Applications of Ribbed Surfaces

James Hamlin

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-123
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-123.html

August 23, 2009

Acknowledgement

Prof. Carlo H. Sequin and Prof. Ruzena Bajcsy served as advisers on this work and as readers of this report.

Applications of Ribbed Surfaces

by James F. Hamlin

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Carlo Séquin
Research Advisor

(Date)

* * * * * * *

Professor Ruzena Bajcsy
Second Reader

(Date)

Applications of Ribbed Surfaces

by

James F. Hamlin

Abstract

Applications of Ribbed Surfaces

by

James F. Hamlin
Master of Science in Computer Science

University of California at Berkeley

Professor Carlo Séquin, Research Advisor

This paper describes various implementations of ribbed surfaces, a versatile generalization of the well-known ruled surfaces. In the case of ruled surfaces, the "ribs" would be straight line segments; in the case of ribbed surfaces, we allow the ribs to be more general curves, which have their end points located on two different segments of the rail curve(s). Several applications of ribbed surfaces are developed and presented, demonstrating the versatility of ribbed surfaces as a design element. Inspired by the ribbed sculptures of Charles O. Perry, two of these applications begin as emulations of his sculptures *Solstice* and *Early Mace*. Extension of these emulation programs then leads to distinct families of ribbed sculptures, each characterized by a small set of parameters. Other applications present the application of ribbed surfaces in the domains of architecture and mathematical visualization. All of these applications concretely demonstrate the design choices one must make when using ribbed surfaces, including guide rail specification, the type of curve used to represent the ribs, and the parameterization of the ribs in terms of their end conditions along the guide rails.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

This work is an investigation of applications of a class of surfaces generalized from the well-known ruled surfaces. Just as a ruled surface may be defined by sweeping a line while transforming it, these surfaces, called "ribbed surfaces," may be formed by sweeping a curve segment, forming the "ribs," along two (possibly identical) "guide rail" curves. The curves are parameterized by functions of the properties of the guide rail curves at each end-point, and so they are not necessarily uniform. Their parametric description allows for the relatively simple specification of a versatile and heterogeneous set of structures, and their visual properties make them suitable for the visualization of highly self-intersecting surfaces and for the realization of large, yet relatively inexpensive sculptures [3].

We explore the space of tools and applications for describing such ribbed surfaces, from several rigidly circumscribed applications to general-purpose tools. The focus is on user-interface design: in particular, identification of the parameters to expose to a user of a system for designing such surfaces. We have found that the parameterization exposed to the user will depend largely on the application, thus we proceed by examining several well-circumscribed domains of surfaces and structures before moving on to applications for the design of ribbed surfaces using the fully general parameterization.

It is hoped that these tools may serve to provide design tools for architecture and visualization, as well as for designing and prototyping ribbed sculptures.

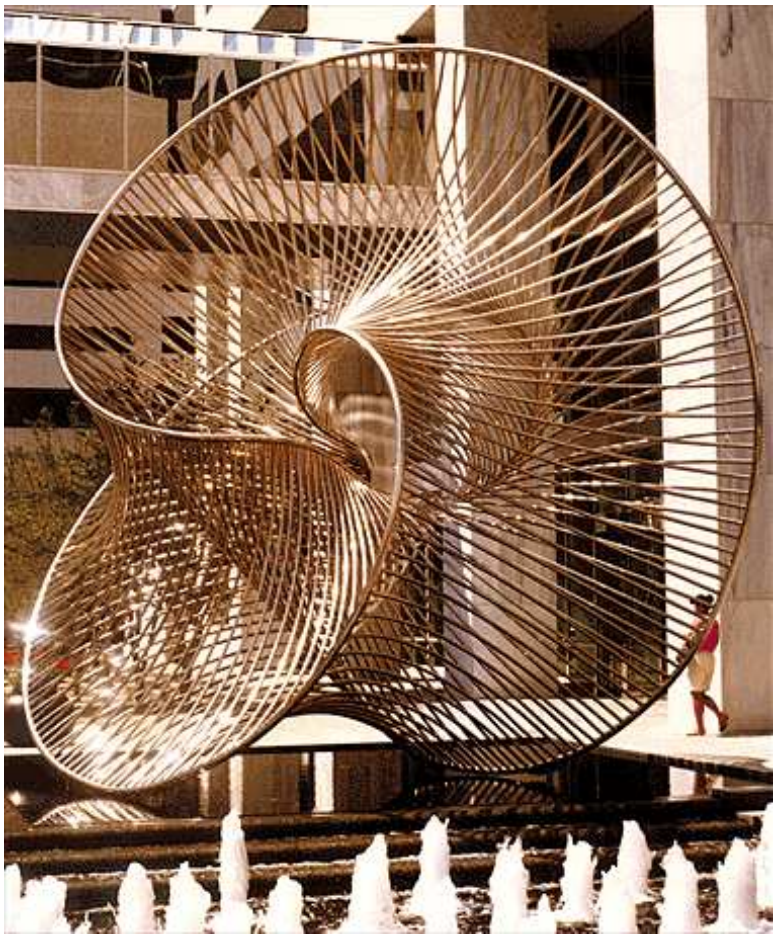


Figure 1: Charles O. Perry's ribbed sculpture *Solstice*, installed in Tampa Bay, Florida.

1.2 Ribbed Surfaces versus Sweep Surfaces

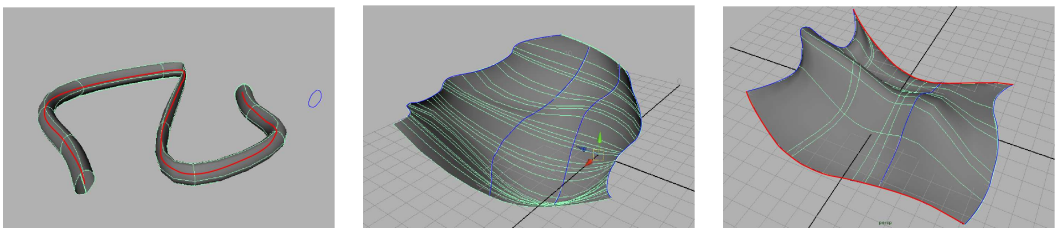
The concept of "ribbed surfaces" emerged from an attempt to reverse-engineer some of the tubular ribbed sculptures created by Charles Perry, beginning with Perry's work *Solstice*, located in downtown Tampa, Florida (Figure 1) [4]. *Solstice* has a thick supporting rail in the shape of a (3,2) torus knot. The three segments of this dominant curve, which pass through any radial section of the embedding torus, are connected by curved "ribs" that roughly form a hyperbolic triangle. This triangle rotates through 240 degrees as it sweeps once around the major loop of the torus. Once this construction principle is understood, it seems natural to try to change some of the key parameters so as to accommodate other types of (p,q) torus knots and create different ribbed surfaces between different segments of the dominant rail forming that knot. *Solstice*'s role in motivating the development of ribbed surfaces is spelled out in more detail in the following chapter. It suffices to say here that it was in exploring the variations of Perry's *Solstice* that the idea of further abstracting such tubular structures emerged.

Ribbed surfaces bear some similarity to a long- and well-established class of surfaces, *sweep surfaces*. Before proceeding further, I will here briefly taxonomize these surfaces and distinguish ribbed surfaces from the common types of sweep surfaces, which are available to designers in popular 3D modeling software packages such as Maya [5].

In their most basic form, sweep surfaces are obtained by sweeping a single curve, called the *profile*, along a path curve. The surface is then defined as the set of points through which the profile passes as it is swept along the path. I will call this class of sweep surfaces *simple sweeps* (an example is shown in Figure 2(a)).

Another member of the family of sweep surfaces is *skinned surfaces* (Figure 2(b)). Skinned surfaces are obtained from a sequence of two or more cross-section curves, called *profile* curves, by interpolating between each adjacent pair of cross-sections. The method of interpolation will determine the particular surface, but this parameter is not important here. Unlike simple sweeps, skinned surfaces do not require a path curve, as the positions and orientations of the profile curves themselves determine the path a sweep would take through space. In Maya, these surfaces are called *lofts*.

A variant of these skinned surfaces requires both a parameterized path curve and two or more profile curves. Each profile curve is associated with a value of the path curve's parameter, and the sweep cross-section is again interpolated between these parameter values. These skinned surfaces differ from the profile-only sort in that the sweep follows the path in-between profile curves rather than being subject to the method



(a) (b) (c)

Figure 2: Three families of sweep surfaces: (a) simple sweep, (b) skinned surface, and (c) birail sweep. Profiles are highlighted in light blue, rails or paths in dark red.

of interpolation used for the latter sort. This variant is actually a super-class of simple sweeps - with identical profile curves at either extreme of the path parameter interval, one produces a simple sweep. These are available the "sweepmorph" modeling element in Berkeley SLIDE [7], a rendering system created by Jordan Smith.

A third category of sweep surfaces (Figure 2(c)) is an extension of this variant of skinned surfaces. Instead of using a single path curve, this class of sweeps uses two "rail" curves and one or more profile curves. The profile curves are then scaled and oriented such that each end-point lies on one of the rail curves. As in the second type of skinned surface, the cross-section is interpolated between the rail parameter values at which the cross-section is explicitly defined by a profile curve. In Maya, these surfaces are called *birail* sweeps.

