualization approach occur in abundance in mathematical visualization, surface and volume visualization problems, and scientific applications such as general relativity. Our approach is based on using a controller to choose a direction in which to "walk" a manifold along a local geodesic path. The method automatically generates orientation changes that produce a maximal viewable region with each step of the walk. Properties of surfaces with high complexity due to strong curvature or self-intersections can be explored dynamically and pieced together mentally by the user in a manner that is difficult to achieve with standard graphical representations alone. The proposed interaction framework has many natural properties to help the user develop a useful cognitive map of a space and is well-suited to haptic interfaces that can be incorporated into desktop virtual reality systems.

This work originates philosophically in our attempts to develop increasingly deeper

understanding of the properties of complex geometric objects (for a general overview, see [39]). Examples of other work in this area are the Geomview system and its auxiliary modules [24, 49], and Bryson's system for visualizing geodesics in gravitational spacetime metrics [14]. However, highly tuned interaction with objects represented by multiple coordinate patches in three, four, or more dimensions requires additional special tools: our own MeshView system (see [46] and Chapter 5) was developed precisely to enable enhanced 4D interaction, flexible 4D depth cues, and especially to display the relationship between a point on an abstract parameter mesh and its mapping onto a self-intersecting surface in a computer graphics image. 4D lighting methods can also supply additional cues [33]. Long experience viewing and manipulating manifolds such as projective planes, everting spheres, and Riemann surfaces (see, e.g., [15, 21, 26]) suggests the need for additional intuitive tools for exploring such spaces. The techniques described here comprise another powerful, interaction-based approach for developing mental models of such structures (see, e.g., Tversky [61]).

Geodesic polyhedral paths have been studied (see, e.g., [47, 55, 65]) mostly for surfaces in 3D. Our algorithms presented in this chapter apply naturally to surfaces embedded in 3D or higher dimensional spaces and to 3-manifolds and higher-dimensional manifolds. Furthermore, we generate not only the geodesic paths on manifolds, but also the local coordinate frames along the geodesics.

Section 4.1 describes the techniques in the simplest case: walking a curve. Then Section 4.2 extends the basic concepts of curve walking to surfaces and higher dimensional manifolds. In Section 4.3, we discuss the issues involved in smooth interpolation and present a smooth interpolation method for surfaces in 3D. Interactive interfaces

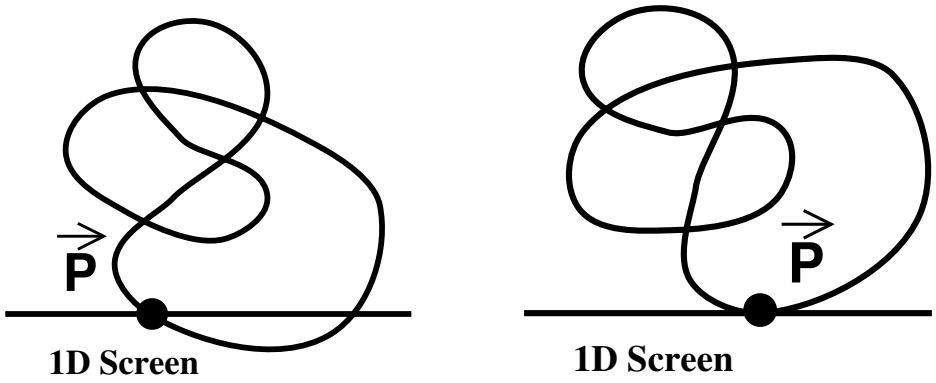


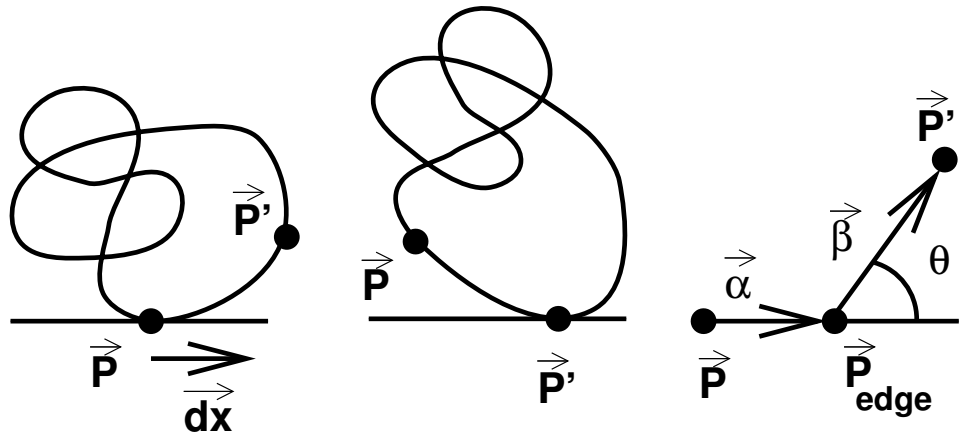
Figure 4.1: (a) A 2D curve with interest point $\vec{\mathbf{P}}$ as it might intersect a 1D display. (b) The curve rotated so the tangent vector at the interest point is aligned with the screen, giving a maximal projection.

are discussed in Section 4.4. Several examples of walking on 3D and 4D surfaces are presented in Section 4.5. We conclude this chapter in Section 4.6.

4.1 Walking a Curve

In this section, we introduce the reader to space walking using the traversal of a space curve, a simple example that exposes the richness of the approach.

Assume that we have a closed 2D space curve along with a particular point of interest $\vec{\mathbf{P}}$ on that curve as shown in Figure 4.1. Now imagine projecting that curve to a single scan-line of a CRT screen. To traverse or "walk" the curve, we must determine both a display strategy that represents the current interest point $\vec{\mathbf{P}}$ in a natural manner, and we must specify how to rigidly transform the full curve to make a transition from $\vec{\mathbf{P}}$ to a neighboring interest point $\vec{\mathbf{P}}'$. In this simple case, the controller can be thought of as a slider, 1D mouse, or 1D joystick.



(a) (b) (c)

Figure 4.2: (a) Controller motion $\vec{\mathbf{dx}}$ initiating transition from interest point $\vec{\mathbf{P}}$ to the next interest point $\vec{\mathbf{P}}'$. (b) Rolling the curve to present a maximal projection at the new interest point. (c) Detail of the geometry involved in the transition using a discrete curve tessellation.

4.1.1 Positioning the Interest Point Using Maximal Projection

As shown in Figure 4.1a, it does not make sense to let the 2D projection of the curve at $\vec{\mathbf{P}}$ have an arbitrary oblique orientation with respect to the screen face; our first assumption, which we will maintain throughout, is that we should always attempt to orient the tangent to the curve *parallel to the screen face* at the interest point $\vec{\mathbf{P}}$, as indicated in Figure 4.1b. We will call this choice the *maximal projection*.

4.1.2 Positioning the Curve

Now let $\vec{\mathbf{dx}}$ be the interactive controller input, which is interpreted to mean "move along the curve in the direction $\vec{\mathbf{dx}}$ by arc length $\|\vec{\mathbf{dx}}\|$." The resulting action changes the interest point from $\vec{\mathbf{P}}$ to $\vec{\mathbf{P}}'$ as we slide along the curve as shown in Figure 4.2a,b.

Suppose for simplicity that the curve is represented as discrete line segments, as shown in Figure 4.2c. If $\vec{\mathbf{P}} + \vec{\mathbf{dx}}$ does not reach the edge vertex $\vec{\mathbf{P}}_{\text{edge}}$, the simplest motion is to translate by $\vec{\mathbf{dx}}$ in the screen tangent direction. Otherwise, we take $\vec{\alpha} = \vec{\mathbf{P}}_{\text{edge}} - \vec{\mathbf{P}}$, $\vec{\beta} = \vec{\mathbf{P}}' - \vec{\mathbf{P}}_{\text{edge}}$, set $\|\vec{\mathbf{dx}}\| = \|\vec{\alpha}\| + \|\vec{\beta}\|$, translate $\vec{\mathbf{P}}_{\text{edge}}$ to the origin, rotate by $\cos\theta = \vec{\alpha}\cdot\vec{\beta}/\|\vec{\alpha}\|\,\|\vec{\beta}\|$, and make a final translation by $\|\vec{\beta}\|$ to place $\vec{\mathbf{P}}'$ at the screen center, where it becomes the new interest point $\vec{\mathbf{P}}$. One obvious strategy for smoothing this transition is to do the rotation in finite time using $\theta(t) = t\theta$ with t increasing from zero to one.

Remark: Self-intersecting curves are treated naturally in this framework by requiring local continuity of the tangent vector. A natural source of interesting self-intersecting curves is the class of non-self-intersecting 3D curves that possess a self-intersecting 2D projection. Accordingly, we next examine 3D space curves, and find that they introduce new interesting issues and intrinsic properties that generalize easily to ND space curves. 3D subspaces turn out to play a special role in walking curves. Later, we will see that 4D subspaces play a similar special role in the analysis of surface walking.

4.1.3 Extension to 3D Space Curves

First suppose we have a 3D space curve, but are still restricted to a 1D display screen, and have chosen a 2D projection. When we restrict the tangent to the interest point of the curve's 2D projection to lie in the 1D screen face, we see that the 3D curve will be tangent to the plane perpendicular to the line of sight through the interest point, but may make an oblique angle to the 1D screen line drawn in this plane. We then have the following choices for positioning the 3D curve:

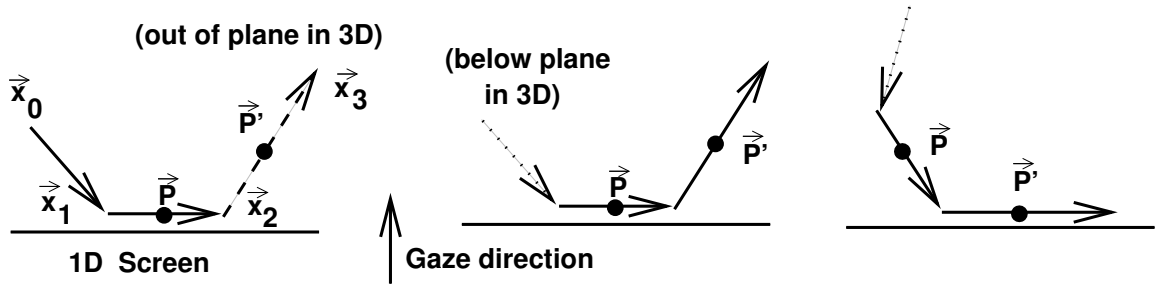
2D tangent Ignore 3D altogether, make the 2D projection tangent to the 1D screen's center at the interest point $\vec{\mathbf{P}}$ and walk the 2D projection only, as in Figure 4.2.

3D tangent, 2D adjustment If the 3D curve's tangent at $\vec{\mathbf{P}}'$ does not lie in the projection plane, first align it with the projection plane as shown in Figure 4.3a,b. Then rotate the result in the projection plane to produce the maximal projection at $\vec{\mathbf{P}}'$ as shown in Figure 4.3c.

3D tangent, 3D adjustment Alternatively, ignore the 2D projection altogether, translate directly from $\vec{\mathbf{P}}$ to $\vec{\mathbf{P}}'$, and rotate the entire 3D curve so that the tangent at $\vec{\mathbf{P}}'$ aligns with the 1D screen line. The only change is that now $\vec{\mathbf{P}}'$ is a 3D vector and the rotation is in the 3D plane defined by the three (assumed non-collinear) points $(\vec{\mathbf{P}}, \vec{\mathbf{P}}_{\text{edge}}, \vec{\mathbf{P}}')$. The angle of rotation corresponds to the 3D dot product, with the geometry as in Figure 4.2c.

At this point, there is an arbitrary rotational degree of freedom about the curve's new tangent vector, the screen line; we have the choice of leaving the orientation where it falls after aligning the tangent, which is quite arbitrary, or choosing a geometrically motivated orientation. Provided the curve's second derivative does not vanish, we can choose an intrinsic orientation based on the Frenet frame (see Eq. 2.1). A natural choice is to force the tangent circle spanned by $\hat{\mathbf{T}}$ and $\hat{\mathbf{N}}$ at $\vec{\mathbf{P}}$ to lie entirely in the 2D projection plane.

Another choice, especially good for curves with long straight sections causing the Frenet frame to be undefined, is the parallel transport frame (see Chapter 2 and [10]). This frame is defined incrementally by integrating along the entire curve from some



(a) (b) (c)

Figure 4.3: One possible approach to maintaining maximal projection of a 3D space curve during the transition between facets. (a) Maximal projection of a tessellated section of a 3D space curve is aligned so that the triangle $(\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2)$ lies in the plane defined by the screen and the gaze direction. (b) Rocking $\vec{\mathbf{P}}'$ into the screen plane by rotating around $\vec{\mathbf{x}}_2 - \vec{\mathbf{x}}_1$. (c) Rotate in screen plane to achieve maximal projection of the segment containing $\vec{\mathbf{P}}'$.

arbitrarily defined starting point and orientation. It has the property that, while it does not have discontinuities when the curve has vanishing derivatives, it also does not provide a natural mechanism for maintaining the locally tangent circle in the projection plane.

4.1.4 Curves in Dimensions Higher Than Three

In higher dimensions, we note the remarkable fact that the local characteristics of the curve that concern us lie within a 3D subspace: the points $(\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \vec{\mathbf{x}}_3)$ defining the tangents to the sequence of interest points as indicated in Figure 4.3 define a tetrahedron, a 3-simplex, regardless of the dimension of the four N-dimensional vectors describing the endpoints of the three edges. Thus we may rephrase the alignment of the Frenet curve normal $\hat{\mathbf{N}}$ with the 2D projection plane for arbitrary dimension as follows: (1) define the plane containing the screen vector and the gaze vector to be aligned with the edges containing $(\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2)$. The desired new orientation has the

tangent at $\vec{\mathbf{P}}'$ parallel to the screen vector; furthermore, we want the plane containing the screen vector and the gaze vector to be aligned with the plane of the edges $(\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \vec{\mathbf{x}}_3)$. Since this transformation is strictly in a 3D subspace, one has several choices: (1) align $\vec{\mathbf{P}}'$ with the screen direction first and then rotate about the line containing $\vec{\mathbf{P}}'$ to place the line containing $\vec{\mathbf{P}}$ back in the projection plane; (2) rotate about the line containing $\vec{\mathbf{P}}$ until the line containing $\vec{\mathbf{P}}'$ is in the projection plane, then rotate in that plane until $\vec{\mathbf{P}}'$ is in the screen tangent direction; (3) better yet, one can perform a geodesic quaternion interpolation between the two frames within the 3D subspace [56].

The strategies that align the 3D tangent instead of the simpler 2D projected tangent have the effect of "pulling the curve through a tube" surrounding the 1D screen line. Also note that in these algorithms the curve may rotate rapidly about the 1D screen line if the maximal projection aligning the curve's Frenet normal with the 2D projection plane containing the gaze vector and the 1D scan line is always enforced.

4.2 Walking in Space

We next outline the extension of the basic concepts of curve walking to surfaces and higher dimensional manifolds. From this point on, we will assume that we have a 2D screen space with a well-defined normal gaze direction, and that surface patches are projected from a 3D graphics space, the "projection volume," onto this 2D screen. Higher dimensional objects will typically first be transformed to the 3D projection volume in a manner analogous to the projection of a 3D curve to a 2D projection plane

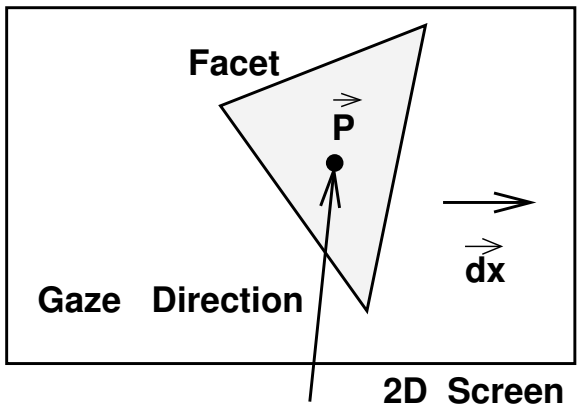


Figure 4.4: Surface facet with interest point $\vec{\mathbf{P}}$ maximally projected onto a 2D screen from an implicit 3D graphics space behind the screen.