I will now distinguish ribbed surfaces from each of these common types of sweeps. There is of course a trivial difference that I will sidestep. That is, ribbed surfaces, as we use them in the applications described in this report, are not continuous surfaces as the above sweep surfaces are, but are instead collections of discrete "ribs" that suggest such a surface. To make the comparison simpler, I will consider the differences between our ribbed surfaces and the class of ribbed surfaces one could generate from each of these other sweep surface schemes. One can generate a ribbed surface from one of these sweep surfaces by taking n uniformly-spaced isoparametric curves, curves along the surface for a constant value of the sweep parameter. Each of these curves will serve as a path curve for each of n simple sweeps (the profile curve used is unimportant). Each of these simple sweeps is a rib in the ribbed surface, and simple sweeps of the guide rails (in the birail case) or some interpolating curve of the rib end-points (in the simple sweep and skinned surface cases). An example demonstrating how this procedure yields a ribbed surface is shown in Figure 3.

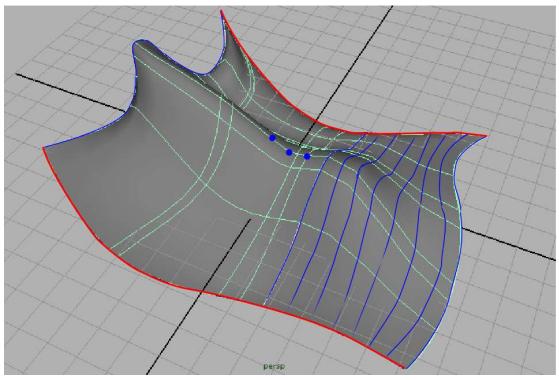


Figure 3: To produce a ribbed surface from such a sweep surface, the curves highlighted in dark red become the "rails" and the curves highlighted in light blue become the "ribs." The ribs follow the isoparametric curves of the sweep surface, evenly spaced in the unit parameter interval.

First, ribbed surfaces are clearly not reducible to mere simple sweeps, which use a single path curve. For the skinned surfaces with a path, the guide rails are determined by the single input path, which is insufficient for capturing the variation allowed by the dual guide rails of ribbed surfaces. Finally, birail sweep surfaces do allow the specification of two distinct guide rails, but they require the explicit input of a discrete number of profile curves whose shapes are then interpolated to generate intermediate ribs. Our general ribbed surfaces, on the other hand, allow for the procedural generation of entirely unique ribs at each parameter value. Furthermore, parameters to the generator functions can be explicitly set and then interpolated, adding a level of abstraction above the birails. So we see that taking some number of isoparametric curves of a skinned or birail surface and curves interpolating cross-section end-points, the result is effectively a ribbed surface, but even so, skinned and birail surfaces should be understood as sub-classes of ribbed surfaces. Thus, ribbed surfaces are not in general reducible to any of these members of the family of swept surfaces. Nor are they reducible to a transformation that produces ribs from the isoparametric curves of such surfaces.

1.3 Overview

The remainder of this text proceeds as follows. Chapter 2 considers a motivating application, the emulation and extension of Charles O. Perry's sculpture *Solstice*, stepping through the decisions one must make when designing a ribbed surface or sculpture. Chapter 3 proceeds to flesh out in detail the various implementations of

ribbed surfaces, including particular rib representations and parameterizations. Chapter 4 presents several applications and their implementations, ranging from sculpture design to mathematical visualization, that have been implemented using ribbed surfaces as their central feature. Finally, Chapter 5 closes the report with a discussion of the high-level benefits of ribbed surfaces as a design element and further discussion of their utility.

Chapter 2

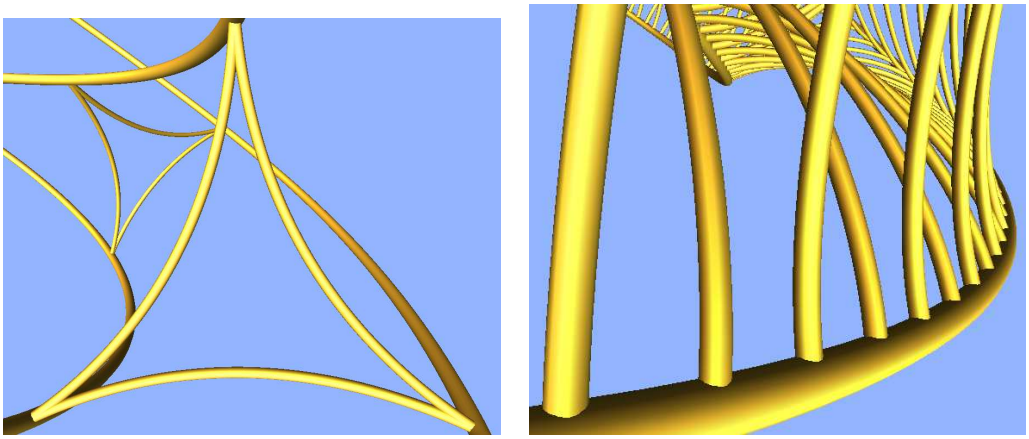
Motivating Application

The introduction of ribbed surfaces drew its inspiration from several works of the sculptor Charles Perry. In this section I describe the generalization of a sculptural paradigm from one of Perry's works, *Solstice*, resulting in a parametric description that is easily translated into a design application atop a system for creating general ribbed surfaces. The decisions that led to the user-exposed parameterization and its realization in ribbed surfaces are then discussed. In a later chapter the implementation of the system is described in detail.

2.1 Solstice Paradigm

A sculpture may be characterized as a ribbed sculpture by identifying the following components: the guide rails, the ribs, the 'mapping' between the guide rails performed by the ribs, and the rib curve parameterization. For *Solstice*, the guide rail is the thicker metal tube, which is topologically a (3, 2) torus knot. One can see that given any slice of the arm of the (imagined) torus, the rail passes through it three times. In such a slice, three ribs connect the three rails so that a "hyperbolic" triangle is formed (Figure 4(a)). For each rib, one end-point is 1/3 ahead of the other along the guide rail (since each third of the guide rail in the curve's parameter space makes a complete revolution around the major circle of the torus). Finally, the ribs are bent so that the triangular shape implicitly enclosed is concave.

This characterization of the ribs is not quite correct; no more than one rib end-point lands at a given point along the guide rail, so the offset between rib end-points is not quite 1/3, but slightly more or slight less so that the rib end-points are staggered along the guide rail (Figure 4(b)).



(a) (b)

Figure 4: Rib endpoints along rails in *Solstice*. (a) shows the "triangular" rib configuration in cross-sectional view, (b) shows the staggering of ribs along the guide rail.

The parameters that characterize *Solstice*-like sculptures are thus p and q, where the guide rail is a (p, q) torus knot, the offset to produce the desired staggering of the ribs, and the shape of the ribs, in particular their concavity (Figure 4(a)).

2.2 From Paradigm Parameterization to Program

We have identified a clear set of parameters that yield a family of sculptures in the style of Perry's *Solstice*. Here I describe the process of taking such a parameterization and describing these parameters in terms of ribbed surfaces. Several questions must be answered:

1. How should the guide rails be specified?

   In the procedural domain, this question reduces to what class of parameterized curves, perhaps of a certain mathematical species, should be used. The parameters required to specify this curve will follow from this choice. The rail curve of *Solstice* is a (3, 2) torus knot, thus it seems natural to allow a designer to pick any particular (p, q) torus knot.

2. How should rib end-points be placed along the guide rails?

   That is, which intervals along each guide rail have ribs between them? For *Solstice*, the "mapping" performed by the ribs is determined almost entirely by the

p parameter of the (p, q) torus knot. But as discussed above, we must allow for an offset to allow for the staggering displayed in the original sculpture.

3. What class of curves should be used for the ribs?

   In other words, what sort of shape must the ribs have? This choice will determine how exactly the end conditions, which vary as the ribs are swept along the rail(s), determine the shape of the ribs. We have found cubic Hermite curves to be very suitable for a host of applications, but many common applications might use higher-degree Hermite curves, circular arcs, or b-spline curves, each parameterized by the properties of the guide rails at the rib end-points. For *Solstice*, the cubic Hermite curve representation suffices.

4. How should ribs be parameterized in terms of their end-points?

   To answer this question, one must identify the desired relationship between rib shape and the rib's position in the sweep along the guide rails. Whether or not the ribs are planar, symmetric, circular, etc. are all determined when answering this question. For *Solstice*, the rib's shape will depend on where the end-points fall relative to the overall toroidal shape of the structure. The ribs are planar and symmetric and lie nearly in the planes of minor circles of the torus.

Chapter 3

Implementing Ribbed Surfaces

In this chapter I describe the various implementations of ribbed surfaces, including alternative parameterizations and how the parameters are used to generate the surface. As a procedural geometric element, ribbed surfaces allow for a wide variety of concrete instances to be generated with simple and straightforward changes to the parameters. Procedural geometry generation allows one to compose encapsulated procedural units to produce classes of compound geometric structures with few parameters. The procedural character of ribbed surfaces is exploited in the example applications presented in the next chapter. The challenges of procedural geometry generation mirror those of the design of object-oriented systems. One must select the proper interfaces for the different components, balancing cleanliness of abstraction with expressive power. Thus the design of procedural geometry systems can adopt principles of object-oriented design. This, combined with the particular requirements of these systems, suggests that one ought to construct fine-grained components that can then be composed in various intricate ways to produce new procedural geometric primitives. If the compound components then expose the same simple interfaces of the atomic components, then they may be recombined to produce further new primitives. The applications for ribbed surfaces described here take this approach. Every level of abstraction is implemented as a procedural composition of simpler procedural elements. With the basic ribbed surface component built, which is a composition of sweeps given rib and rail curves as input, we can in turn generate the rail curves and rib parameters procedurally. This terminates with a minimal set of parameters exposed to a user/designer.