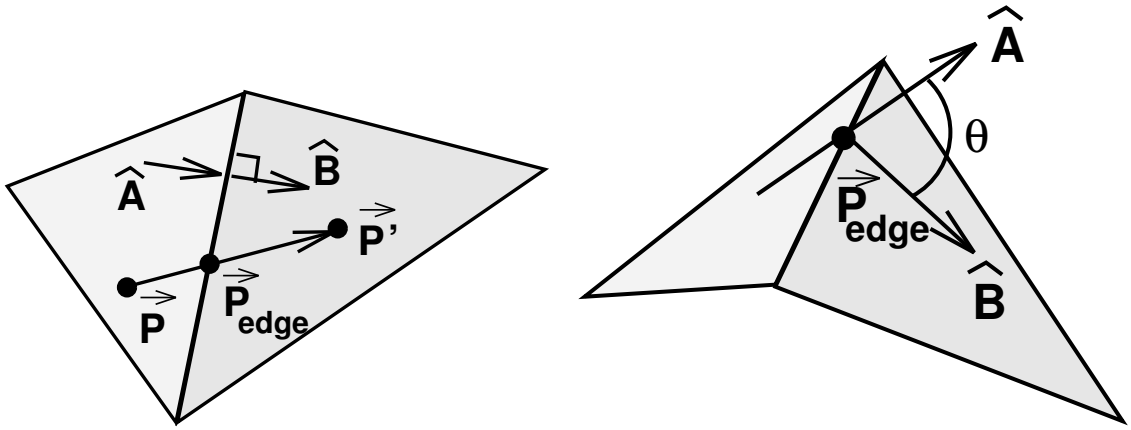
noted earlier. We begin by examining surfaces (2-manifolds) with vertex coordinates in 3D Euclidean space. We assume that a surface is defined by a collection of vertices $\{\vec{\mathbf{x}}_i\}$ with a data structure such as a *winged edge* data structure that allows one to determine the faces adjoining each edge and each vertex (see Section 5.4).

4.2.1 Paths on a Surface in 3D

First, we consider a surface represented by triangular facets in ordinary 3D space. Our basic hypothesis of *maximal projection* then requires that the facet of interest lies completely in the screen plane.

Next, as illustrated in Figure 4.4, we choose a point of interest $\vec{\mathbf{P}}$ in the facet and give an algorithm for transforming the display as we move to a new interest point $\vec{\mathbf{P}}'$. Assuming a 2D controller motion $\vec{\mathbf{dx}}$, we proceed as follows:

Clip. First, we clip the vector $\vec{\mathbf{P}} + \vec{\mathbf{dx}}$ to the triangle containing $\vec{\mathbf{P}}$. That is, working within the 2D screen coordinate system with triangle coordinates $(\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2)$, compute



(a) (b)

Figure 4.5: (a) Flattened facet-to-facet transition. (b) 3D facet rotation.

each normal $\vec{\mathbf{n}}_{01} = (-(y_1 - y_0), (x_1 - x_0))$, etc., and each $t = \vec{\mathbf{n}}_{01} \cdot (\vec{\mathbf{x}}_0 - \vec{\mathbf{P}})/\vec{\mathbf{n}}_{01} \cdot \vec{\mathbf{dx}}$, etc.; if all positive t's are greater than one, $\vec{\mathbf{P}}' = \vec{\mathbf{P}} + \vec{\mathbf{dx}}$ lies in the face; otherwise the minimum positive t gives the intersection point $\vec{\mathbf{P}}_{\text{edge}} = \vec{\mathbf{P}} + t\vec{\mathbf{dx}}$.

If $\vec{\mathbf{P}}'$ lies in the current face, simply translate that point to the screen center. If $\vec{\mathbf{P}}'$ lies in the adjacent face, rotate as prescribed below and then translate $\vec{\mathbf{P}}'$ to the screen center. If $\vec{\mathbf{P}}'$ crosses the clipping boundary of the adjacent face, repeat the clipping and rotation procedure until it lies within a face. This procedure amounts to flattening the neighboring facets as shown in Figure 4.5a to find the destination interest point $\vec{\mathbf{P}}'$ in the local 2D coordinate system of the final face.

Rotating between facets. In the facet model, vertices are points of vanishing measure that in fact carry the Gaussian curvature in their angular deficits (see Appendix E). For now, we assume we never actually pass through a vertex; all paths will be across edges from one face to another; smoothing models can in principle distribute the curvature across the faces and permit direct traversal of vertices.

We now compute the transition across a given edge, assuming we know the unit vectors $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ perpendicular to the transition edge, as indicated schematically in Figure 4.5b. We want to translate the origin to the point $\vec{\mathbf{P}}_{\text{edge}}$ and rotate by the angle $\cos\theta = \hat{\mathbf{A}} \cdot \hat{\mathbf{B}}$ in the plane containing $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ so that $\hat{\mathbf{B}}'$ now lies in the screen plane and is identical to the original $\hat{\mathbf{A}}$.

We use a Gram-Schmidt procedure to derive the norm-preserving rotation using the orthonormal basis vectors $\hat{\mathbf{A}}$ and $\hat{\mathbf{C}} = (\hat{\mathbf{B}} - \hat{\mathbf{A}}\cos\theta)/\sin\theta$. (If $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are parallel, no rotation takes place and we translate directly to $\vec{\mathbf{P}}'$ in the plane of the original facet.) Every vertex $\vec{\mathbf{V}}$ in the *entire* data structure undergoes the following rotation:

$$\begin{aligned}\vec{\mathbf{V}}' = \vec{\mathbf{V}} &- \hat{\mathbf{A}}(\hat{\mathbf{A}} \cdot \vec{\mathbf{V}}) - \hat{\mathbf{C}}(\hat{\mathbf{C}} \cdot \vec{\mathbf{V}}) \\ &+ \begin{bmatrix} \hat{\mathbf{A}} & \hat{\mathbf{C}} \end{bmatrix} \begin{bmatrix} \cos t\theta & \sin t\theta \\ -\sin t\theta & \cos t\theta \end{bmatrix} \begin{bmatrix} \hat{\mathbf{A}} \cdot \vec{\mathbf{V}} \\ \hat{\mathbf{C}} \cdot \vec{\mathbf{V}} \end{bmatrix}\end{aligned} \tag{4.1}$$

where we may vary t slowly between zero and one to avoid sudden jumps in the orientation.

Remark: One *cannot* use the vectors $\vec{\mathbf{P}}_{\text{edge}} - \vec{\mathbf{P}}$ and $\vec{\mathbf{P}}' - \vec{\mathbf{P}}_{\text{edge}}$ to define the rotation plane, since, although $\vec{\mathbf{P}}'$ would end up in the screen plane, the resulting rotation would in general tilt the common edge *out* of the screen plane.

4.2.2 Surfaces in Dimensions Higher Than Three

The equations presented so far in fact extend trivially to arbitrary dimensions for the vertex coordinates. The only essential difference is that the vector $\hat{\mathbf{B}}$ may have a

component in some direction $\hat{\mathbf{w}}$ outside the projection volume spanned by $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$, and the procedure given so far rotates only the *projection* of the face of the new interest point $\vec{\mathbf{P}}'$ into the screen plane. That projection may be slightly tilted and therefore not maximal unless we make an additional rotation in the z-w plane to line it up and eliminate the $\hat{\mathbf{w}}$ component; we may use Gram-Schmidt to define $\hat{\mathbf{w}}$ in any dimension as follows:

$$\begin{aligned}
\hat{\mathbf{B}}' &= \frac{\hat{\mathbf{B}} - \hat{\mathbf{x}}(\hat{\mathbf{x}} \cdot \hat{\mathbf{B}})}{(1 - (\hat{\mathbf{x}} \cdot \hat{\mathbf{B}})^2)^{1/2}} \\
\hat{\mathbf{B}}'' &= \frac{\hat{\mathbf{B}}' - \hat{\mathbf{y}}(\hat{\mathbf{y}} \cdot \hat{\mathbf{B}}')}{(1 - (\hat{\mathbf{y}} \cdot \hat{\mathbf{B}}')^2)^{1/2}} \\
\hat{\mathbf{w}} &= \frac{\hat{\mathbf{B}}'' - \hat{\mathbf{z}}(\hat{\mathbf{z}} \cdot \hat{\mathbf{B}}'')}{(1 - (\hat{\mathbf{z}} \cdot \hat{\mathbf{B}}'')^2)^{1/2}} \; .
\end{aligned}$$

Tessellated surfaces in arbitrary dimensions $N > 3$ can therefore be handled in a manner similar to our discussion for curves in dimensions $N > 2$. The essence of the argument, represented in Figure 4.6, is that the two triangles and four vertices of a winged edge in any dimension define a 3D subspace that we can require to be aligned with the 3D volume that eventually projects to the 2D screen. When making a transition to a new interest point $\vec{\mathbf{P}}'$, we introduce a *new* 3D subspace that shares the x-y plane with the facet containing $\vec{\mathbf{P}}$ and has a new component in the $\hat{\mathbf{B}}$ direction. If $\hat{\mathbf{B}}$ has components only in the $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ subspace, we are done.

If not, we rotate the entire object and generate a maximal projection that not only has $\vec{\mathbf{P}}'$'s face lying in the screen x-y plane, but also has the tetrahedron containing $\vec{\mathbf{P}}$ and $\vec{\mathbf{P}}'$ contained completely in the 3D projection volume. *This is the analog for surfaces of computing the Frenet frame curve normal and readjusting the curve*

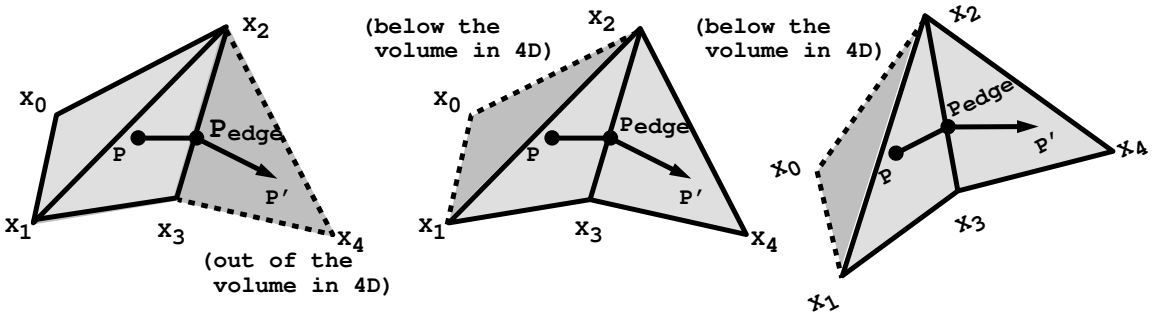


Figure 4.6: One possible approach to maintaining maximal projection of a 4D surface during the transition between facets. (a) Maximal projection of a tessellated section of a 4D surface is aligned so that the tetrahedron $(\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \vec{\mathbf{x}}_3)$ lies in the projection volume defined by the screen and the gaze direction. (b) Rocking $\vec{\mathbf{P}}'$ into the projection volume by rotating around the plane defined by $(\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \vec{\mathbf{x}}_3)$. (c) Rotate in the projection volume to achieve maximal projection of the facet containing $\vec{\mathbf{P}}'$.

normal to lie in the plane normal to the screen.

As in the space curve treatment, we have three alternatives: (1) Apply the analog of Eq. (4.1) to rotate $\hat{\mathbf{B}}$ in the z-w plane so that it has no $\hat{\mathbf{w}}$ component, then perform the standard 3D facet transition to align the new $\hat{\mathbf{B}}$ with the original $\hat{\mathbf{A}}$. (2) First align the new $\hat{\mathbf{B}}$ with the original $\hat{\mathbf{A}}$, then apply Eq. (4.1) in the z-w plane to eliminate the $\hat{\mathbf{w}}$ component. (3) Identify the 4D subspace (the 4-simplex) generated by the 5 vertices shown in Figure 4.6, compute the 4D frame with normal $\hat{\mathbf{m}}$ to the 3-simplex defined by the "old" winged edge faces, compute the distinct 4D frame with normal $\hat{\mathbf{n}}$ relative to the 3-simplex defined by the "new" winged edge faces containing $\vec{\mathbf{P}}$ and $\vec{\mathbf{P}}'$, then perform a geodesic rotation between the two 4D frames. Details of the required methods may be found in [27, 29].

Remark: Earlier, we showed how the Frenet-frame treatment of space curves is essentially exhausted in 3D. The method just described shows how a winged-edge

treatment of arbitrary surfaces naturally leads to 4D subspaces, and that no further complexities arise in developing an analogous treatment of surface geometry in higher dimensions. Extending this treatment to winged faces for walking three-manifolds intrinsically requires 5D subspaces; K-manifolds whose tessellated geometry is described by winged $(K-1)$-planes will require $(K+2)$-dimensional subspaces to treat the most general maximal projection requirements in arbitrary dimensions.

4.3 Facet-based vs Smoothly Interpolated Control

Our approach so far has treated manifolds as locally planar segments in the manner of flat-shaded polyhedral computer graphics models and the Regge calculus treatment of Riemannian geometry (Appendix E). Thus all transitions between facets across winged edges are indistinguishable from walking on flat space; all the curvature is contained in the "angular deficits," the difference between 2π and the actual sum of facet angles at each vertex. For relatively dense manifold triangulation, the angular deficit at each vertex is very small, and navigation of the manifold appears fairly smooth (see the examples in Figures G.16, G.17, and G.18). However, just as we may wish to approximate the appearance of smooth shaded surfaces starting from an underlying discrete tessellation, we may wish to extend the walker application with an interpolation scheme that makes the manifold appear to be smooth. For example, we see in Figure 4.3 that we really want a curve's segment-based Frenet normal and tangent aligned in the screen projection plane at $\vec{\mathbf{x}}_1$, not at $\vec{\mathbf{P}}$, which should already be interpolating between the orientations at $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$; a similar observation holds for surface walking as well.

The first step in generating a smoothly-varying manifold walk is to produce a local tangent line at each vertex of a curve, a local tangent plane at each vertex of a surface, and so on. The second, and more problematic, task is to produce a compelling interpolation procedure that connects these tangent frames continuously to one another over the entire manifold.

4.3.1 Surfaces in 3D

In 3D, we can use the standard interpolated shading algorithm that computes a vertex normal by averaging the normals to the surrounding faces; this normal defines the 3D tangent plane at the vertex. Then we can use bilinear interpolation of the vertex normals to generate face-interior normals at every point on the surface.

We modify the facet-based surface walking algorithm as follows (see Figure 4.7). Assume we start from point P on a facet and walk in a particular direction. We first find the edge point S as before. Then we calculate v_p and v_s, the point normals at P and S respectively, by linearly interpolating vertex normals at x_0, x_1 and x_2. If the distance between P and S and the angle between v_p and v_s are not big, we first translate from P to S and then do a 3D facet rotation in the winged-edge determined by vectors v_p, v_s and points P, S (the edge of the winged-edge is $\{S,\ S + v_p\}$, and the two triangles of the winged-edge are $\{P,\ S,\ S + v_p\}$ and $\{S,\ S + v_p,\ S + v_s\}$).