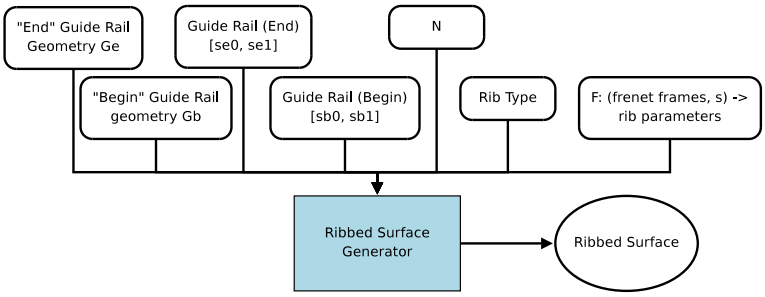


Figure 5: Abstract diagram of ribbed surface module

3.1 Parameterization Overview

In order to create a ribbed surface, we need to define two parameterized guide rail curves (which may be identical) and the rail cross-sections that will be swept along the rail curves. For the ribs we have to specify: the total number of ribs; their cross-section information; two parameter intervals [sb0, sb1] and [se0, se1], for the beginning (b) and end (e) point locations on the appropriate guide rails; an application-dependent set of geometric end-point parameter functions that define the shape of the individual ribs as the sweep parameter s is running through the interval from 0 to 1. These parameters control the curve parameters in various ways depending on the type of rib curves used. The specified number of ribs is then generated and uniformly positioned over the domain of s. Some extra higher-level control functions make it easy for the user to create sets of rib end points that coincide or, alternatively, are evenly interleaved when multiple sets of ribs are ending on the same rail. Different shape parameters are made available depending on the application; these issues are discussed below.

To put it abstractly, suppose we have guide rails and parameter intervals for curves $Gb(sb)$ and $Ge(se)$: $[sb0, sb1]$, $[se0, se1]$; the number of ribs N; rib type T, a function mapping from the parameters for that rib type (e.g. cubic Hermite) to the rib curve; and rib parameter function F, a function of the Frenet frames for guide rail curves at the rib end-points. To produce the nth rib (at sweep parameter value $s = n/(N-1)$, we create a simple sweep along the curve:

$$T(F(frenet(Gb(sb0 + s * (sb1 - sb0))), frenet(Ge(se0 + s * (se1 - se0))), s))$$

3.2 Rib Parameterization

The guide rails are explicitly defined and provided as inputs to the ribbed surface generator, but the ribs are generated procedurally from the end-point parameter

functions. These functions map from the end-point information for a rib (positions, Frenet frames) to the parameters for the rib curve (e.g. end-point tangents for cubic Hermite ribs). The end-point information includes the global position and the Frenet frame along the guide rail curve at each end-point, as well as the sweep parameter s for that rib. The function is then expected to return the parameterization of the rib with those end-points. The parameters the function must return depend on which rib type has been specified. We have found two to be sufficient for many applications: cubic Hermite splines and circular arcs. For the cubic Hermite spline ribs, the function must provide the derivative information (tangent directions and magnitudes) at the end-points. For circular arc ribs, the function must provide two angles: one that determines the plane in which the rib lies, and a second that specifies the angle between the rib tangents and the chord between the end-points. These parameters are described in further detail below. Additionally, several default functions are provided that implement some useful idioms for describing the rib parameters. These functions are themselves parameterizable. They are also described in more detail below.

3.2.1 Circular Arcs

The simplest type of rib (after a straight line) is a circular arc. For circular arc ribs, the default function is parameterized by the desired values for the two angles at given sweep parameter values. The function merely interpolates these user-provided values across the sweep. The angles control the rib shape and orientation as depicted in Figure 6. The angle ρ selects the plane of the rib and the angle θ is the measure of the angle between the chord connecting the end-points and the take-off direction of the rib. While circular ribs have been sufficient for the applications created thus far, more general elliptical ribs might be a useful extension. For these, we could add another rib parameter to specify the rib shape in terms of elliptic eccentricity.

3.2.2 Cubic Hermite Splines

Several alternative rib parameterizations have been implemented as default functions for the cubic Hermite spline rib type. All depend on the properties of the two end-points of the rib and the properties of the rail curves there. For cubic Hermite curves, there are four parameters: two end-point positions and two end-point derivatives. The end-points are simply points along the guide rail curves, evaluated at the proper rail parameter values for that rib. The end tangents might be oriented in any

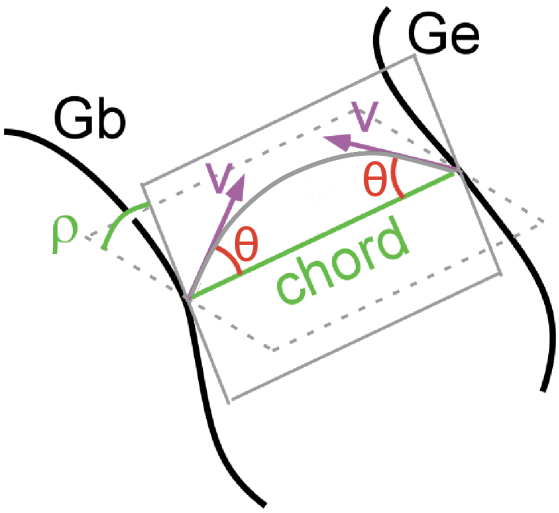
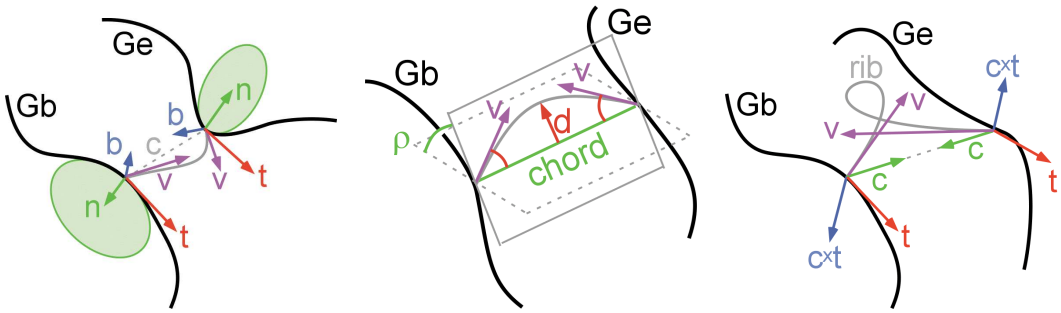


Figure 6: The parameters for circular arc ribs. ρ specifies the plane in which the rib lies, the vectors V are the tangents to the circular arcs at the end-points, and the θ is the angle between these vectors and the chord between the end-points.

direction, and different coordinate systems have different advantages for specifying the derivative information, as will be discussed below. The orientation angles of the end tangents in these rib-local coordinate systems may vary as a function of the sweep parameter s, which ranges from zero to one across the sweep along the guide rails. This is implemented by parameterizing these rib shape functions by, in the simplest case, specifying rib end condition parameters for the start and end of the sweep (sweep parameter s=0 and s=1, respectively). The function then linearly interpolates these values for all ribs in between (that is, for $0 < s < 1$). For more complicated cases, any number of parameter settings might be given to the rib shape function, each associated with a particular value of the sweep parameter, and the function will interpolate them over the domain of s.

The most general coordinate system would simply use the Frenet frame of the rail curves as the coordinate system to define the directions of the rib tangents (Figure 7(a)). A first parameter alpha specifies the angle between the rail tangent and the rib tangent. A second parameter theta specifies the rotation angle around the rail tangent, starting from the osculating plane; thus a parameter combination of alpha=90 and theta=180 would result in a rib that takes off perpendicularly to the rail curve in the outward direction of its bend. This coordinate system is most convenient when we are primarily concerned with the structuring of the rib ends around the rail curves. Of course, there is one more ("velocity") parameter to specify the length of the derivative



(a) (b) (c)

Figure 7: Three default rib end-point coordinate frames: (a) the Frenet frame coordinate system, (b) the chord-based coordinate system, and (c) the hybrid coordinate system.

(tangent) vector of the Hermite rib curve.

Often we are more concerned with the overall shape of the ribs themselves. In this case we would associate the specifications of the end conditions of the rib in a coordinate system that is more intimately tied to the rib (Figure 7(b)). The chord that connects the two end points of the rib sliding along the rails forms the dominant axis of such a coordinate system. A plane that passes through this dominant axis is selected by a single angle parameter. The plane corresponding to a zero angle might be application-specific or default to some function of the components of the Frenet frames at the rib end-points. In many cases, it might be desirable to keep the ribs planar and symmetric; in this case all we need to specify is the orientation angle of the rib plane around the dominant coordinate axis and the angle that the rib ends form with the chord. Alternatively the amount of bending of the rib in the given plane could be characterized as an offset distance, d, of the rib mid-point from the chord mid-point.

As a third alternative we have used a mixture of these two paradigms, resulting in end-point coordinate frames in which the three axes may not be mutually orthogonal (Figure 7(c)). For each end point, the first axis is the vector from that point to the opposite rib end-point. The second axis is the tangent of the guide rail curve. The third axis is simply the cross product of the first and second axes.

The proper parameterization of rib end conditions is ultimately application-specific, and may reflect both aesthetic and pragmatic requirements. In such cases, a user may implement a rib shape function from scratch that uses non-local information to determine the derivative vectors of the Hermite splines. One of the example applications described below does exactly this.