If the distance between P and S or the angle between v_p and v_s is too big, we can devide the path between P and S into smaller steps, as indicated by points Q and R in Figure 4.7. We then calculate the point normals at Q and R and do transitions from P to Q, from Q to R and from R to S in the same way as described in the previous paragraph. Because we use linear interpolation, the point normals v_p, v_q, v_r

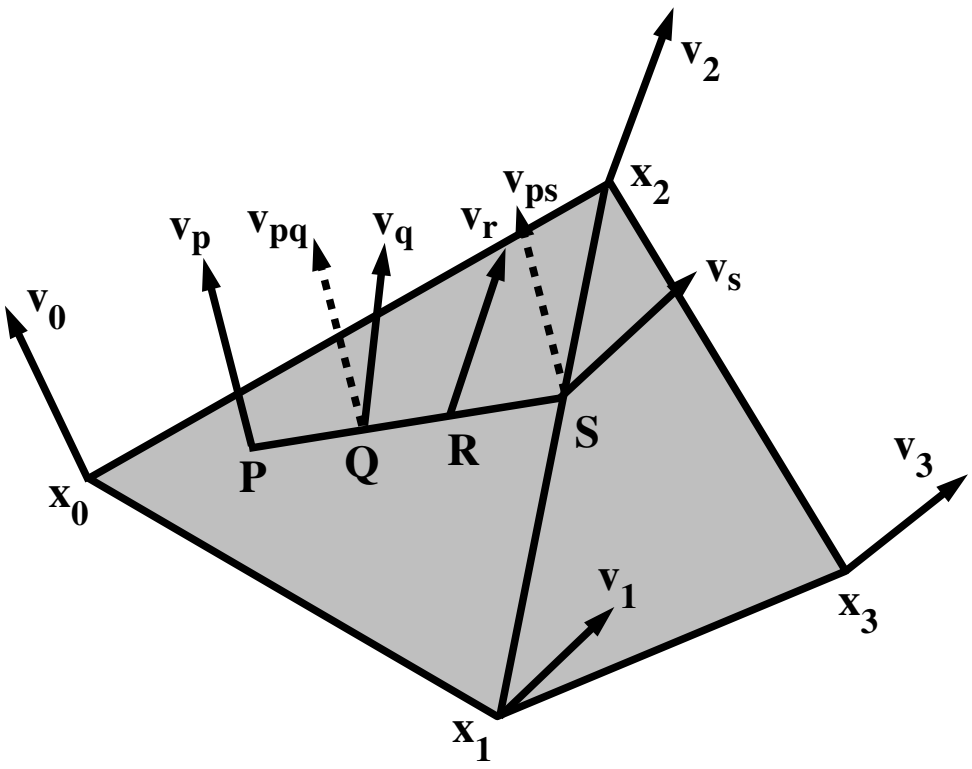


Figure 4.7: A smooth interpolation method for surfaces in 3D. We can either take one big step of translation (from P to S) and rotation (at the point S), or take smaller steps from P to Q, from Q to R and from R to S. Each step consists of a small translation and rotation. In this diagram, v_0, v_1, v_2 and v_3 are vertex normals at x_0, x_1, x_2 and x_3, respectively. v_p, v_q, v_r and v_s are point normals that are bilinearly interpolated using v_0, v_1, v_2 and the relative positions of P, Q, R and S in the triangle $x_0x_1x_2$. v_{pq} and v_{ps} are the same as v_p.

and v_s are in the same 2D plane. So all the 2D rotations at points Q, R and S can be combined and the combined 2D rotation is the same as the one when we take one step from P to S as in the previous paragraph. Hence, we achieve smoother animation while keeping the overall transformation the same.

4.3.2 In Dimensions Higher Than Three

In dimensions higher than three, there is no longer a normal vector to a surface, but a normal subspace of dimension $N - 2$.

Here is one way to find the "average" tangent plane. In 4D, we take two neighboring facets to define a 3-simplex, and take the 4D normal to that 3-simplex as a "facet normal." Repeating the procedure for successive pairs of facets around a vertex yields a set of 4D normals that can be averaged as in the 3D case. The resulting 4D vertex normal defines a 3D tangent hyperplane at the vertex. We then project the vertex and its surrounding facets into this 3D subspace and repeat the usual averaging of 3D face normals in this subspace to get a candidate for the tangent plane. We note that a 3-facet vertex is effectively in a 3D subspace to start with, so we need a 4-facet vertex to perform the 4D procedure. In 5D, a 4-facet vertex reduces to the 4D case, so the analogous procedure would start with 5-facet vertices and then iterate down through dimensions.

4.4 Interactive Interfaces

The interactive interfaces for the algorithms presented in the previous sections can be extended in many ways for particular applications and virtual reality devices.

4.4.1 User Alignment with Maximal Projection

The plane in which the maximal projection is enforced can be oriented arbitrarily in the 3D graphics coordinates of the projection volume. This plane is exactly analogous to the tube $y^2 + z^2 = r^2$ through which a space curve is forced to pass to align it with the x-axis of a 1D screen; the flat facets of a surface are effectively drawn through a hypertube $z^2 + w^2 = r^2$ in 4D and thus forced into the x-y plane. But this x-y plane can be oriented with the *screen* coordinate system, in which case we are "walking" on the surface with gravity pulling us in the gaze direction, or it could be oriented with the *floor*, in which case the maximal projection is aligned with the gaze direction and we get the impression of walking on a treadmill oriented to the gravitational pull of the outside world as we sit at the computer console.

4.4.2 Extension to Haptic Interfaces

Given any 3D haptic interface, we can make the surface, in either screen or treadmill orientation, an impenetrable barrier or a cage constraining our motion to 2D. In static mode, one could keep the object fixed with the center maximally projected and allow the user to feel a limited continuous neighborhood of the center point, thus getting a tactile sense of the surface shape. In dynamic mode, one can assign a velocity and direction to the displacement of the tactile probe from the center, thus empowering the user effectively to drive around the manifold while feeling the changes in shape and slope a short distance away from the central maximal projection point.

4.4.3 Direct Manipulation Characteristics

The interface we have described is a generalization of a context-free, direct manipulation 3D rolling-ball orientation controller (see, e.g., [25]) to arbitrary manifolds and dimensions. Properties such as non-commutativity of paths between points that are observed in this style of orientation control are transformed into the rich geometric properties and symmetry transformations of arbitrary topological spaces. In addition, just as a scene controlled by a rolling ball can move either in the same direction as the controller motion or in the opposite direction, we can traverse a manifold either way: walking forward to a new position on the manifold, or pushing the manifold in the controller direction.

4.4.4 Hansel and Gretel Navigation

A classic problem in the navigation of spaces is to determine where one has been and how to return to the starting point. Laying a trail of "bread crumbs" in the manner of the fairy tale of Hansel and Gretel is one approach to helping the user maintain his or her orientation in the space. A good user interface exploiting this technique would have a mode where all travel is marked and can be constrained to a 1D motion along the path already traversed. That way one can trivially return to any visited point without even watching where one is going. One-sided surfaces of course require distinct markings on opposite sides of the same facet.

4.4.5 Extension to Three-manifolds and Wands

The approach can be naturally extended to three-manifolds with winged faces, and so on.

In contrast to the Maniview [24] approach, which uses an "insider's" view emphasizing discrete symmetry groups and face identifications by producing repeated copies of the manifold sewn to itself in the 3D viewing space, we would navigate specific embedding of 3-manifolds in four or more dimensions.

This approach would fit well with applications treating 3D scalar fields as 4D terrain elevation maps [32], and would be ideally adapted to 4D orientation control using three-degree-of-freedom control devices such as a wand [18, 29]. Natural applications of this technique occur not only in topology and volume visualization, but also in the Regge calculus approach to general relativity [51].

4.5 Examples

We conclude with several visual examples. In practice, the concepts presented are strongly dependent on motion cues and the sense of direct manipulation, and so are difficult to represent with static images.

- In Figure G.16, we show a family of geodesic paths that would be taken by a walker exploring an ordinary 3D torus.
- We display in Figure G.17 selected geodesic paths resulting from walking on the spun trefoil knotted sphere embedded in 4D, projected to 3D. Note how the paths follow the continuous surface in 4D despite the presence of self-intersecting

surfaces in the 3D projection. The color coding on the 3D surface denotes the 4D depth from which it was projected, with blue near and red far in 4D.

- Figure G.18 shows a family of walker trajectories on the one-sided projective plane embedded in 4D and projected to 3D. The chosen projection results in a surface that it is part way between a cross-cap and Steiner's Roman surface. Here 4D depth is also denoted by color. One-sided surfaces present interesting logistical problems for path-tracing strategies, but the paths themselves are straightforward.

4.6 Conclusion

In this chapter, we have introduced a method for visualizing topological data structures by "walking" their geometry in a way that we feel has much potential for enhancing users' cognitive maps of a space. It is uniquely adapted to exploring high-dimensional mathematical data structures because of its ability to continuously flatten out the local neighborhood into the user's viewing space while maintaining a global context. For surfaces that appear self-intersecting in a particular projection, the method's locally-continuous traversal of the surface provides information interactively that is hidden even in a stereographic display. One can imagine many other applications, such as relational databases, 2D scalar fields, 3D scalar fields, and lattice models of general relativity. Furthermore, the method is ideally adaptable for newly-available haptic technologies, since it relies on local continuity that embodies an intrinsic model for force-feedback and tactile navigation.

Chapter 5

MeshView Visualization System

The purpose of this chapter is to present an overview of *MeshView*, an interactive system for viewing and manipulating objects embedded in 3D or 4D.

MeshView is in the public domain. It is available via anonymous ftp from `ftp.cs.indiana.edu:/pub/hma` (see [46]). It is relatively small (the size of the whole compressed package is under 1 MB) and is fine-tuned for speed. It is portable, written in OpenGL and X/Motif.

MeshView is not only a general-purpose visualization system, but also serves as a prototyping system for our ongoing research. It supports several unique features, such as the 4D rolling ball algorithm, context-free manipulation, color maps for 4th-dimensional depth, two-sided coloring for faces, a parametric space picker. It also supports *momentum* options for all motion controls and the 3D rolling ball algorithm. See Section 5.1 for more details. The package also comes with dozens of 3D and 4D data files, including *hypercube*, *Klein bottle*, *4D torus*, *Fermat surfaces*, *projective planes*, etc.

In many aspects, MeshView resembles *Geomview* ([49]). It reads Geomview data files (OFF, MESH, LIST) and has similar momentum features for motion controls. But MeshView provides many more specific tools for various types of objects, and has much faster 4D control.

In Section 5.1, we look at the features in version 1.0 and those that will be supported in a future release. Section 5.2 describes the philosophy and control of the 3D/4D rolling ball interface. We discuss the general-purpose C++ library *NL* and winged-edge based data structure *WING* in Sections 5.3 and 5.4 respectively.

5.1 Features

5.1.1 Version 1.0

We released the first version (version 1.0) of MeshView in July 1994. Color Plates G.19 and G.20 show the user interface and several 4D objects from the released package. Among other things, version 1.0 has the following features.

- Reads Geomview/OOGL (developed at the Geometry Center, University of Minnesota) MESH, OFF and LIST file formats.
- Rotates/translates/scales objects in 3D and 4D interactively under mouse control.
- Applies 3D and 4D rolling ball models to accomplish rotations. (see Section 5.2 for a description, also see [25, 29]).
- Momentum option available on all motions.

- Optionally draws faces, edges, vertices, normals, palette, lighting vectors, and a reference set of 4D axes.
- Supplies a wide range of color palette options for 4D depth color coding.
- Supplies an interactive parametric space "picker" (or point locator) for a MESH file or list of MESH files. Any individual mesh in a set of loaded patches can be selected in turn.
- Saves system state for later recovery, including current 3D and 4D viewing matrices, camera setting, background color, light direction and rendered `ppm` image of the current scene. This is useful for reconstructing the state of an illustration for a publication.
- Loads palettes and saved system states.
- Uses two different colors (or one color) for front-facing and back-facing surfaces. This feature is quite useful for surfaces embedded in 4D with self-intersecting 3D projections.
- Supports 3D perspective and orthogonal projections.
- Supports a choice between applying the 4D rolling ball to the current screen coordinates of the object's 3D projection (context-free, the default), or applying to the object's local 3D coordinate system context (use the "axes" display to help show the context). The latter is useful for looking at different sides of the object's 3D projection while performing a 4D rotation. This is needed mostly to accommodate a 2D mouse interface, and would not be as useful for a 3D mouse interface.

- Includes a selection of example geometry files, including 4D flat torus, Steiner surface (RP^2 embedded in 4D), 4D Fermat surfaces and a lot more. (see the README file in the distribution package for details)
- Supplies online help.

5.1.2 Version 2.0

In the next release (version 2.0, expected February 1996), the following features will be added.

- Supports 4D polar (perspective) projection.
- Supports a new data file format WING and provides converters from MESH and OFF file format.
- Supplies automatic manifold patch sewing and optimization to support complicated ad hoc topologies.
- Supplies 3D/4D geodesic generator.
- Supplies 3D/4D manifold walker.
- Supports interactive curve tubing and ribboning.
- Supplies tools for creating movies.

5.2 The Rolling Ball Interface

This section explains the rolling ball interface in MeshView.

To understand the basic philosophy of the rolling ball interface, start with a 3D object such as square.3mesh in the distribution package. Use only the left mouse button (3D rolling ball): pulling the mouse in the +x (screen-right) direction tilts the 2D object so that the left edge comes towards the viewer, acquiring positive z-components, and the right edge dips away from the viewer into the screen, acquiring negative values of z. Pushing the mouse in the +y direction (screen-up) tilts the lower edge out into positive z and the upper edge into the screen in the negative z direction.

Now load a solid 3D object like sphere.3mesh or torus.3mesh. These objects automatically have vanishing 4th components (call it w) at every point when loaded, just as the flat objects had vanishing z-components. Now use only the SHIFT-LEFT (4D rolling ball) mouse button: pulling the mouse in the +x (screen-right) direction tilts the 3D object in 4D so that the left side is acquiring a positive w component and getting squashed in x, while the right side is tilting away in w, getting a negative w component. Turn on 4D depth pseudo- coloring to make this dramatically apparent (Ctrl+3, Ctrl+b) - the left side turns red while the right turns blue with the default colormap. Pulling the mouse in the +y direction (screen-up) gives the bottom a positive w and the top a negative w component while squashing the whole object in the y direction. The shift-left mouse control generates x-w and y-w 4D rolling ball rotations. Now use only the SHIFT-MIDDLE mouse button, which generates x-w and z-w 4D rolling ball rotations. Moving the mouse in the +x direction has the same effect as shift-left, but pulling the mouse down (screen-down direction) pushes the BACK side of the object out into positive w, and the FRONT side acquires a negative w component; pushing the mouse up (screen-up direction) has the opposite

effect.

If a 3D mouse is available, all 3 of these motions, or any linear combination, are available at once and the splitting into the two shift-left and shift-middle controls is not needed. The default mode, in which all 4D rolling ball motions are referred to the screen-based xyz coordinate system with z out of the screen, is the most natural one to emulate a 3D mouse. However, when only a 2D mouse is available, an alternate mode is provided that can be quite useful: deactivate the "Context_free" check box on the "Motion" menu. This will allow the user to rotate the object to any 3D orientation (using mouse-left or mouse- shift-right), but now shift-left will follow the object's original x-y axes, and rotate the object in 4D as though the 2D mouse were a 3D mouse moving in the object's own original x-axis direction, NOT the current screen x. Analogous behavior occurs for y motions and for shift-middle x and y mouse motions. Turn on the drawing of xyzw axes to see the different effects.

5.3 NL – An nD Mathematics C++ Library

NL is a C++ template library of nD vectors and nD matrices. It uses templates, inheritance, operator overloading and other features of C++ to support abstractions for the creation of and operations on nD vectors and matrices.

Figure 5.1 shows the hierarchies of classes in NL. In the base class `nlVector<int>`, we provide general implementations of nD vector operations. These methods are overridden by faster implementations in the derived classes `nlVector3`, `nlHPoint3` and `nlVector4`. The same methodology has been applied to the matrix hierarchy.

The NL library also contains the `nlQuaternion` class, which is derived from

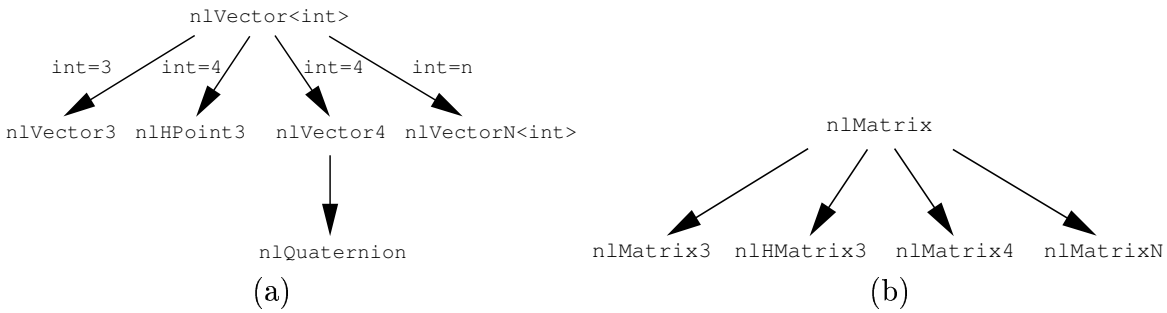


Figure 5.1: (a) Class hierarchy of nlVector; (b) Class hierarchy of nlMatrix.

`nlVector4`. It supports operations like multiplication, division, conjugation, exponential functions and logarithmic functions, and conversions to and from `nlMatrix3` and `nlHMatrix3`.

The following is a complete list of the header files in the NL library:

```
nlBoolean.h  nlMacros.h
nlVector.h   nlVector3.h  nlVector4.h  nlVectorN.h
nlHPoint3.h  nlPoint3.h   nlPoint4.h   nlQuaternion.h
nlMatrix.h   nlMatrix3.h  nlMatrix4.h  nlHMatrix3.h
```

5.4 WING – Winged-edge Data File Format

In this section, we describe the WING data format and compare it with another data format which is popular in several commercial and non-commercial packages. The winged-edge data structure was first introduced by Baumgart in [8], and further explored in [45, 63]. Glassner in [22] solved some problems in implementing a winged-edge library. Our data structure is simpler because we only allow triangles as faces.

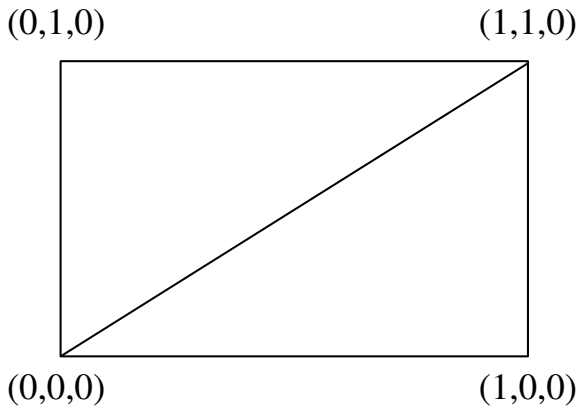


Figure 5.2: A simple object: a rectangle or two triangles.

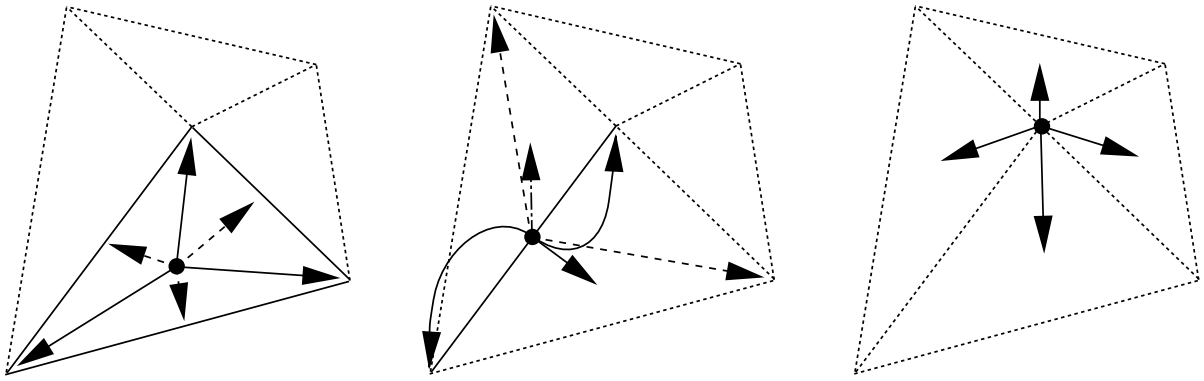
```
OFF
4 2 5    # Nvertices, Nfaces, Nedges (optional)
0.0 0.0 0.0    # vertex0 coordinates
0.0 1.0 0.0    # vertex1 coordinates
1.0 1.0 0.0    # vertex2 coordinates
0.0 1.0 0.0    # vertex3 coordinates
3 0 1 2  # face0, connects vertices 0, 1 and 2
3 0 2 3  # face1, connects vertices 0, 2 and 3
```

Figure 5.3: A simple OFF object

5.4.1 An Example

Let's first look at how to represent the simple object in Figure 5.2. In most graphics packages, such as Geomview and SGI/WaveFront, the description of an object like this is usually a list containing information about each vertex, followed by another list containing information about each face. The information about a vertex may include coordinates, normals and colors; the information about a face may include indices to the vertices on the face. Figure 5.3 shows a valid Geomview OFF description for the object in Figure 5.2.

The information stored in the OFF format is enough for general rendering purposes. We can easily access the information of each vertex of a triangle to draw either



(a) (b) (c)

Figure 5.4: Relationships between vertex, edge and face in WING data format. (a) A vertex contains pointers to all its neighboring faces; (b) An edge contains pointers to its two vertices, its two neighboring faces and corresponding third vertices; (c) A face contains pointers to its three vertices and three edges.

the triangle, the edges or just the vertices. But it is not enough for the walking interface as described in Chapter 4. We need to store more information at vertices, edges and faces.

For example, at a vertex we need to know which faces it connects to (see Figure 5.4a); at an edge, we need to know the two vertices that form the edge, the two (or one, if the edge is a boundary edge) faces it connects and the two corresponding vertices on the two faces (see Figure 5.4b); at a face, we need to know the three vertices and the three edges that form the face (see Figure 5.4c).

Figure 5.5 shows the description of the object in Figure 5.2 using WING format. As you can see, for each vertex, after the information about coordinates, we store the number of faces this vertex connects to, followed by the list of indices of the faces. For each edge, we store the indices of the two vertices that form the edge, the indices of the two faces that shared the edge (`-1` means the corresponding face doesn't exist and the edge is a boundary edge), and the indices of the two vertices on the two faces.

```
WING
4 5 2    # Nvertices Nedges Nfaces
# vertices (coordinates, nFaces, faces)
0.0 0.0 0.0   2 0 1    # 0
0.0 1.0 0.0   1 0      # 1
1.0 1.0 0.0   1 1      # 2
0.0 1.0 0.0   2 0 1    # 3
# edges: (2 vertices, 2 faces, 2 3rd vertices)
0 2  1 -1  3 -1        # 0
1 3  0 -1  0 -1        # 1
0 1  0 -1  3 -1        # 2
2 3  1 -1  0 -1        # 3
0 3  0  1  1 2         # 4
# faces: (3 vertices, 3 edges)
0 1 3  1 2 4           # 0
0 3 2  0 3 4           # 1
```

Figure 5.5: A simple WING object

For each face, we store the indices of the three vertices and the three edges that form the triangle.

Users may find that it is tedious to create a WING object by drawing and counting. In MeshView, we provide converters from MESH and OFF formats to WING format. Users can load either MESH or OFF objects and convert them to WING format dynamically, or they can convert files to WING format first to save the preprocessing time.

In our implementation of MeshView, we store a list of vertices, a list of edges and a list of faces for each surface object. With the WING data structure, we can access all the information needed for the walking interface. We believe that this data structure could be useful in other interactive applications.

Mark

```
for each pair of vertices vi and vj
  if vi and vj are very close
    Check if they share an edge.
      If so, delete that edge;
    For all vertices w (0, 1 or 2 of them), s.t.,
      both w-vi and w-vj form boundary edges,
      replace w-vj with w-vi;
    Replace all the pointers to vj by vi;
    Add the face list at vj to the face list at vi;
    Set self-index of vj to -1;
  endif
endfor
```

Sweep

```
Find the number of valid vertices, edges and faces and
  allocate new memory for them;
Copy the old valid data structure to the new data
  structure and reset the self-indices within both old
  and new data structures;
Reset pointer-indices within the new data structure;
Delete the old data structure.
```

Figure 5.6: A mark-and-sweep algorithm for eliminating degenerate triangles within a WING object.

5.4.2 Data Structure Manipulation

Both the data file stored on disk and the data structure in memory are index-based. We store two kinds of indices, the *pointer-index* as used in Figure 5.5 and the *self-index*, which is stored in each vertex, edge and face. The self-index of a vertex is used to record the vertex's position within a vertex list, and similarly for edges and faces.

In many situations, e.g., walking on a surface, we need to eliminate degenerate triangles, the triangles in which two of the three vertices are very close. We can use

a mark-and-sweep based algorithm to do this. See Figure 5.6 for the pseudo-code.

Chapter 6

Future Research and Conclusion

6.1 Future Research

The major results of this dissertation concern the exploitation of the detailed properties of coordinate frames to study the nature of curves and surfaces. Many interesting questions have arisen in the course of this research that suggest challenging new topics for investigation. Among these directions for future research, we note the following topics:

6.1.1 Framing Methods

A rich set of mathematical and numerical problems related to framing methods remains to be explored (also see Appendix F):

- Extend smooth interpolation methods for frames on surfaces in 3D (see Section 4.3) to frames on surfaces or other manifolds in higher dimensional space.

- Explore numerical methods for generating parallel transport frames along manifolds that are embedded in non-Euclidian space.
- Explore smooth interpolation methods for high dimensional coordinate frames.

6.1.2 Applications of Coordinate Frames

Coordinate frames provide substantially more information about the surface being visualized than traditional surface normals. Frames can be used to enhance many traditional computer graphics methods. The following is one example (which is not examined in this dissertation).

Static and dynamic texture mapping. For texture mapping, we usually need to specify texture coordinates at the vertices of the surface. This is the same as attaching a coordinate system to the surface. With our framing methods, we can assign a texture mapping that depends on the geometry of the surface, or we can dynamically reassign a texture mapping at every step of navigation to reflect the changing environment. This resembles a bump map [11] but is more general, since frames have one additional degree of freedom.

Essentially any application that requires local or global orientation alignment in 3D space could potentially benefit from our static or dynamic framing methods. In particular, 4D dot products between quaternions form a uniform similarity measure that can be exploited for applications such as anisotropic shading heuristics [42, 50, 62, 64], and forces between oriented particles in a particle system [60].

6.1.3 Domain-dependent Navigation

A picture is worth a thousand words, a movie even more. Most desktop workstations are or will soon be capable of interactive graphical animation. But the current navigation tools are quite primitive. We believe our domain dependent navigation methodology can greatly enhance the user's mental model of the environment being visualized. We can apply this methodology to other domains besides curves and surfaces. For example:

Navigating Relational Databases. We may construct a 3D scene using three attributes of a database table as 3D coordinates and color-code the entities using another attribute. When we navigate through the 3D scene, we may constrain our camera, both the position and the orientation, using yet another attribute or a property which depends upon several attributes. Taking a database of stocks as an example, when we navigate through a huge amount of stocks, we may want the camera to automatically stop at the stocks that have big percentage gains.

Graph Traversal. When we fly through a 3D graph, we may constrain the camera motion using information about nearby nodes and edges.

6.2 Conclusion

In the world of computer technology, 3D graphics is playing an increasingly important role. Methods of displaying lighted objects on a 2D computer screen using their 3D positions and surface normals are well understood and are now supported by graphics hardware. But for many applications such as interactive navigation, curve tubing and

anisotropic shading, 3D positions and surface normals are not sufficient. Users need to specify coordinate frames at points of interest.

In this dissertation, we have systematically studied the problem of attaching moving coordinate frames to curves and surfaces, and thus have built an important foundation for a wide range of applications in computer graphics and visualization. Our methods of constructing the frames are based on the intrinsic geometry of the objects and have optimal behavior for many classes of applications.

We have made a great deal of progress on the problem of interactive tools that exploit coordinate frames and their quaternion forms for visualizing complex curves and surfaces, but there are many exciting facets yet to be explored.

Appendix A

3D Rotation Matrix

The 3D rotation matrix used in the algorithm 2.6 in Section 2.2 is (also see [20]):

$$\mathbf{R}(\hat{\mathbf{n}}, \theta) = \begin{bmatrix} c + (n_1)^2(1-c) & n_1 n_2(1-c) - sn_3 & n_3 n_1(1-c) + sn_2 \\ n_1 n_2(1-c) + sn_3 & c + (n_2)^2(1-c) & n_3 n_2(1-c) - sn_1 \\ n_1 n_3(1-c) - sn_2 & n_2 n_3(1-c) + sn_1 & c + (n_3)^2(1-c) \end{bmatrix},$$

where $c = \cos\theta$ and $s = \sin\theta$ and $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$.[1]

[1] In this dissertation, $\vec{\mathbf{x}}$ indicates that $\vec{\mathbf{x}}$ is a vector while $\hat{\mathbf{x}}$ indicates that $\hat{\mathbf{x}}$ is a unit vector.

Appendix B

Computing the Tangent of Osculating Circle

Suppose we are given three points $(\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2)$ in a Euclidean space of arbitrary dimension. Then we may compute the center and radius of the osculating circle, as well as the tangent direction at $\vec{\mathbf{x}}_1$ as follows. First use a Gram-Schmidt procedure to compute the direction $\vec{\mathbf{u}}$ perpendicular to the chord $(\vec{\mathbf{x}}_1 - \vec{\mathbf{x}}_0)$:

$$\vec{\mathbf{u}} \;=\; (\vec{\mathbf{x}}_2 - \vec{\mathbf{x}}_1) - (\vec{\mathbf{x}}_1 - \vec{\mathbf{x}}_0)\frac{(\vec{\mathbf{x}}_2 - \vec{\mathbf{x}}_1)\cdot(\vec{\mathbf{x}}_1 - \vec{\mathbf{x}}_0)}{(\vec{\mathbf{x}}_1 - \vec{\mathbf{x}}_0)\cdot(\vec{\mathbf{x}}_1 - \vec{\mathbf{x}}_0)} \;.$$

Next, note that the center $\vec{\mathbf{x}}_c$ may be written as a vector from chord midpoint $\vec{\mathbf{a}} = (\vec{\mathbf{x}}_1 + \vec{\mathbf{x}}_0)/2$ in the perpendicular direction

$$\vec{\mathbf{x}}_c = \vec{\mathbf{a}} + t\hat{\mathbf{u}}$$

where $\hat{\mathbf{u}} = \vec{\mathbf{u}}/\|\vec{\mathbf{u}}\|$ and t is to be computed. Then, since each of the vectors $(\vec{\mathbf{x}}_0,\ \vec{\mathbf{x}}_1,\ \vec{\mathbf{x}}_2)$ is a distance R from $\vec{\mathbf{x}}_c$,

$$\begin{aligned} R^2 &= \|\vec{\mathbf{x}}_c - \vec{\mathbf{x}}_1\|^2 \\ &= \|\vec{\mathbf{x}}_c - \vec{\mathbf{x}}_2\|^2 \\ &= \|\vec{\mathbf{a}} - \vec{\mathbf{x}}_1\|^2 + 2t\hat{\mathbf{u}} \cdot (\vec{\mathbf{a}} - \vec{\mathbf{x}}_1) + t^2 \\ &= \|\vec{\mathbf{a}} - \vec{\mathbf{x}}_2\|^2 + 2t\hat{\mathbf{u}} \cdot (\vec{\mathbf{a}} - \vec{\mathbf{x}}_2) + t^2 \end{aligned}$$

Since $\hat{\mathbf{u}} \cdot (\vec{\mathbf{a}} - \vec{\mathbf{x}}_1) = 0$ by construction and the t^2 terms cancel, we find

$$t = \frac{\|\vec{\mathbf{a}} - \vec{\mathbf{x}}_2\|^2 - \|\vec{\mathbf{a}} - \vec{\mathbf{x}}_1\|^2}{2\hat{\mathbf{u}} \cdot (\vec{\mathbf{x}}_2 - \vec{\mathbf{a}})}$$