Functionally Optimized Ribs

Hermite curves are mathematically smooth in their parameter domain, but they are not necessarily geometrically smooth; cubic Hermite curves might still have loops, cusps, or folds. When the intention is to realize a ribbed surface by bending physical materials, geometric smoothness is desirable. To this end one can replace the simple curves with optimized curves that adjust the available curve parameters so as to minimize overall strain energy. For a cubic Hermite representation, the initial end-point tangents could be specified as detailed above, and then sent to an optimization procedure to construct the energetically optimized and geometrically smoothed curves [8].

Chapter 4

Applications and User Interfaces

4.1 Solstice Emulation

We have constructed a program that procedurally generates sculptures in a class generalized from Perry's *Solstice*. We expose a small set of parameters that in turn determine the ribbed surface parameter values. Next we describe in detail the application-specific parametrization, including discussion of possible alternatives and the motivation for each particular choice.

4.1.1 Parameters

Rail Parameters

The first four parameters in our *Solstice* program, P, Q, *P Radius*, and *Q Radius*, determine the guide rail as a $(P,\ Q)$ torus knot, where the curve makes P revolutions around the major circle of the torus and Q revolutions around the minor circle (that is, around the toroidal arm). In the implementation described, we require that P and Q be mutually prime to construct a single, proper torus knot. An alternative implementation might instead create multiple torus knot guide rails at regular rotational offsets whenever P and Q are not mutually prime. This extension is not discussed here. *P Radius* specifies the radius of the major circle, and *Q Radius* specifies the radius of the minor circle, or the thickness of the toroidal arm. Thus, these two parameters determine the scale of the torus knot. Finally, the *Rail Radius* parameter controls the thickness of the guide rail sweep.

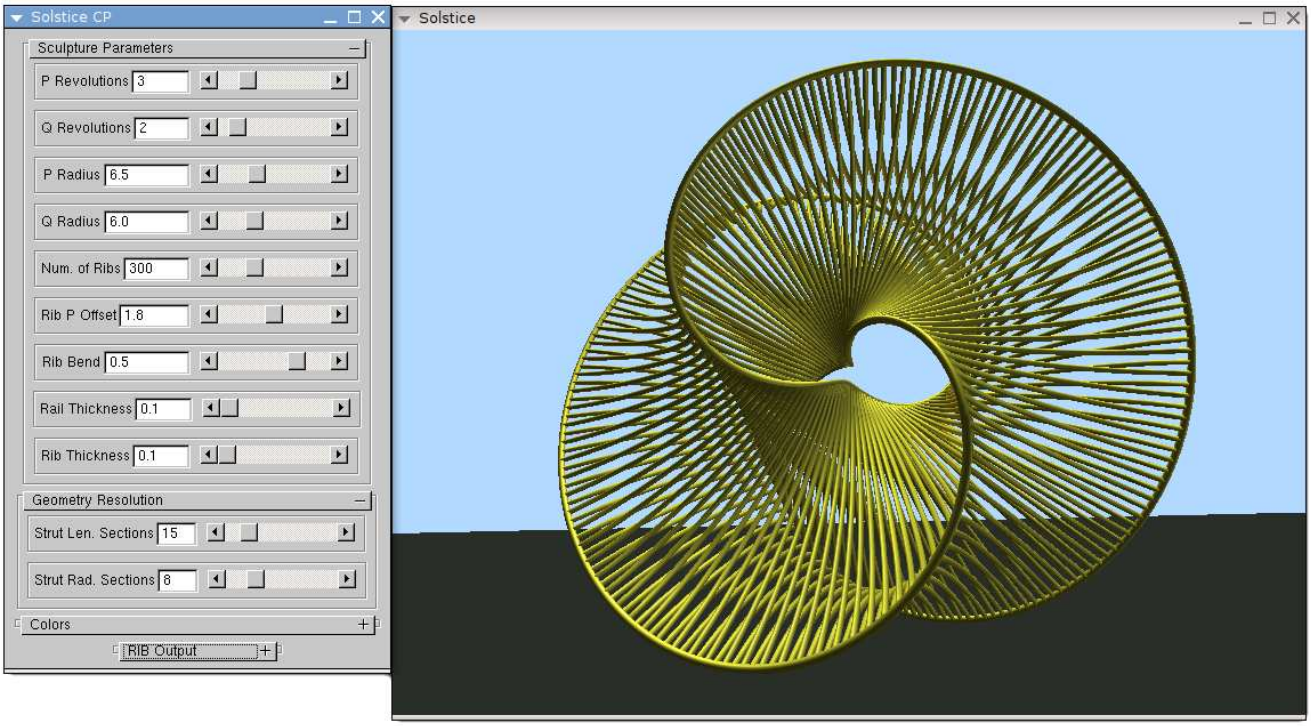


Figure 8: *Solstice* emulation program employing ribbed surfaces

Rib Parameters

The *Number of Ribs* parameter controls the number of ribs that are uniformly distributed around the knot. The next two parameters, *Rib Offset* and *Rib Bend*, control the rib destination parameterization and the shape of the ribs.

The n rib starting points are distributed evenly in the rail curve parameter interval $[0, 1)$. Since the guide rail is a single $(P,\ Q)$ torus knot, the end-points lie on this same rail, but are distributed in an interval offset from the unit interval: by default, the interval $[1/p, 1 + 1/p)$, where parameter values greater than 1 wrap around to stay in $[0, 1)$. This makes the ribs, by default, lie in the planes of minor circles of the torus, since the rib end-points are all exactly one revolution around the major circle of the torus from the starting points. The *Rib Offset* parameter shifts the ribs out of these planes of minor circles of the torus. At the default value of 0, the end-point interval is as described above. For non-zero values, the end-point interval is shifted, thus moving the rib end-points along the guide rail curve. The parameter itself is not presented in terms of the curve parameter but instead in terms of the angle around the major toroidal circle, measured in degrees. Again, for a value of 0, the interval is $[1/p, 1 + 1/p)$, and so the ribs become P-gons in planes of minor circles of the torus. A full revolution

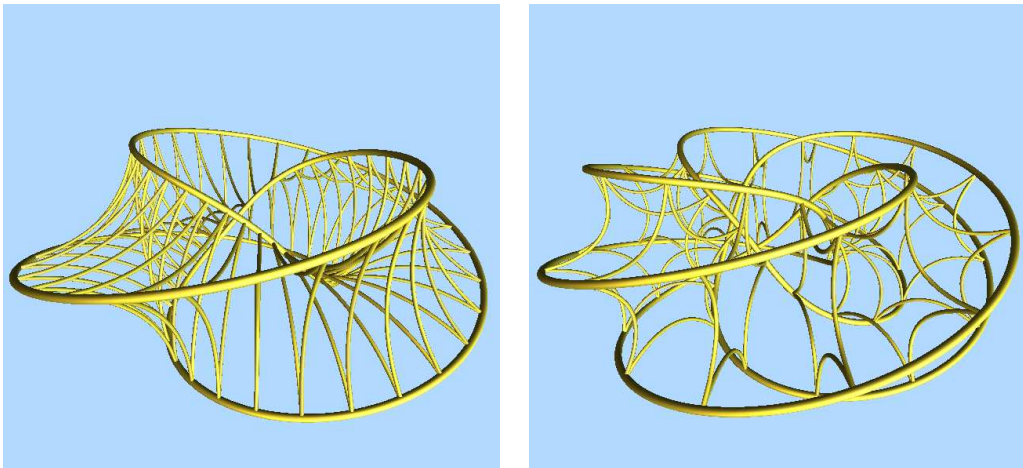
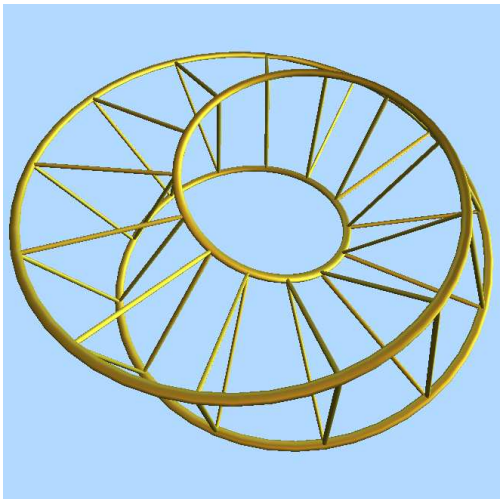


Figure 9: Effect of varying P while *Rib Offset* remains constant.

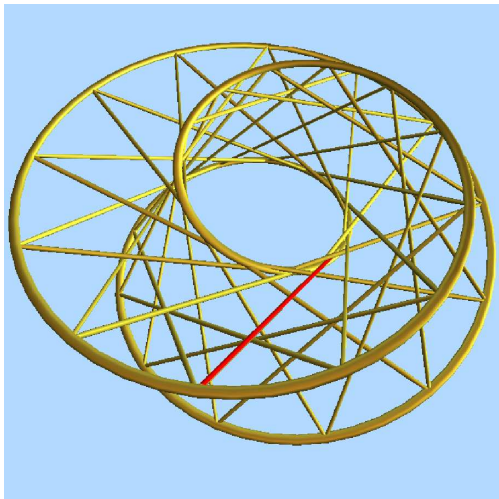
of the end-point around the major toroidal circle, an offset of 360°, corresponds to an offset of the end-point interval by $1/p$. The reason for this mapping is the desire to keep this parameter and the guide rail parameter P reasonably independent. If a user specified the offset explicitly in terms of the rail curve parameter interval, modifying the P parameter would have a significant impact on the configurations of the ribs. With the mapping, we get the behavior depicted in Figure 9, where the offset remains at zero, and so while P varies the ribs remain in the planes of the minor circles of the torus. The result of different settings for the offset parameter are shown in Figure 10.