The value of R follows at once, and the direction of the tangent vector may be computed by applying Gram-Schmidt to $(\vec{\mathbf{x}}_0 - \vec{\mathbf{x}}_1)$:

$$\vec{\mathbf{T}}(\vec{\mathbf{x}}_1) = (\vec{\mathbf{x}}_0 - \vec{\mathbf{x}}_1) - (\vec{\mathbf{x}}_c - \vec{\mathbf{x}}_1)\frac{(\vec{\mathbf{x}}_c - \vec{\mathbf{x}}_1) \cdot (\vec{\mathbf{x}}_0 - \vec{\mathbf{x}}_1)}{(\vec{\mathbf{x}}_c - \vec{\mathbf{x}}_1) \cdot (\vec{\mathbf{x}}_c - \vec{\mathbf{x}}_1)}$$

Appendix C

Correctness of Continuous Limit

C.1 Background

For any twice-differentiable curve $\vec{\mathbf{x}}(s)$ with tangent vector $\hat{\mathbf{T}}(s)$ and any corresponding coordinate frame $(\hat{\mathbf{T}}(s), \hat{\mathbf{N}}(s), \hat{\mathbf{B}}(s))$, the following formula holds for some functions $k_1(s)$, $k_2(s)$, $\tau(s)$ because of the orthonormality constraints:

$$\begin{bmatrix} \hat{\mathbf{T}}'(s) \\ \hat{\mathbf{N}}'(s) \\ \hat{\mathbf{B}}'(s) \end{bmatrix} = \begin{bmatrix} 0 & k_1 & k_2 \\ -k_1 & 0 & \tau \\ -k_2 & -\tau & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}}(s) \\ \hat{\mathbf{N}}(s) \\ \hat{\mathbf{B}}(s) \end{bmatrix} . \tag{C.1}$$

Theorem 1. Let $\hat{\mathbf{V}}(s)$ be a unit normal vector field on the curve $\vec{\mathbf{x}}(s)$, let $(\hat{\mathbf{T}}(s), \hat{\mathbf{N}}(s), \hat{\mathbf{B}}(s))$ be any framing on the curve, and let $\alpha(s)$ be the angle between $\hat{\mathbf{V}}(s)$ and $\hat{\mathbf{N}}(s)$. Then $\hat{\mathbf{V}}(s)$ is parallel if and only if, for any s_1 and s_2,

$$\alpha(s_2) = \left(\alpha(s_1) - \int_{s_1}^{s_2} \tau(s)\, ds \right) \bmod (2\pi) . \tag{C.2}$$

Proof:

$$\hat{\mathbf{V}} = \cos(\alpha)\hat{\mathbf{N}} + \sin(\alpha)\hat{\mathbf{B}}$$

$$\hat{\mathbf{V}}' = \cos(\alpha)\hat{\mathbf{N}}' - \alpha' \sin(\alpha)\hat{\mathbf{N}} + \sin(\alpha)\hat{\mathbf{B}}' + \alpha' \cos(\alpha)\hat{\mathbf{B}}$$

Plugging in Eq. (C.1), and remembering that, by definition, $\hat{\mathbf{V}}$ is parallel if and only if both the coefficients of $\hat{\mathbf{N}}$ and $\hat{\mathbf{B}}$ are zero, we find

$$-\sin(\alpha)\alpha' - \sin(\alpha)\tau = 0$$

$$+\cos(\alpha)\alpha' + \cos(\alpha)\tau = 0,$$

which are equivalent to

$$\alpha' = -\tau,$$

or

$$\alpha(s_2) = (\alpha(s_1) - \int_{s_1}^{s_2} \tau(s)\, ds) \mod (2\pi) \ .$$

QED.

Remark. If the frame is a Frenet frame, τ is the classical torsion.

C.2 Proof of Smooth Limit

First, define the *norm of a tessellation* to be the maximum length of line segment in the representation of a curve. We wish to show that, as the norm of a curve's tessellation approaches zero, the result of parallel transporting a normal vector using

our algorithm (see Figure 2.6) approaches the parallel transport vector field on a smooth curve.

We assume that the input $\{\hat{\mathbf{T}}_i\}$ to the algorithm are the actual tangents of the smooth curve.

Let

$$\begin{aligned} \hat{\mathbf{B}}_i &= \frac{\hat{\mathbf{T}}_i \times \hat{\mathbf{T}}_{i+1}}{\|\hat{\mathbf{T}}_i \times \hat{\mathbf{T}}_{i+1}\|} \\ \hat{\mathbf{N}}_i &= \hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i \ , \end{aligned}$$

then $(\hat{\mathbf{T}}_i, \hat{\mathbf{N}}_i, \hat{\mathbf{B}}_i)$ gives a local frame at $\vec{\mathbf{x}}_i$.

Let

$$\hat{\mathbf{V}}_i = \cos(\alpha_i)\hat{\mathbf{N}}_i + \sin(\alpha_i)\hat{\mathbf{B}}_i \ . \tag{C.3}$$

The algorithm preserves $\sin(\alpha_i)\hat{\mathbf{B}}_i$, and rotates $\cos(\alpha_i)\hat{\mathbf{N}}_i$ to $\cos(\alpha_i)(\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i)$, hence

$$\hat{\mathbf{V}}_{i+1} = \cos(\alpha_i)(\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i) + \sin(\alpha_i)\hat{\mathbf{B}}_i \ . \tag{C.4}$$

Projecting out the components of $\hat{\mathbf{V}}_{i+1}$ in the direction of $\hat{\mathbf{N}}_{i+1}$ and $\hat{\mathbf{B}}_{i+1}$, we find, with $\cos(\theta_i) = \hat{\mathbf{B}}_{i+1} \cdot \hat{\mathbf{B}}_i$,

$$\begin{aligned} \hat{\mathbf{V}}_{i+1} \cdot \hat{\mathbf{N}}_{i+1} &= \cos(\alpha_i)(\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i) \cdot \hat{\mathbf{N}}_{i+1} + \sin(\alpha_i)\hat{\mathbf{B}}_i \cdot \hat{\mathbf{N}}_{i+1} \\ &= \cos(\alpha_i)(\hat{\mathbf{N}}_{i+1} \times \hat{\mathbf{T}}_{i+1} \cdot \hat{\mathbf{B}}_i) + \sin(\alpha_i)(\hat{\mathbf{B}}_i \cdot \hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_{i+1}) \\ &= \cos(\alpha_i)(\hat{\mathbf{B}}_{i+1} \cdot \hat{\mathbf{B}}_i) + \sin(\alpha_i)(\hat{\mathbf{B}}_{i+1} \times \hat{\mathbf{B}}_i \cdot \hat{\mathbf{T}}_{i+1}) \\ &= \cos(\alpha_i)\cos(\theta_i) - \sin(\alpha_i)\sin(\theta_i) \\ &= \cos(\alpha_i + \theta_i) \ , \\ \hat{\mathbf{V}}_{i+1} \cdot \hat{\mathbf{B}}_{i+1} &= \cos(\alpha_i)(\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i) \cdot \hat{\mathbf{B}}_{i+1} + \sin(\alpha_i)\hat{\mathbf{B}}_i \cdot \hat{\mathbf{B}}_{i+1} \end{aligned}$$

$$\begin{aligned} &= \cos(\alpha_i)(\hat{\mathbf{B}}_i \times \hat{\mathbf{B}}_{i+1} \cdot \hat{\mathbf{T}}_{i+1}) + \sin(\alpha_i)(\hat{\mathbf{B}}_i \cdot \hat{\mathbf{B}}_{i+1}) \\ &= \cos(\alpha_i)\sin(\theta_i) + \sin(\alpha_i)\cos(\theta_i) \\ &= \sin(\alpha_i + \theta_i) \ . \end{aligned}$$

Therefore $\alpha_{i+1} = (\alpha_i + \theta_i) \mod (2\pi)$. By induction, for any indices $i_1 < i_2$,

$$\alpha_{i_2} = \alpha_{i_1} + \sum_{j=i_1}^{i_2-1} \theta_j \mod (2\pi) \ .$$

As the tessellation becomes finer and finer, the discrete frame field $\{(\hat{\mathbf{T}}_i, \hat{\mathbf{N}}_i, \hat{\mathbf{B}}_i)\}_{i=0}^{N}$ approaches a framing $(\hat{\mathbf{T}}, \hat{\mathbf{N}}, \hat{\mathbf{B}})$ on the smooth curve. Since $\hat{\mathbf{B}}_i \perp \hat{\mathbf{T}}_{i+1}$ and $\hat{\mathbf{B}}_{i+1} \perp \hat{\mathbf{T}}_{i+1}$, then

$$\hat{\mathbf{B}}' \perp \hat{\mathbf{T}} \qquad \Rightarrow \qquad k_2 = 0 \qquad \Rightarrow \qquad \hat{\mathbf{B}}' = -\tau\hat{\mathbf{N}} \ .$$

Since $\|\hat{\mathbf{B}}_{i+1} - \hat{\mathbf{B}}_i\| \approx |\theta_i|$, then

$$\tau = -\theta'$$

Fixing $\vec{\mathbf{x}}_{i_1}$ and $\vec{\mathbf{x}}_{i_2}$, and letting the norm of the tessellation approach zero, we have

$$\begin{aligned} \alpha_{i_2} \quad &= \quad (\alpha_{i_1} + \sum_{j=i_1}^{i_2-1} \theta_j) \mod (2\pi) \\ &\longrightarrow \quad (\alpha_{i_1} + \int_{\vec{\mathbf{x}}_{i_1}}^{\vec{\mathbf{x}}_{i_2}} \theta'(s)\, ds) \mod (2\pi) \\ &= \quad (\alpha_{i_1} - \int_{\vec{\mathbf{x}}_{i_1}}^{\vec{\mathbf{x}}_{i_2}} \tau(s)\, ds) \mod (2\pi) \ . \end{aligned}$$

Hence, by **Theorem 1**, the parallel transport algorithm approaches the parallel transport vector field as the norm of the tessellation goes to zero. QED.

C.3 Comparison of the PT and the Projection Algorithms

In this section, we are going to use Taylor's expansion to show that both the PT algorithm (Figure 2.3) and the projection algorithm (Eq. 2.4) approach the parallel vector field as the norm of the tessellation approaches zero. But the PT algorithm converges faster than the projection algorithm.

C.3.1 The Projection Algorithm

Taylor's expansion of $\hat{\mathbf{T}}$ gives

$$\hat{\mathbf{T}}_{i+1} = \hat{\mathbf{T}}_i + \vec{\mathbf{T}}' \, \Delta s + O(\Delta s^2), \tag{C.5}$$

where Δs is the length of the line segment.

If we let $\vec{\mathbf{W}} = \hat{\mathbf{V}}_i - \hat{\mathbf{T}}_{i+1}(\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_{i+1})$ (see Eq. 2.4) and assume $\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_i = 0$, then

$$\begin{aligned}
\vec{\mathbf{W}} &= \hat{\mathbf{V}}_i - (\hat{\mathbf{T}}_i + \vec{\mathbf{T}}'\Delta s + O(\Delta s^2)) \; (\hat{\mathbf{V}}_i \cdot (\hat{\mathbf{T}}_i + \vec{\mathbf{T}}'\Delta s + O(\Delta s^2))) \\
&= \hat{\mathbf{V}}_i - (\hat{\mathbf{T}}_i + \vec{\mathbf{T}}'\Delta s + O(\Delta s^2)) \; ((\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\Delta s + O(\Delta s^2)) \\
&= \hat{\mathbf{V}}_i - (\hat{\mathbf{T}}_i(\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\Delta s + O(\Delta s^2)) \\
&= \hat{\mathbf{V}}_i - \hat{\mathbf{T}}_i(\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\Delta s + O(\Delta s^2), \\
\|\vec{\mathbf{W}}\| &= (\vec{\mathbf{W}} \cdot \vec{\mathbf{W}})^{1/2} \\
&= (\hat{\mathbf{V}}_i \cdot \hat{\mathbf{V}}_i - 2(\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_i)(\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\Delta s + O(\Delta s^2))^{1/2} \\
&= 1 + O(\Delta s), \\
\hat{\mathbf{W}} = \frac{\vec{\mathbf{W}}}{\|\vec{\mathbf{W}}\|} &= \hat{\mathbf{V}}_i - \hat{\mathbf{T}}_i(\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\Delta s + O(\Delta s^2).
\end{aligned}$$

Hence,

$$\frac{\Delta\vec{\mathbf{V}}_i}{\Delta s} = \frac{\hat{\mathbf{V}}_{i+1} - \hat{\mathbf{V}}_i}{\Delta s} = \frac{\hat{\mathbf{W}} - \hat{\mathbf{V}}_i}{\Delta s} = -(\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\ \hat{\mathbf{T}}_i + O(\Delta s). \tag{C.6}$$

Therefore, as $\Delta s \to 0$, the component of $\frac{\Delta\vec{\mathbf{V}}_i}{\Delta s}$ that is not parallel to $\hat{\mathbf{T}}_i$ *linearly* approaches zero.

C.3.2 The PT Algorithm

From Eqs. C.3 and C.4, we have,

$$\begin{aligned}
\Delta\vec{\mathbf{V}}_i = \hat{\mathbf{V}}_{i+1} - \hat{\mathbf{V}}_i &= \cos(\alpha_i)\ (\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i - \hat{\mathbf{N}}_i) \\
&= (\hat{\mathbf{V}}_i \cdot \hat{\mathbf{N}}_i)\ (\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i - \hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i) \\
&= (\hat{\mathbf{V}}_i \cdot (\hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i))\ (\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i - \hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i).
\end{aligned} \tag{C.7}$$

Notice that both $\hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i$ and $\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i$ live in the space spanned by $\hat{\mathbf{T}}_i$ and $\hat{\mathbf{T}}_{i+1}$. There exist real numbers a, b, c and d such that:

$$\hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i = a\ \hat{\mathbf{T}}_i + b\ \hat{\mathbf{T}}_{i+1} \tag{C.8}$$

$$\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i = c\ \hat{\mathbf{T}}_i + d\ \hat{\mathbf{T}}_{i+1}. \tag{C.9}$$

If we let $\cos\theta = \hat{\mathbf{T}}_i \cdot \hat{\mathbf{T}}_{i+1}$ and apply dot products to both sides of Eq. C.8 with $\hat{\mathbf{T}}_i$ and $\hat{\mathbf{T}}_{i+1}$, then

$$\begin{aligned}
(\hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i) \cdot \hat{\mathbf{T}}_i &= a\ \hat{\mathbf{T}}_i \cdot \hat{\mathbf{T}}_i + b\ \hat{\mathbf{T}}_{i+1} \cdot \hat{\mathbf{T}}_i \\
(\hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i) \cdot \hat{\mathbf{T}}_{i+1} &= c\ \hat{\mathbf{T}}_i \cdot \hat{\mathbf{T}}_{i+1} + d\ \hat{\mathbf{T}}_{i+1} \cdot \hat{\mathbf{T}}_{i+1},
\end{aligned}$$

or

$$\begin{aligned} 0 &= a + b\ \cos\theta \\ -\sin^2\theta &= a\ \cos\theta + b. \end{aligned}$$

We get $a = 1 - \cos\theta$ and $b = -1$, so

$$\hat{\mathbf{T}}_i \times \hat{\mathbf{B}}_i = (1 - \cos\theta)\ \hat{\mathbf{T}}_i - \hat{\mathbf{T}}_{i+1}. \tag{C.10}$$

Similarly, we can get:

$$\hat{\mathbf{T}}_{i+1} \times \hat{\mathbf{B}}_i = \hat{\mathbf{T}}_i - \cos\theta\ \hat{\mathbf{T}}_{i+1}, \tag{C.11}$$

If we now apply Eqs. C.10 and C.11 to Eq. C.7 and assume $\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_i = 0$, we have

$$\begin{aligned} \Delta\vec{\mathbf{V}}_i &= (\hat{\mathbf{V}}_i \cdot ((1-\cos\theta)\hat{\mathbf{T}}_i - \hat{\mathbf{T}}_{i+1}))\ (\cos\theta\ \hat{\mathbf{T}}_i + (1-\cos\theta)\ \hat{\mathbf{T}}_{i+1}) \\ &= -(\hat{\mathbf{V}}_i \cdot \hat{\mathbf{T}}_{i+1})\ (\cos\theta\ \hat{\mathbf{T}}_i + (1-\cos\theta)\ \hat{\mathbf{T}}_{i+1}) \\ &= -(\hat{\mathbf{V}}_i \cdot (\hat{\mathbf{T}}_i + \vec{\mathbf{T}}'\ \Delta s + O(\Delta s^2))\ (\cos\theta\ \hat{\mathbf{T}}_i + (1-\cos\theta)\ \hat{\mathbf{T}}_i \\ &\quad +(1-\cos\theta)\vec{\mathbf{T}}'\ \Delta s + O(\Delta s^2)) \\ &= -((\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\ \Delta s + O(\Delta s^2))\ (\hat{\mathbf{T}}_i + (1-\cos\theta)\ \vec{\mathbf{T}}'\ \Delta s + O(\Delta s^2)) \\ &= -((\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}')\ \Delta s + O(\Delta s^2))\ \hat{\mathbf{T}}_i - (1-\cos\theta)\ O(\Delta s^2), \\ \frac{\Delta\vec{\mathbf{V}}_i}{\Delta s} &= -(\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}' + O(\Delta s))\ \hat{\mathbf{T}}_i - (1-\cos\theta)\ O(\Delta s). \end{aligned} \tag{C.12}$$

But $1 - \cos\theta = 1 - \hat{\mathbf{T}}_i \cdot \hat{\mathbf{T}}_{i+1} = 1 - (\hat{\mathbf{T}}_i \cdot \hat{\mathbf{T}}_i + \hat{\mathbf{T}}_i \cdot \vec{\mathbf{T}}'\ \Delta s + O(\Delta s^2)) = -(\hat{\mathbf{T}}_i \cdot \vec{\mathbf{T}}')\ \Delta s + O(\Delta s^2) = O(\Delta s)$, so Eq. C.12 becomes:

$$\frac{\Delta\vec{\mathbf{V}}_i}{\Delta s} = -((\hat{\mathbf{V}}_i \cdot \vec{\mathbf{T}}') + O(\Delta s))\ \hat{\mathbf{T}}_i + O(\Delta s^2). \tag{C.13}$$

Therefore, as $\Delta s \to 0$, the component of $\frac{\Delta \vec{\mathbf{V}}_i}{\Delta s}$ that is not parallel to $\hat{\mathbf{T}}_i$ *quadratically* approaches zero, and thus is faster than the projection method.

The projection method also has other disadvantages, see Section 2.2.2.

Appendix D

Tangent Spaces and Geodesics

For every point p on a surface M in R^n, let T_pM be the *tangent plane* of M at p. The disjoint union of all the tangent planes of M forms the *tangent bundle* TM of M. The induced *Riemannian metric* (from the ambient space R^3) on M is called the *first fundamental form* of M, and is denoted by ds^2. ds^2 is used to measure lengths on M. One can also define the *second fundamental form* of M, denoted by II, which describes the shape of M in R^3. II allows a quantitative study of the shape of M in the neighborhood of a point. It measures how far M is from being a plane: If II is identically zero, M is a plane, and vice versa. It is also used in calculating the curvatures of curves on M. Combining ds^2 and II, we can compute the two *principal curvatures* and the *mean curvature* of M. Another important invariant of M is the *Gaussian curvature* (also called *total curvature*). The Gaussian curvature turns out to be determined by the first fundamental form ds^2.

Similar to the definition of a parallel normal vector field on a curve, a tangent vector field on M is said to be *parallel* along a curve on M if its derivative along the

curve is everywhere perpendicular to M. And a curve on M is called a *geodesic* if the tangent vector field determined by the curve itself is parallel along the curve itself. The shortest path on M that connects two points on M is a geodesic. Some geodesic paths on 3D and 4D surfaces generated by the algorithms in Chapter 4 are shown in Color Plates G.16, G.17 and G.18.

For a surface in a higher dimensional space, we can still define the tangent bundle, the first fundamental form, the second fundamental form, the geodesics, mean curvature and Gaussian curvature (everything mentioned above except the principal curvatures). But the curvatures are no longer scalar values, they are vectors that are perpendicular to the surface. The mathematical theory of surfaces is far too rich to fit in this article, even very briefly. There are many good books on this subject, [9] is an example.

For higher dimensional manifolds, most of these concepts still exist, especially the tangent bundle and the geodesics.

The above is a very brief introduction to the theory of differential geometry. For more information, read [23] if you like figures and Mathematica, or [58] for the most comprehensive introduction.

Appendix E

Regge Calculus

Consider the dome that covers a big auditorium, made of many flat triangles joined edge to edge and vertex to vertex. Similarly in the Regge calculus, manifolds are made of flat *simplexes* (two dimensions, triangle; three dimensions, tetrahedron; four dimensions, 4-simplex and so on) joined face to face, edge to edge, and vertex to vertex. To specify the lengths of the edges is to give the engineer all he needs in order to know the shape of the roof, and the scientist all he needs in order to know the geometry of the manifold under consideration. A smooth auditorium roof can be approximated arbitrarily closely by a dome constructed of sufficiently small triangles. A smooth manifold can be approximated arbitrarily closely by a locked-together assembly of sufficiently small simplexes.

Figure E.1 shows how a smoothly curved surface can be approximated by flat triangles. For the smooth surface on the left, the curvature exists everywhere on the surface and varies continuously. But after tessellation (on the right), all the curvature is concentrated at the vertices. No curvature resides on the triangle faces or the edges.

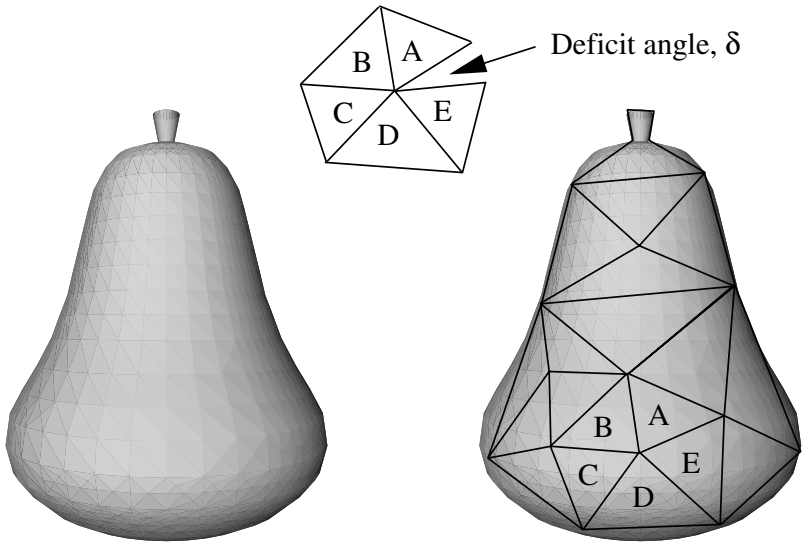


Figure E.1: A surface with continuously varying curvature can be approximated arbitrarily closely by a polyhedron built of triangles. The curvature of the surface shows up in the amount of deficit angle at each vertex (portion $ABCDE$ of the polyhedron laid out above on a flat surface).

A vector carried by parallel transport along a closed route that doesn't encircle any vertices (e.g, A to B to C and back to B to A) will remain unchanged in direction, as one sees most easily by laying out this complex of triangles on a flat surface. Only if the route encircles the vertex common to A, B, C, D and E does the vector experience a net rotation. The magnitude of the rotation is equal to the indicated deficit angle, δ, at the vertex. The sum of the deficit angles over all the vertices has the same value, 4π (2π $\times$ the Euler characteristic of the surface), as does the integral of the continuously distributed curvature taken over the original smooth figure.

Generalizing from two dimensional surfaces, the Regge calculus approximates a smoothly curved n-dimensional manifold as a collection of n-dimensional blocks, each free of any curvature at all, joined by $(n-2)$-dimensional regions in which all the curvature is concentrated. So as an example, for the four-dimensional spacetime of

general relativity, where it was originally invented, the "hinge" at which the curvature is concentrated has the shape of a triangle. See [51] for Regge's original paper.

Appendix F

Grassmann and Stiefel Manifolds

The *Grassmann manifold* $G_k(R^n)$ is the space of all k-planes containing 0 in R^n $(0 < k < n)$. The Lie group $O(n)$ (all the $n \times n$ othorgonal matrices) acts transitively on $G_k(R^n)$, and the action gives $G_k(R^n)$ a differentiable structure:

$$G_k(R^n) \cong O(n)/(O(k) \times O(n-k)),$$

with dimension $k(n-k)$.

The *Stiefel manifold* $V_k(R^n)$ is the space of all orthonormal k-frames in R^n. The Lie group $O(n)$ also acts transitively on $V_k(R^n)$, but the action results in a different differentiable structure:

$$V_k(R^n) \cong O(n)/O(n-k),$$

with dimension $\frac{n(n-1)}{2} - \frac{(n-k)(n-k-1)}{2}$.

In computer graphics, Grassmann manifolds can be used in shading calculations. Because in shading, we need to know where the plane lies. But we don't care about

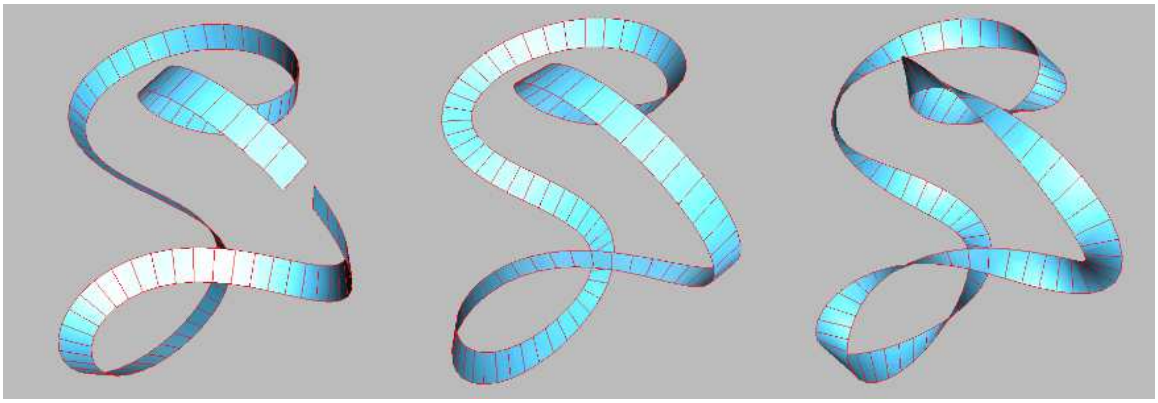
rotations within the plane. For smooth shading, we need to find the approximate plane, which may be obtained by averaging or interpolating among existing planes. The popular 3D *Phong shading* and *Phong normal interpolation* methods can be viewed as interpolation methods over the special Grassmann manifold, $G_2(R^3)$ (tangent planes in 3D, or equivalently $G_1(R^3)$, normal vectors in 3D), which is the same as (diffeomorphic to) S^2, the ordinary sphere.

At the mean time, Stiefel manifolds can be used in camera controls. Because when we control a camera, we need to know the complete orientation of the camera in the viewing space. The special case, $V_3(R^3)$ ($\cong SO(3) \cong RP3$) for 3D camera frames, has been studied by Shoemake ([56]). Shoemake uses unit quaternions, which is the universal covering space of $RP3$, to interpolate camera frames and achieve smooth animations of rotations.

Generalization of these applications to $n > 3$ requires more understanding of the metrics on Grassmann and Stiefel manifolds, and Lie groups and Lie algebras. [13] is a good geometric introduction to Lie groups.

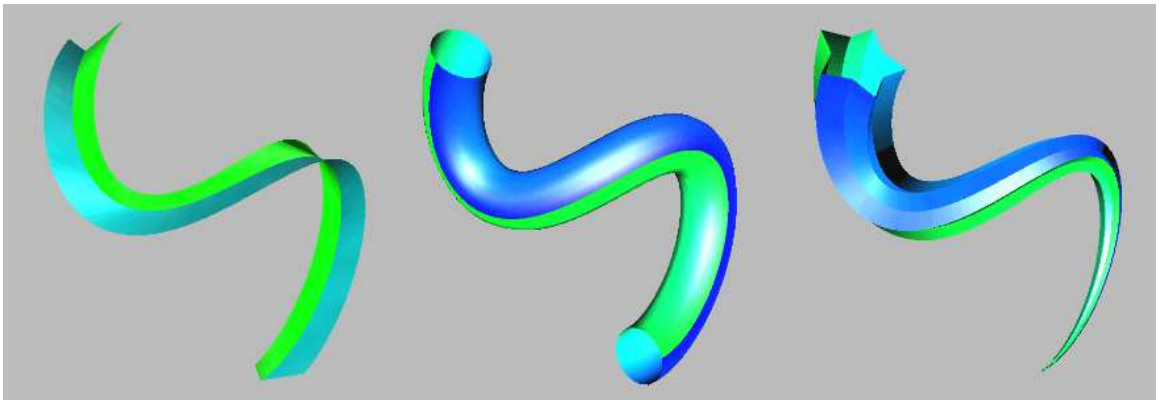
Appendix G

Color Plates



(a) (b) (c)

Figure G.1: (Section 2.2.3) Parallel transport on closed curves. (a) A ribbon produced by parallel transport of an initial vector does not generally return to the same orientation after one circuit. (b) Closure can be enforced by distributing the angular deficit around the curve. (c) Adding multiples of π to the total axial rotation gives any desired amount of twisting (this example adds 6π).



(a) (b) (c)

Figure G.2: (Section 2.2.4) Creating ribbons and tubes using parallel transport. (a) A ribbon generated by a pair of vectors. (b) A tube with a circular cross-section. (c) A tube with a star-shaped cross-section and a varying radius.