Several different alternatives were explored for how best to specify the curvature of the ribs, controlled by the *Rib Bend* parameter. Here each is described in detail. The first decision made was that the rib curves would be constrained to be planar. Thus the first task is to select a plane in which to form the ribs. In the case where *Rib Offset* is zero, these planes are most naturally the planes of minor circles of the torus, and thus they include the center of these minor circles. This suggests that we do not select the plane based on the properties of the guide rail curves at a given rib's end-points, but instead based on some global information that takes the toroidal structure into account. A second constraint is that the ribs must be symmetric; the end tangents at either end-point must be mirror images of each other. This gives us two more parameters: the length of the tangents, and their direction. Each scheme for constructing a rib's shape below sets these two parameters in a slightly different way.

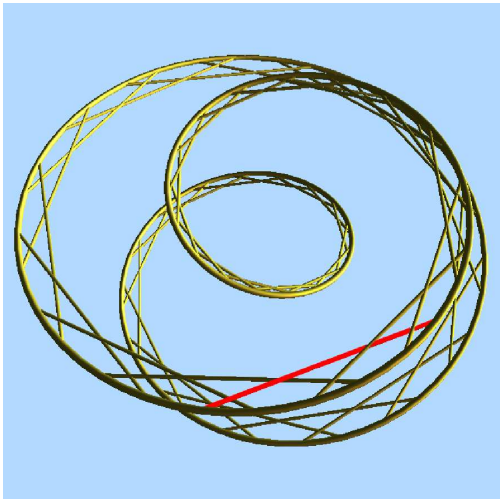
The first approach was motivated by the understanding of Solstice as a "hyperbolic" triangle swept around a circle. Representing the ribs as a cubic Bezier curve and enforcing symmetry, the two inner Bezier control points may be translated perpen-



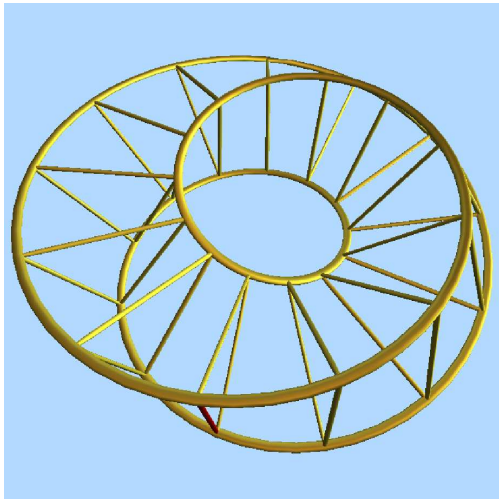
(a) *Rib Offset* = 0



(b) *Rib Offset* = 83



(c) *Rib Offset* = 303 = -57



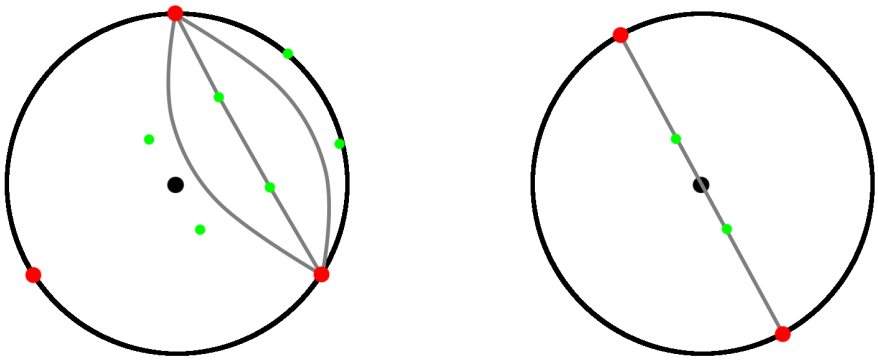
(d) *Rib Offset* = 360 = 0

Figure 10: Effect of *Rib Offset* parameter for different values.

dicular to the chord between the end-points. It is natural to constrain this translation to the plane in which each of these "hyperbolic" triangles lies. Thus, in this scheme, the *Rib Bend* parameter translates the two inner control points towards or away from the center of the circular cross-section of the toroidal arm in which the triangle lies. With a value of 1 the control points are translated towards the center so that the control points and the center are collinear, and with a value of -1 they are translated this same distance but away from the center 11(a). To conform with the interface of ribbed surfaces, described above, this description of the ribs is used to specify the end conditions as a special function of the sweep parameter and rib end-points rather than one of the default functions for cubic Hermite spline ribs (Figure 7(a) or 7(c)).

This approach, while sufficient for many cases, presented two problems. The first is that because of the *Rib Offset* parameter, ribs do not necessarily lie in the plane of a minor circle of the torus. To overcome this, we instead take the midpoint of the chord between the end-points and use the center of the minor circle that passes through that point. The *Rib Bend* parameter then translates the inner control points towards and away from this center-point. This scheme is shown in Figure 11(a). The second problem is that for generalized toroidal ribbed sculptures, it is possible that a rib chord actually passes through the center of the minor circle used to define the plane of the rib and the amount of rib bending. In this case, the vector from the centroid of the end-points to the center of the minor circle is null, and no translation will result, yielding a straight rib. This case is depicted in Figure 11(b).

This problem is resolved by understanding the task at a more abstract level. Since we require planar ribs in this application, the basic problem, as is mentioned above, is to select a plane that contains the rib end-points. In the previous scheme, the inner control points are then translated perpendicular to the chord between the end-points within this plane. The plane was implicitly selected by the vector from the centroid of the end-points to the center of the minor circle in which the chord midpoint lies. The final approach, shown in Figure 12 instead selects the plane of the rib explicitly. The naïve approach selects a desirable plane in all cases except when the midpoint of the rib end-points is at the center of a minor circle of the torus. In these cases, we desire a plane parallel to a vector that both lies in the plane of this minor circle and is perpendicular to the direction of the chord between the rib end-points. To summarize the final implementation, consider the midpoint of this chord, m_c. The normal of the minor circle in whose plane m_c lies is $\vec{n_m}$, where $\vec{n_m}$ is simply the unit tangent to the major circle of the torus at this slice of the toroidal arm. To determine the plane of the



(a) (b)

Figure 11: Green dots represent the inner control points of the cubic Bezier curve representing the rib. Red dots represent the intersection of the rail curve with this minor circle of the torus. Gray lines represent the resulting rib curve. (a) shows the naïve *Rib Bend* implementation. The control points are translated either towards or away from the center of the minor circle, corresponding to negative and positive values of the *Rib Bend* parameter, respectively. (b) shows the naïve *Rib Bend* problem case. When the center of the chord between the rib endpoints is identical to the center of this minor circle, no translation will result, and so the rib will always be linear.

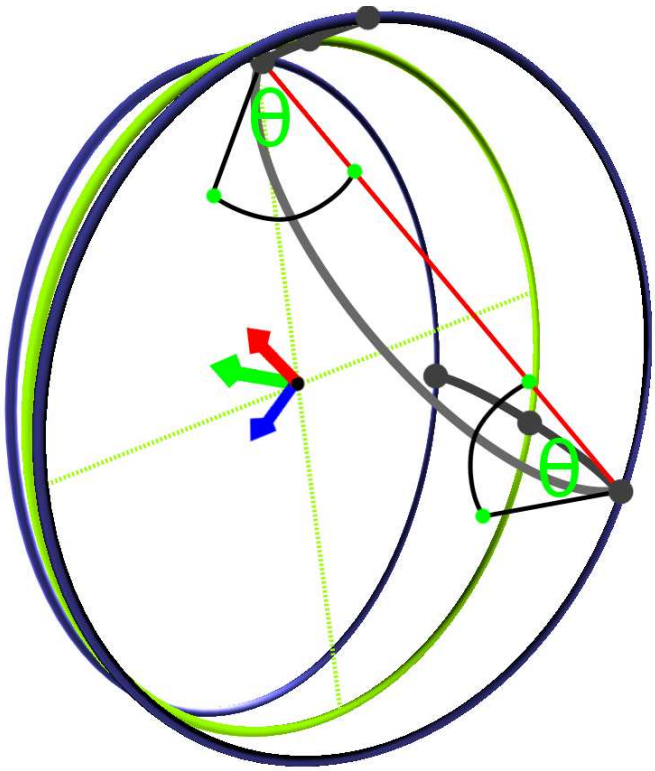


Figure 12: The final *Rib Bend* implementation. The red arrow represents the unit vector from the first rib endpoint to the second. The green arrow represents the unit normal of the plane of the minor circle in which the center of the chord between the endpoints lies (this minor circle is represented by the large green circle). The blue arrow is the cross product of these two vectors. The rib curve control points are rotated in this plane, where the *Rib Bend* parameter specifies the tangent of the angle θ.

rib, we want a vector $\vec{v}$ such that $\vec{v} \cdot \vec{n_m} = 0$ and $\vec{v} \cdot \vec{d_r} = 0$. Thus, $\vec{v} = \vec{d_r} \times \vec{n_m}$. The rib is then constrained to lie in the plane containing the rib endpoints and parallel to the span of the vectors $\vec{d_r}$ and $\vec{v}$. This solution breaks down when $\vec{d_r}$ is parallel to $\vec{n_m}$ (i.e. perpendicular to the plane of the minor circle under consideration), but we disregard this, since we wish only to support ribs that deviate slightly from lying in planes of minor circles of the torus. Since this scheme handles all of these desirable cases except for the pathological one mentioned, it turns out to be a good condition for selection of a rib plane in general. The *Rib Bend* parameter is then tied directly to the tangent of the angle (θ in Figure 12) that the chord between the two rib end-points makes with the end tangent vectors at either end of the rib. The length of these vectors is constrained to 1/3 of the length of the chord. Thus, for a *Rib Bend* value of zero, straight ribs are created, and as *Rib Bend* approaches $+/-\infty$, the ribs take off perpendicular to the chord between the end-points.

As with the *Rail Radius* parameter, the *Rib Radius* parameter simply controls the thickness of the rib sweeps.

4.1.2 Implementation

The sculpture visualization program, named Solstice after the motivating work by Perry, is implemented in C++. It has been built and tested on both GNU/Linux and Windows.

The user is presented with two windows. The first presents a column of sliders and text boxes that alter the parameters. Additional GUI controls, implemented with the GLUI library, a user interface library built atop GLUT, affect secondary features such as color and geometric resolution. Changes to the sculpture parameters are observed immediately in the second window, which presents a real-time 3D rendering of the sculpture with an ArcBall interface with zoom and translation controls [6]. The geometry is reconstructed from scratch on every change to the sculpture parameters.

The user has simple control over some surface properties of the sculpture, such as color. The visibility of a ground plane may be toggled, and the orientation of the up vector may be specified. The program supports saving the scene geometry with these surface parameters and ground plane visibility to a RIB file that may then be rendered by a renderer that supports the RenderMan interface, such as Pixie [2]. Figures 13 and 14 present some renderings of this output.

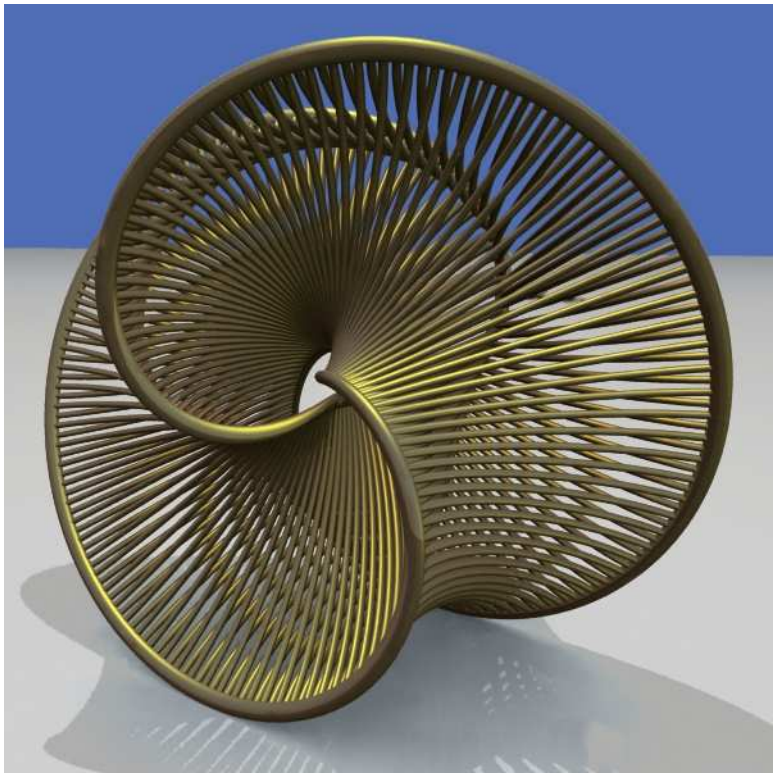


Figure 13: Emulation of Perry's *Solstice* with denser ribs, rendered in Pixie.

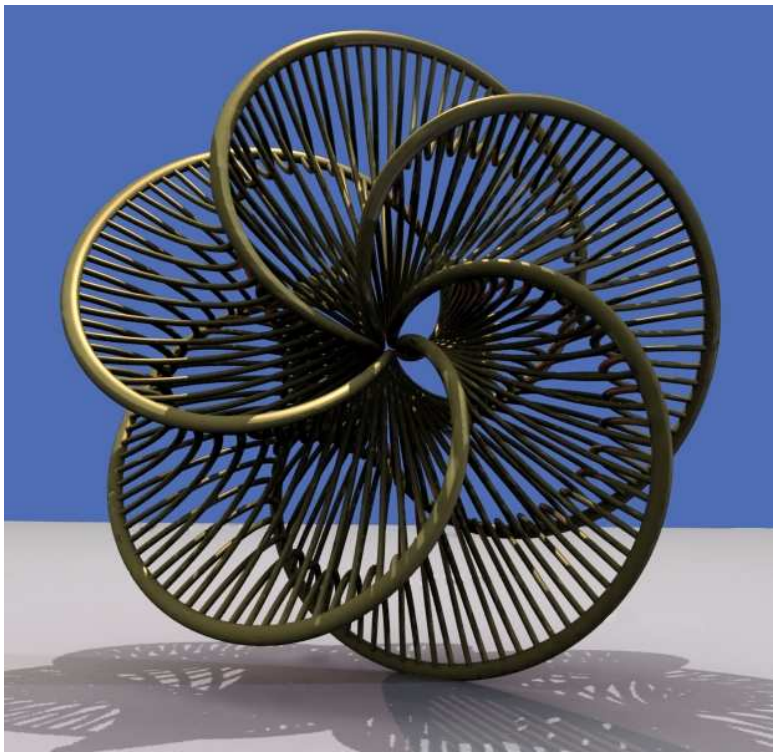


Figure 14: A variant of Perry's *Solstice*, based on a (4, 5) torus knot, rendered in Pixie.

Figure 15: Charles O. Perry's ribbed sculpture *Early Mace*, installed in Atlanta, Georgia.

4.2 Early Mace Emulation

Development of ribbed surface applications has also been influenced by a second sculpture, Charles Perry's "Early Mace", installed in Atlanta, Georgia (Figure 15). Here the guide rails are pairs of simple semi-circles with two widely different radii. The ribs are almost quarter circles, always connecting one of the large semi-circles with one of the small ones. Under this description, a simple but intriguing set of variations can be produced by changing the bulge of all the ribs, which can vary from the concave appearance of the original sculpture to a convex nearly spherical shape with two semi-circular slots. As we will see, this parameter can be directly captured in one of the end condition representations for ribbed surfaces.

4.2.1 Parameterization

For this application, the default function for circular arc ribs is used. The plane for an angle value of 0 gives us the desired plane for each rib, the plane perpendicular to the guide rail circles. The second angle parameter is exposed to the user, allowing the convexity/concavity of the ribs to be adjusted. Figure 16 demonstrates the result of setting this parameter to three different values, one with concave ribs, one with straight ribs, and one with convex ribs, resulting in an emulation of *Early Mace* and variations.

(a) (b) (c)

Figure 16: *Early Mace* emulation and variations. (a) is an emulation of *Early Mace*, (b) is a variation with straight ribs, and (c) a variation with convex ribs.

4.2.2 Implementation

This application was implemented atop SLIDE [7]. The SLIDE application was supplemented via Berkeley SLIDE's TCL interface with a module for constructing ribbed surfaces and managing dynamic geometry. A single ribbed surface, using two semi-circular guide rails as described, is created. This surface is then replicated three times and the four instances transformed to produce the whole sculpture.

4.3 More Ribbed Sculptures

Our own experimentation with creating novel ribbed sculptures uses hemi-elliptical ribs spanning a ribbed double helix that is bent into a parabolic arch. The two guide rail curves form this double helix, revolving a user-specified number of times from start to end. The radius of the two helices may be specified at the end-points and and at the apex of the arch. The double helix is curved to follow a hemi-elliptical path, which is specified in terms of the parameters of interest for the overall arch structure: the width of its base and its height. The rib end conditions are specified using the more general coordinate system of chord vector, guide rail tangent, and their cross product. The ribs take off in the direction of this cross product, and these end conditions are specified for each end-point of the arch to cause them to take off upwards at both ends. The values are then interpolated for each rib in-between. Figures 17 and 18 show two variations of such sculptures output by this program, which were then rendered in Pixie.