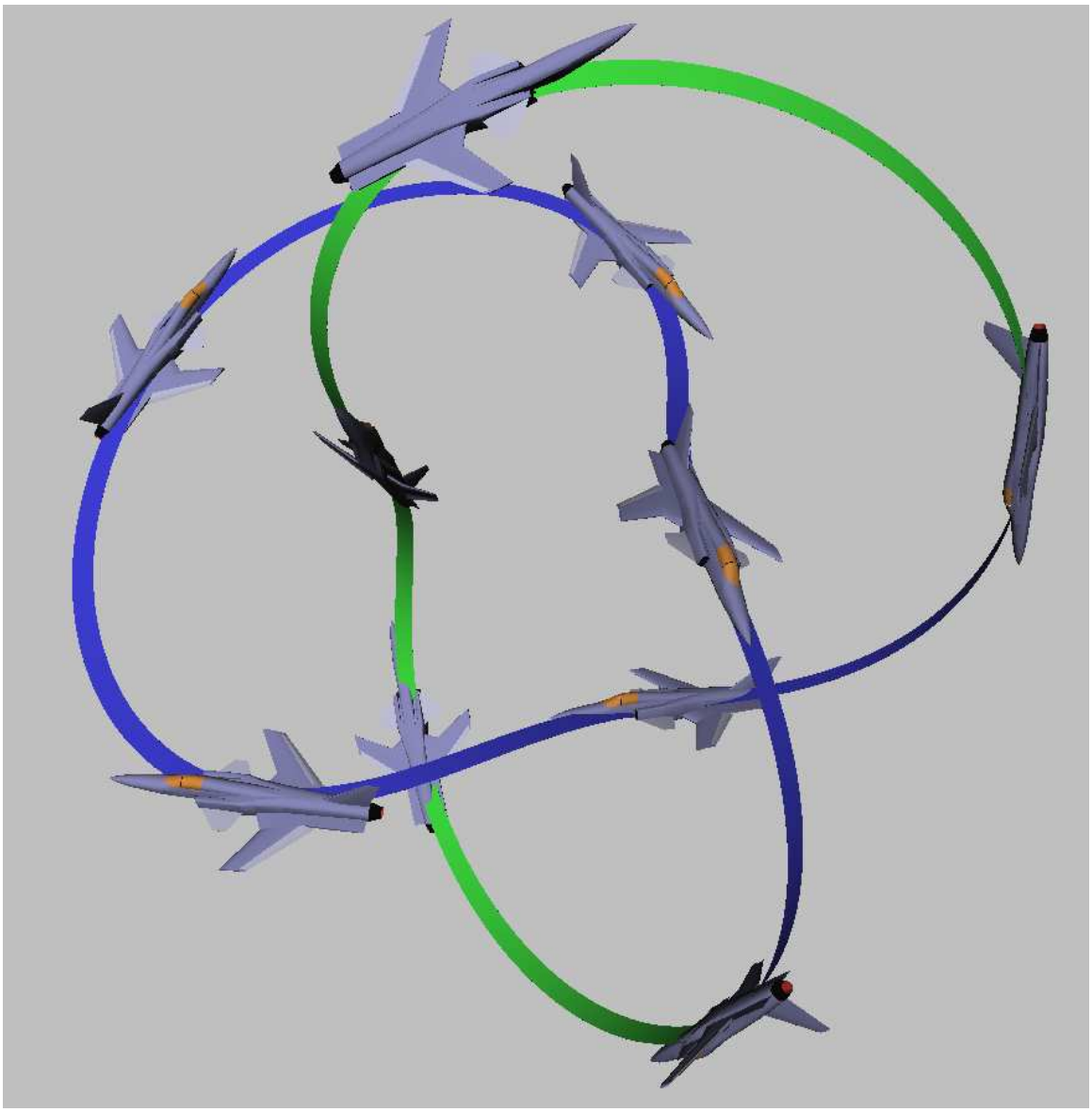
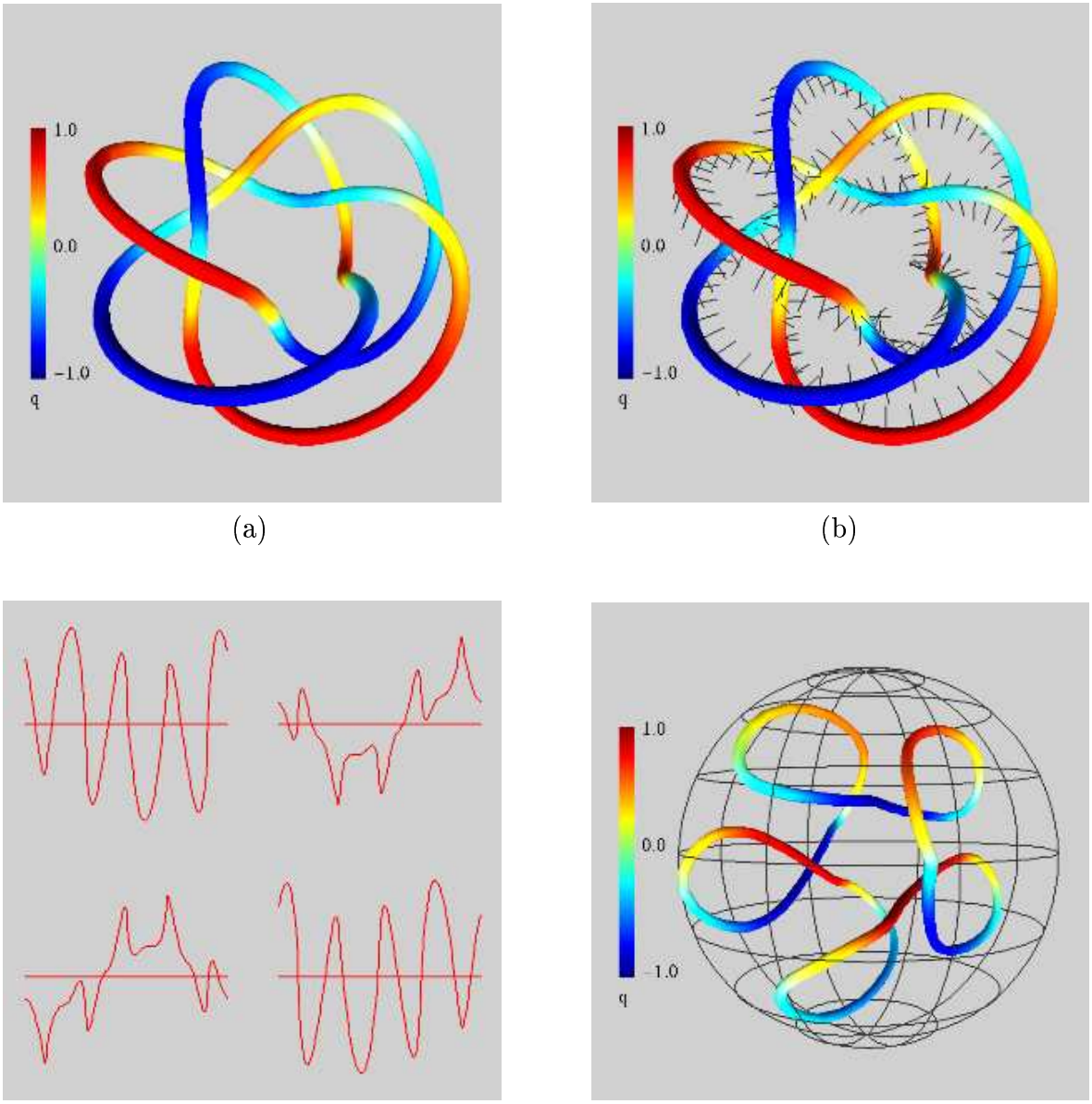


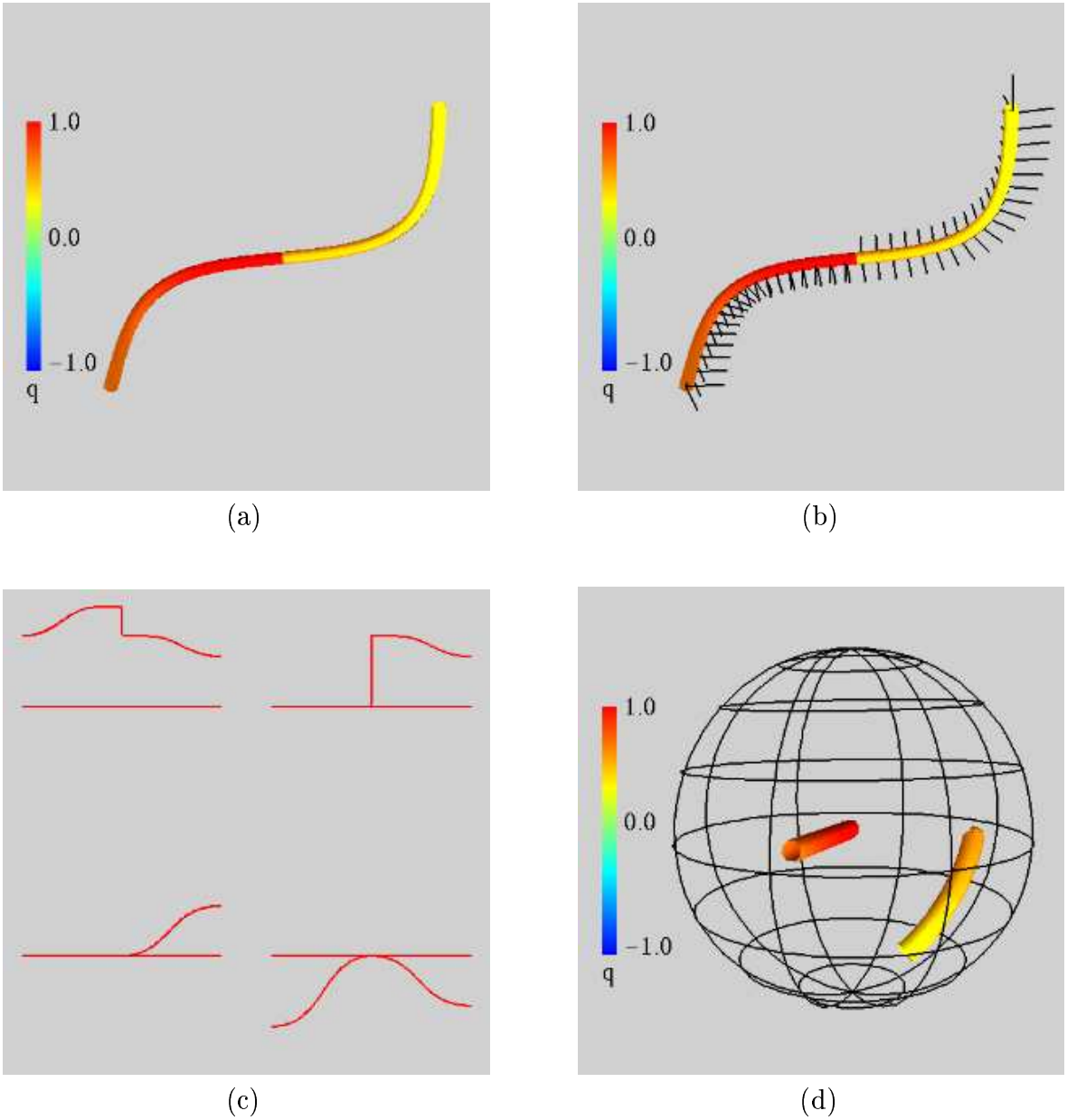
Figure G.3: (Section 2.2.5) An application of the parallel transport frame to the generation of a moving camera orientation automatically determined by the geometry of the flight path itself. In this illustration, the orientation of the aircraft represents the camera orientation. When the ribbon appears blue, the top of the aircraft is facing us, while when the ribbon is green, we are looking at the bottom of the aircraft.



(a) (b)

(c) (d)

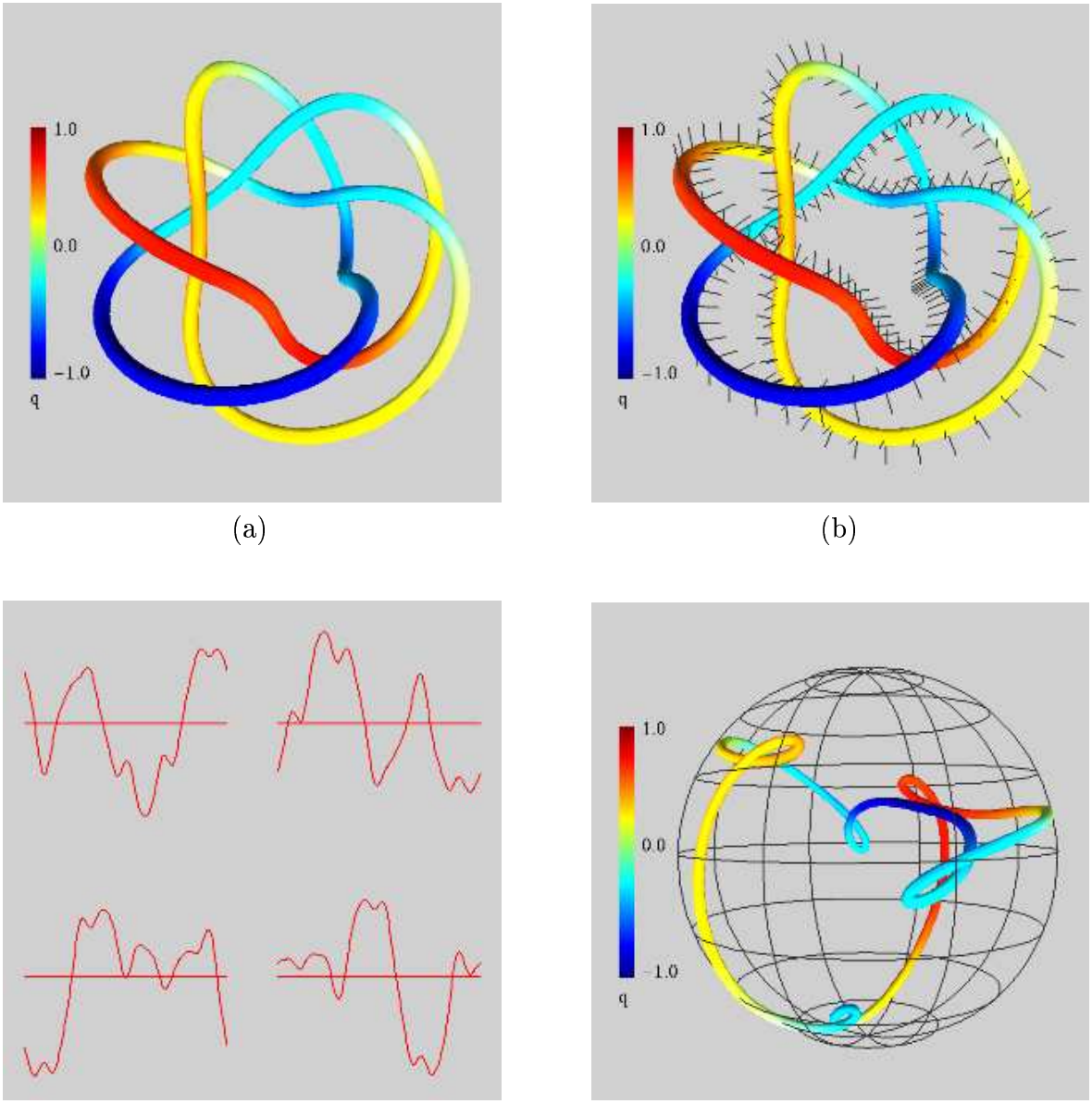
Figure G.4: (Sections 3.2.1, 3.2.2 and 3.4.1) Quaternion-Frenet frame on a 3D torus knot. (a) Projected image of a 3D (3,5) torus knot. (b) Selected Frenet frame components displayed along the knot. (c) The corresponding smooth quaternion frame components. (d) The path of the quaternion frame components in the three-sphere projected from four-space. Color scales indicate the 0-th component of the curve's four-vector frame (upper left graph in (c)).



(a) (b)

(c) (d)

Figure G.5: (Sections 3.2.1, 3.2.2 and 3.4.1) Quaternion-Frenet frame on a pathological curve segment. (a) Projected image of a pathological curve segment. (b) Selected Frenet frame components, showing a sudden change of the normal. (c) The quaternion frame components, showing discontinuity in values. (d) The discontinuous path of the quaternion frame components in the three-sphere. Color scales indicate the 0-th component of the curve's four-vector frame (upper left graph in (c)).



(a) (b)

(c) (d)

Figure G.6: (Sections 3.2.2 and 3.4.1) Quaternion-Parallel-transport frame on a 3D torus knot. (a) Projected image of a 3D (3,5) torus knot. (b) Selected parallel transport frame components displayed along knot. (c) The corresponding smooth quaternion frame components. (d) The path of the quaternion frame components in the three-sphere projected from four-space. Color scales indicate the 0-th component of the curve's four-vector frame (upper left graph in (c)).