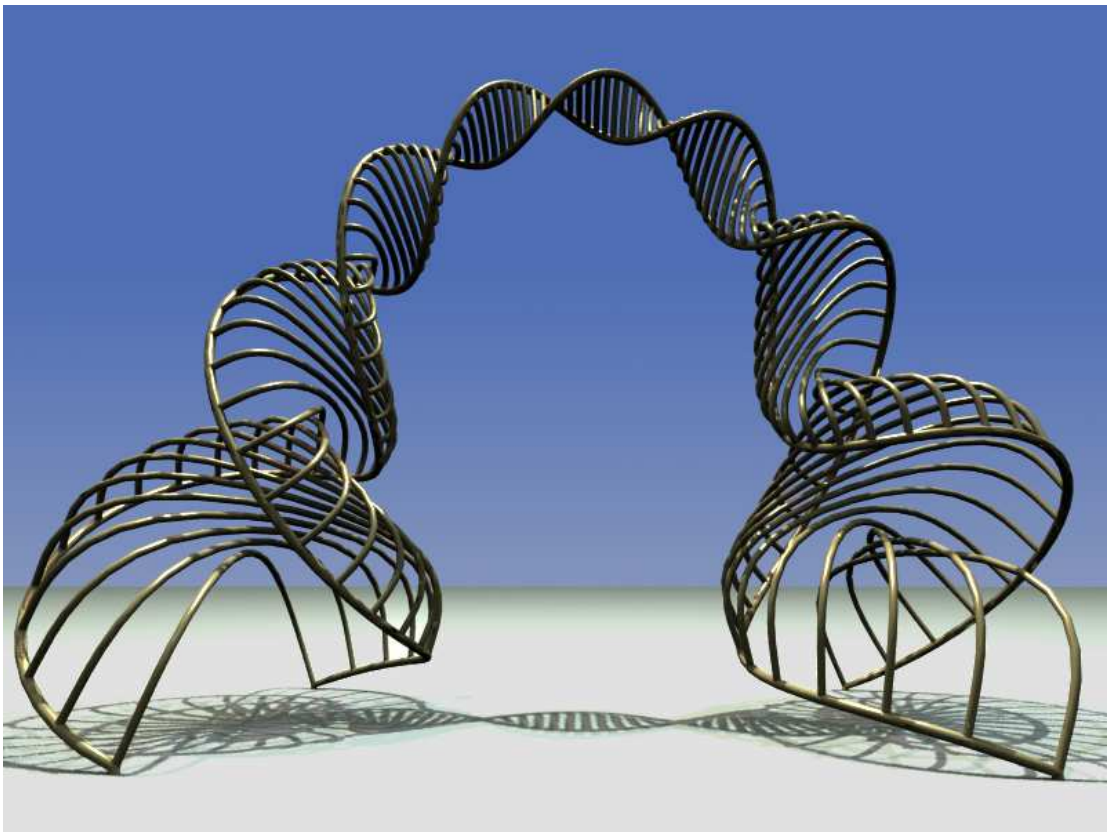


Figure 17: Helical arch with narrow apex, designed with ribbed surfaces

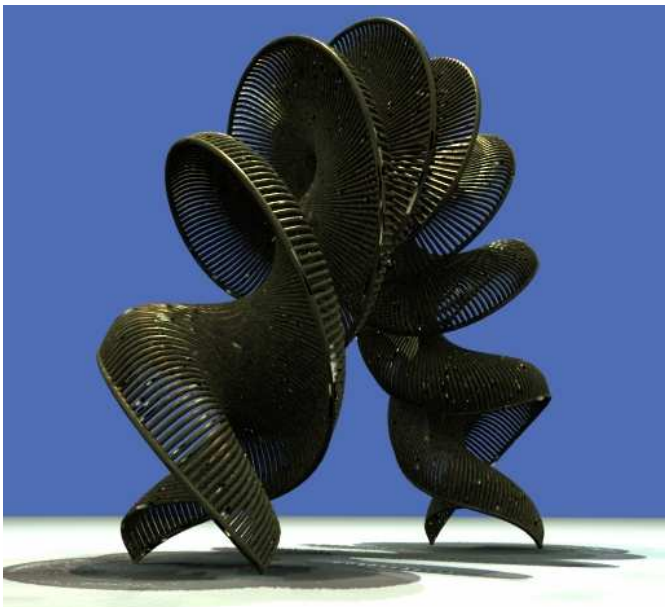


Figure 18: Helical arch with wide apex and many ribs, designed with ribbed surfaces

(a) (b)

Figure 19: Two balcony railings generated with the ribbed surface module

4.4 Architectural Applications

There are several other domains where a parameterized ribbed surface may be the natural choice for modeling the objects to be designed. The examples that readily come to mind range from architectural facades and balconies to ship building. Figure 19 demonstrates two balcony railing designs characterized as ribbed surfaces. The first design, (Figure 19(a)), sets the rib end tangents to fixed locations in the Frenet frames of the guide rail curves, as per the rib parameterization scheme in Figure 7(a). The second design, (Figure 19(b)), is composed of two ribbed surfaces with offset parameter intervals, so the ribs from each surface slant in opposite directions, crossing each other as shown.

4.5 Mathematical Visualization

Another domain in which ribbed surfaces find special utility is in mathematical visualization, where this type of "ribbed representation" can provide surfaces with adjustable "transparency." This is of particular importance when providing visualizations for complex surfaces with high depth complexity, where it is desirable to accommodate a look to the inside or to the backside of the structure. Mathematicians have modeled ruled surfaces with a "ribbed" approach in the form of "string-art" models to depict surfaces such as the hyperboloid depicted in Figure 20(a). A sculpture depicting Boy's surface, a finite immersed model of the projective plane, uses metal bands to create a surface of some transparency that allows one to look inside and see the internal lines of self-intersection and the triple intersection point (Figure 20(b)). And many string-art sculptures make use of the partial transparency that they can achieve for enhanced aesthetic results (Figure 20(c)). For large public sculptures, a transparent look and feel is

(a) (b) (c)

Figure 20: Models with transparent ruled or ribbed surfaces: (a) Hyperboloid, (b), and (c) ruled string art by Ray Schechter.

often preferred to a massive, solid shape that would cast stark shadows. Tubular construction of ribbed surfaces can provide this airy realization, and it often also reduces the overall weight and construction costs.

As an example of the added visualization power that ribbed surfaces can provide because of their adjustable "transparency," Figure 21 shows models of abstract polytopes that exhibit intersecting faces when modeled in 3-dimensional space. Figure 21(a) depicts a physical model of a hemi-cube comprising 4 vertices, 6 edges, and 4 non-planar mutually intersecting quadrilateral faces. Figure 21(b) is a ribbed model of Steiner's Roman surface, the simplest and most symmetrical object of a single-sided, non-orientable surface of genus 1. Both these models were made on a Fused Deposition Modeling machine. Figures 21(c) and 21(d) depict two views of a computer model of a hemi-dodecahedron consisting of six pentagonal faces, 15 edges, and 10 vertices. These polytopes form the building blocks for the construction of some intriguing higher-dimensional polychora of very high symmetry . They have been generated with an early precursor program in our family of ribbed surface generator programs. They mostly just use straight ribs and thus depict models that are mostly bounded by intersecting, piecewise ruled surfaces.

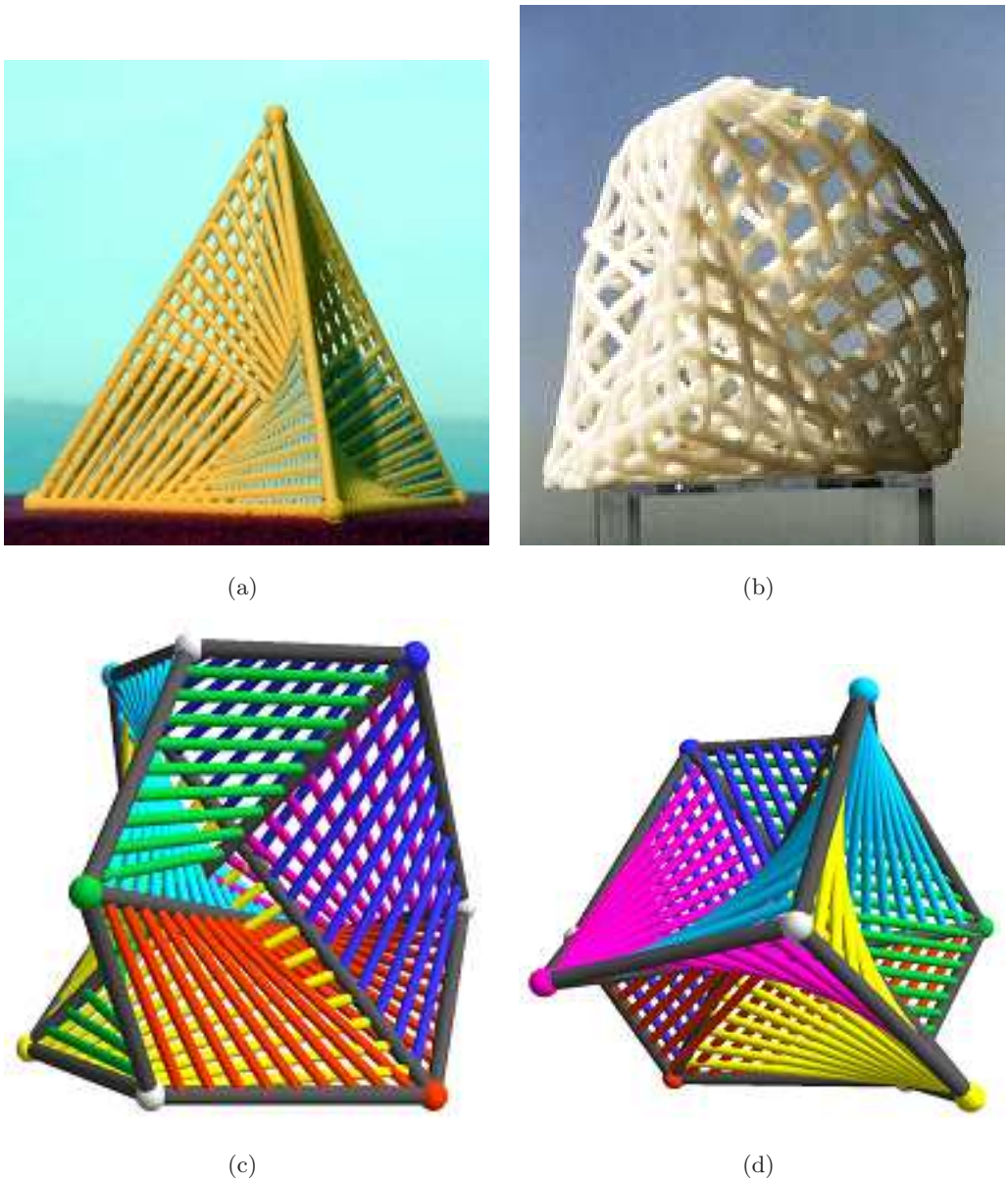


Figure 21: Mathematical visualization models: (a) hemi-cube, (b) Steiner surface by Carlo Séquin, (c) hemi-dodecahedron, and (d) another view of the hemi-dodecahedron.