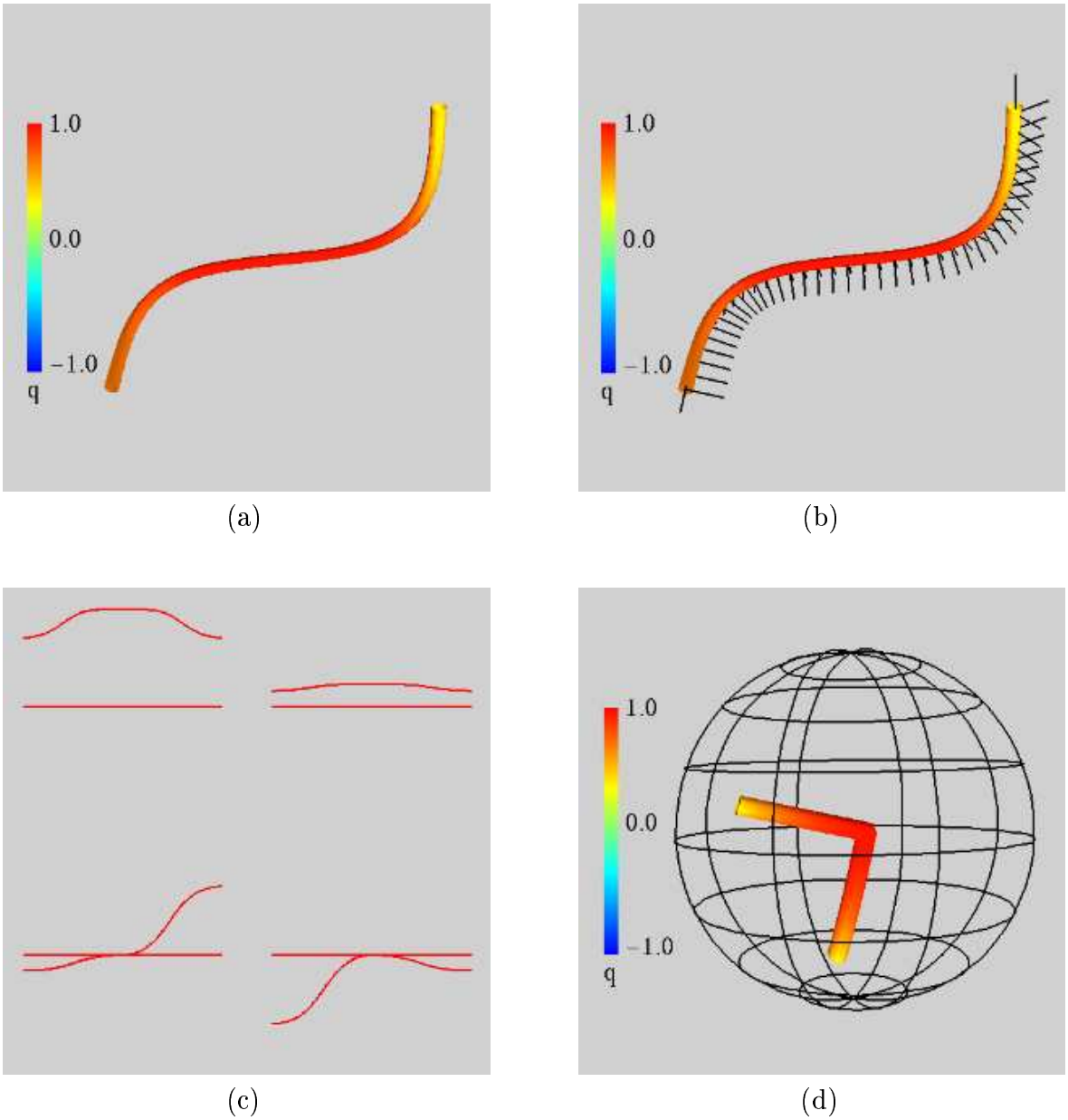
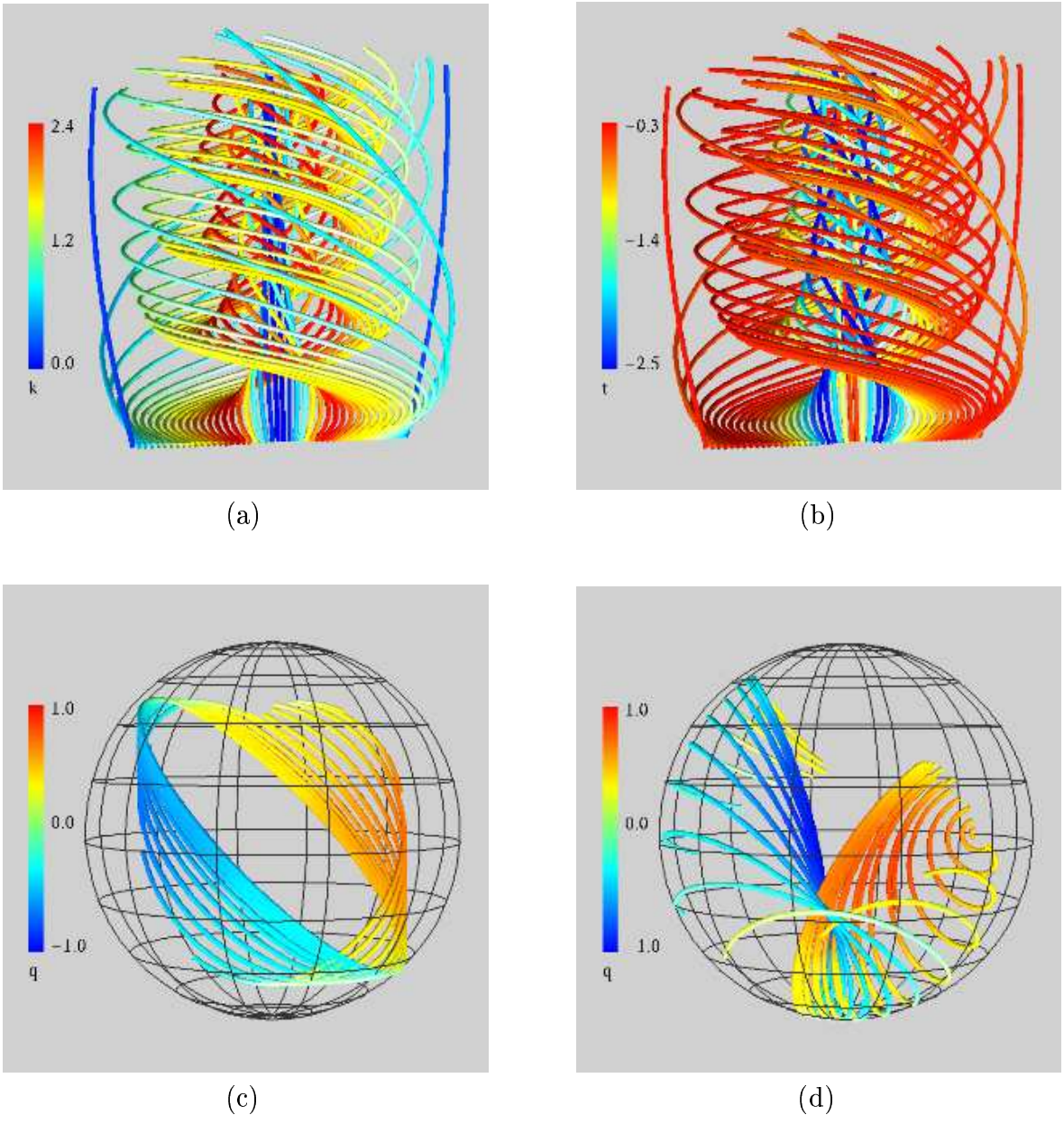


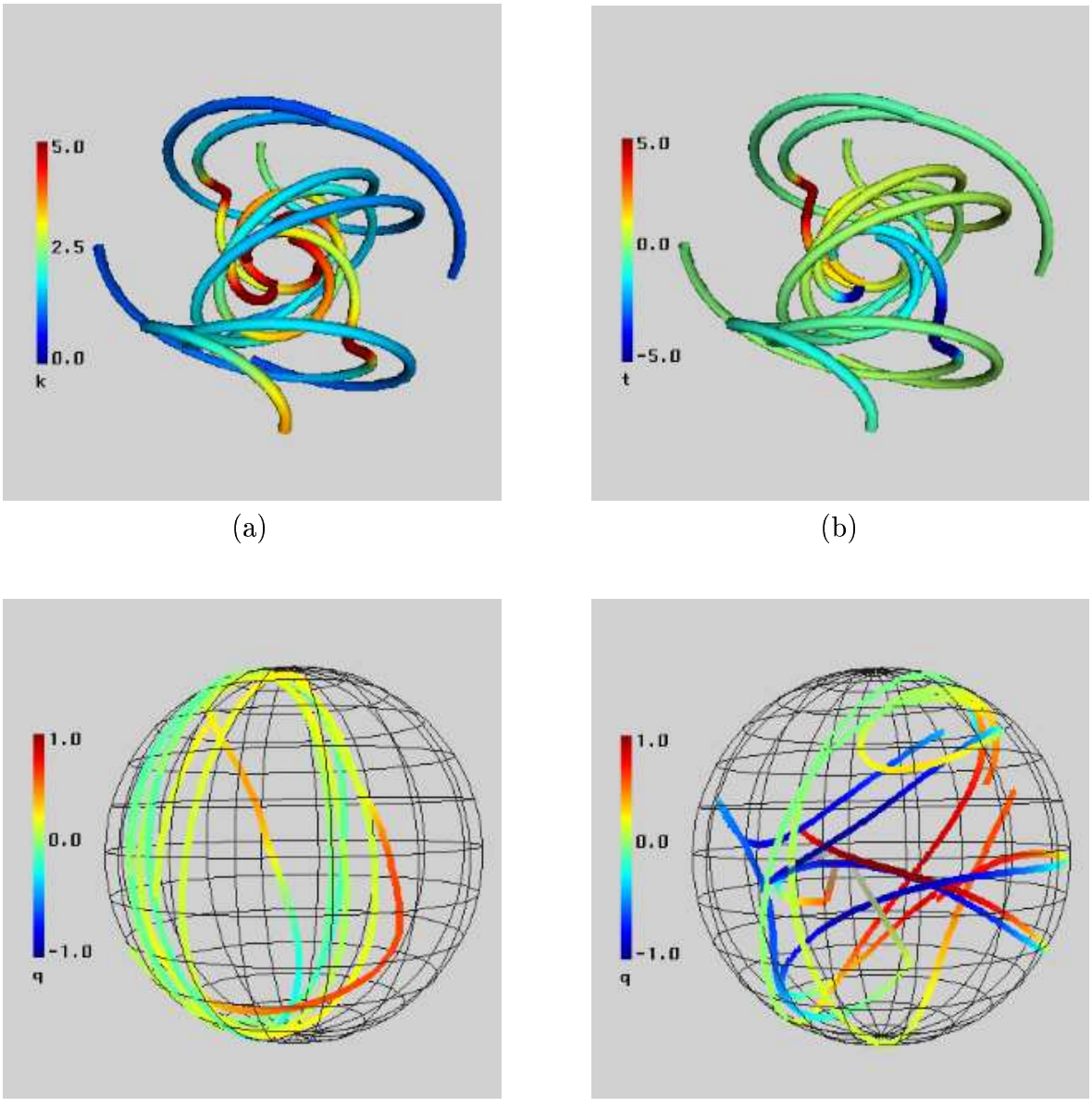
Figure G.7: (Sections 3.2.2 and 3.4.1) Quaternion-Parallel-transport frame on a pathological curve segment. (a) Projected image of a pathological curve segment. (b) Selected parallel transport frame components, showing smooth change of the normal. (c) The quaternion frame components, showing continuity in values. (d) The continuous path of the quaternion frame components in the three-sphere. Color scales indicate the 0-th component of the curve's four-vector frame (upper left graph in (c)).



(a) (b)

(c) (d)

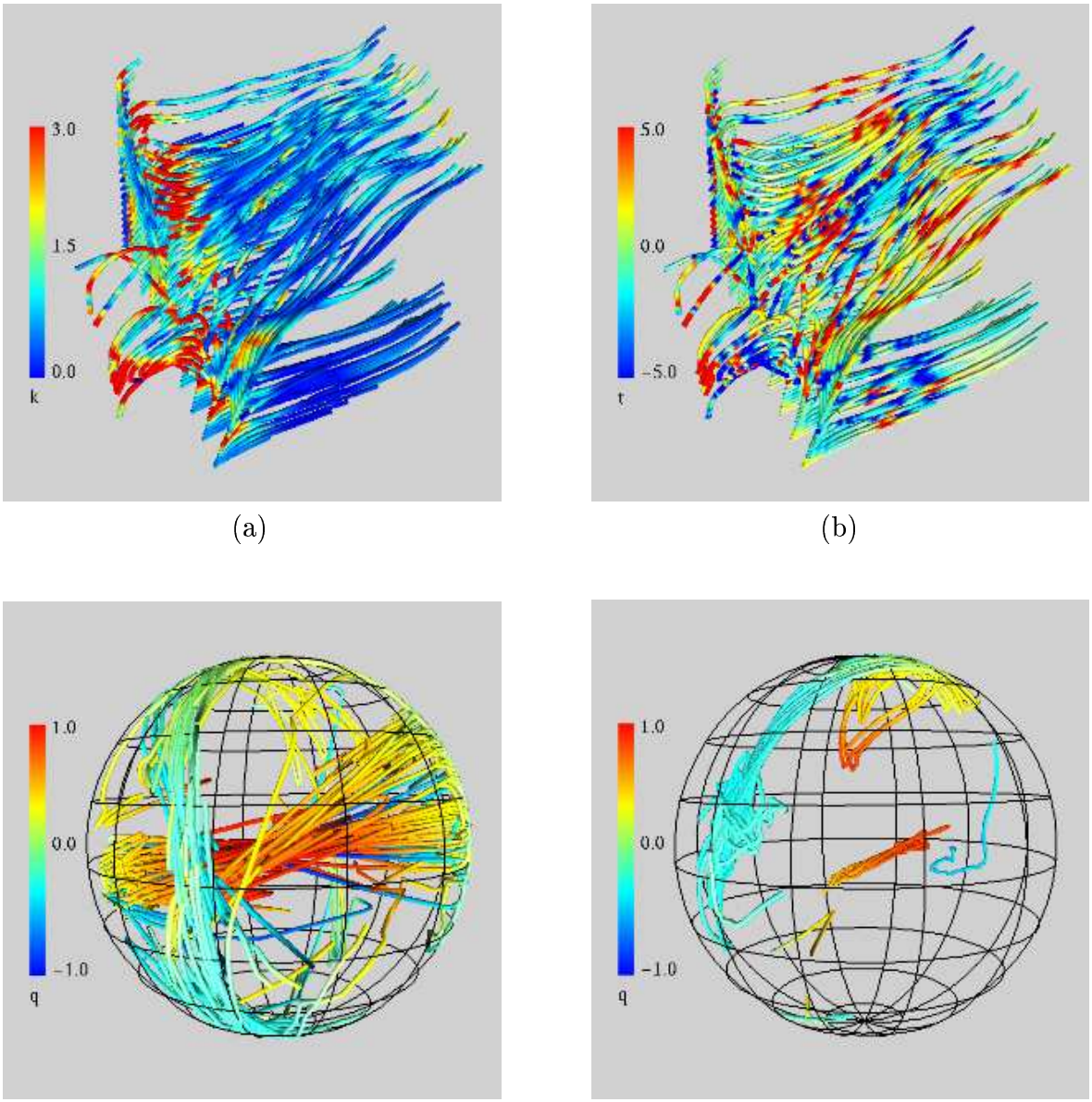
Figure G.8: (Sections 3.3, 3.4.1 and 3.4.2) Quaternion frames on curves related with tying a knot. (a) Deformed volume related to tying a knot, color coded by curvature. (b) Deformed volume related to tying a knot, color coded by torsion. (c) The corresponding quaternion field paths for the Frenet frames. (d) The corresponding quaternion field paths for the parallel transport frames. The color code is keyed to the value of the quaternion component q_0 that is collapsed in the projection from 4D to 3D.



(a) (b)

(c) (d)

Figure G.9: (Sections 3.3, 3.4.1 and 3.4.2) Quaternion frames on Dirac strings. (a) Dirac strings color coded by curvature. (b) Dirac strings color coded by torsion. (c) The corresponding quaternion field paths for the Frenet frames. (d) The corresponding quaternion field paths for the parallel transport frames. The color code is keyed to the value of the quaternion component q_0 that is collapsed in the projection from 4D to 3D.



(a) (b)

(c) (d)

Figure G.10: (Sections 3.3, 3.4.1 and 3.4.2) Quaternion frames on vector field streamlines. (a) Vector field streamlines, color coded by curvature. (b) Vector field streamlines, color coded by torsion. (c) The corresponding quaternion field paths for the Frenet frames. (d) The corresponding quaternion field paths for the parallel transport frames. The color code is keyed to the value of the quaternion component q_0 that is collapsed in the projection from 4D to 3D. (AVS data set.)