4.6 General Ribbed Surfaces

4.6.1 Programmatic Interface

Having implemented and tested these systems fruitfully, the creation of an application supporting the creation of general ribbed surfaces is an obvious next step. Such a system would have to support the following:

1. Procedural generation of geometry using procedural components.
2. User interface specification.
3. Programmatic updates from user input.
4. Specification of non-geometric parameters, e.g. surface properties.

Instead of requiring that curve generation code be inserted into the C++ implementation of the system for every style of ribbed sculpture, the system defers this task to a Python script. Boost.Python is used to expose functions for rail curve generation and mapping ribs between rails, allowing for arbitrary sculptures within the ribbed sculpture paradigm [1]. Functions for linking sculpture parameters to user interface elements are also provided.

We expose a class and some functions in a Python module to allow for generation of these "ribbed" sculptural forms. Appendix A lists the objects, methods, and functions exposed by the Python API, including function signatures and behavior. To demonstrate the use and capabilities of this module, we have implemented an emulation of Charles Perry's sculpture *Harmony* (Figure 23), installed in Hartford, Connecticut. Figure 22 shows a screenshot of the resulting program, and the Python code that generated it is listed in Appendix B.

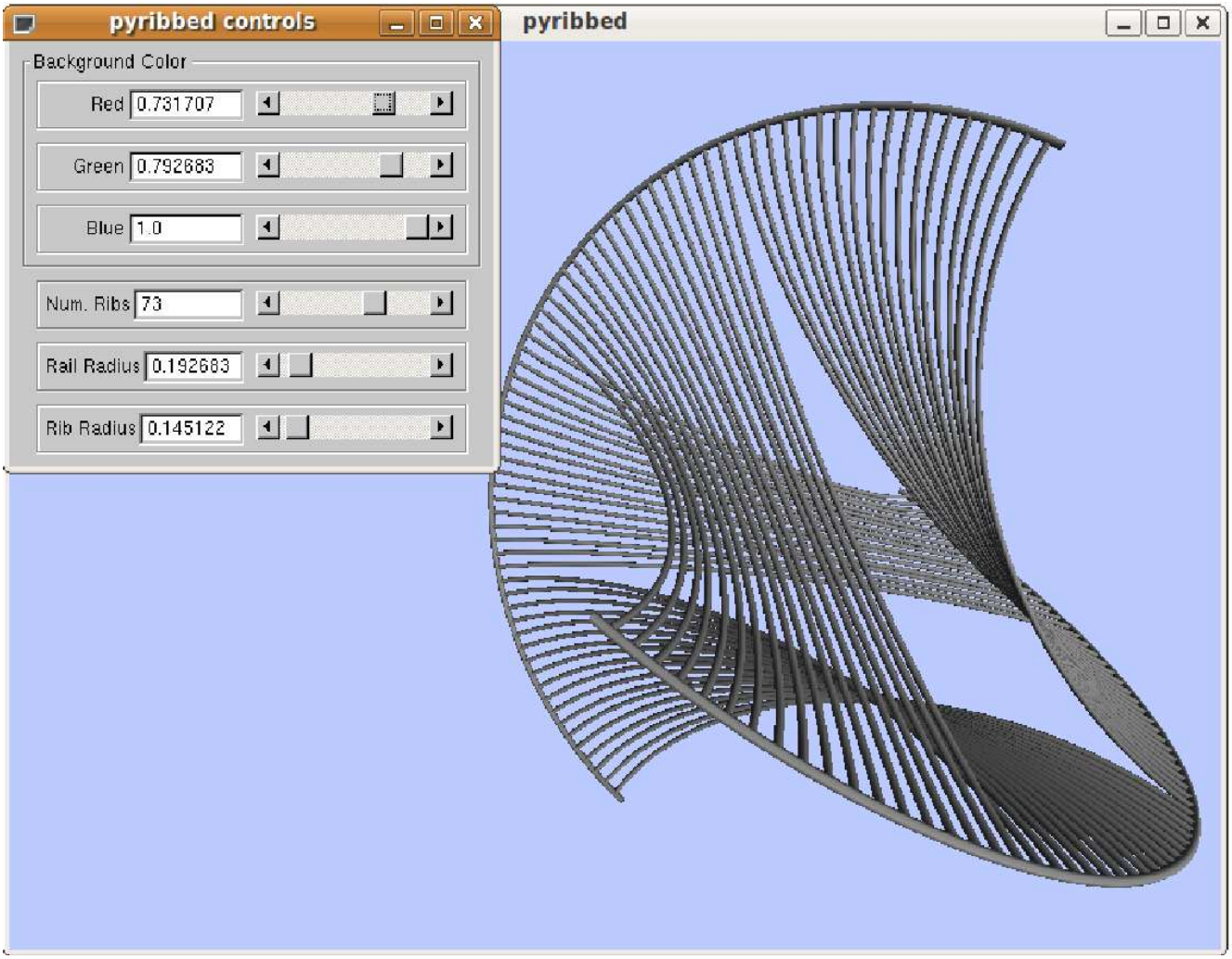


Figure 22: Python extension example: Emulation of Perry's *Harmony*

Figure 23: Charles O. Perry's *Harmony*

Chapter 5

Summary and Conclusion

I have analyzed the ribbed surfaces, distinguishing them from other families of sweep surfaces, and explaining their motivational source in the tubular ribbed sculptures of Charles Perry. From a relatively small number of parameters, which include one or two guide rails, two guide rail intervals, rib type, number of ribs, and a function providing rib end conditions for each value of the overall sweep parameter, various heterogeneous forms can be described more economically than with other sweeps.

Several applications of ribbed surfaces and their implementations have been presented. Ribbed surfaces may be utilized in domains from mathematical visualization to the design of architectural elements and stand-alone sculptures. In mathematical visualization, ribbed surfaces may be leveraged for their particular mode of "transparency", which allows a user to see through otherwise occluding surfaces, while still conveying the shape of both occluded and unoccluded surfaces through the opaque individual ribs and their visual density and shape. Ribbed surfaces might also be used for architectural components such as balcony railings or certain furniture that are naturally composed out of ribbed elements. In particular, ribbed surfaces are a natural choice for the design and realization of certain large-scale outdoor sculptures, since they may be composed out of relatively cheap and lightweight components and their "transparency" allows for large but unimposing forms that cast interesting shadows. Further beneficial work would be to add an extention to a popular design program to generate and represent ribbed surfaces. This could be done in Maya via a MEL script [5].

Appendix A

Python Module API: pyribbed

- – class Vector(xVal, yVal, zVal)

  A class representing a vector in 3-space.

  – Vector.__get_item__(i)

  Index operator; returns the value of element i.

  – Vector.__set_item__(i, val)

  Index operator; sets the value of element i.

  – mag_squared(vec)

  Returns the squared magnitude of a Vector.

  – mag(vec)

  Returns the magnitude of a Vector.

  – normal(vec)

  Returns a unit vector in the direction of a Vector.

  – normalize(vec)

  Normalizes a Vector, returning the same input Vector.

  – dot(vec0, vec1)

  Returns the dot product of two Vectors.

  – cross(vec0, vec1)

  Returns the cross product of two Vectors.

- – class Curve()

  A class representing a parameterized curve in 3-space.

- Curve.sample(t)

  Evaluates the curve at a parameter value t. Values of t outside of [0, 1], are mapped into [0, 1].

- class BezierCurve(degree)

  A subclass of Curve representing a Bezier curve. The degree may be specified in the constructor, initializing the curve with degree+1 control points.

- Vector.__get_item__(i)

  Index operator; returns the ith control point.

- Vector.__set_item__(i, val)

  Index operator; sets the ith control point.

- BezierCurve.push_back(cp)

  Adds a new last control point, increasing the degree by 1.

- BezierCurve.size

  Returns the number of control points for the curve.

- BezierCurve.degree

  Returns the degree of the curve.

- class BSplineCurve()

  A subclass of Curve representing a B-spline curve.

- BSplineCurve.push_back(cp)

  Adds a new last control point.

- BSplineCurve.size()

  Returns the number of control points for the curve.

- BSplineCurve.set_degree(degree)

  Sets the degree of the BSpline curve (i.e. of the basis curves).

- BSplineCurve.degree()

  Returns the degree of the BSpline curve.

- BSplineCurve.set_closed(closed)

  If closed is true, wraps control points so that the curve is closed. If closed is false, this is not done.

- BSplineCurve.is_closed()

  Returns true if the curve is closed, false otherwise.

- class Polyline()

  A subclass of Curve representing a polyline.

- Polyline.push_back(vertex)

  Adds a vertex to the end of the polyline.

- Polyline.size()

  Returns the number of vertices in the polyline.

- class SculptureProgram()

  The class representing the sculpture program. Any particular program must define a class that inherits from SculptureProgram.

- SculptureProgram.param_color_chooser(name, default)

  Adds a color selection GUI control with *default* as the default color and *name* as the label.

- SculptureProgram.param_slider_int(name, int min, int max, int step, int def)

  Adds a slider GUI control for an integer value.

- SculptureProgram.param_slider_float(name, int min, int max, int step, int def)

  Adds a slider GUI control for a float value. The arguments are as for the integer slider.

- SculptureProgram.set_background_color(color)

  Set the background color for the display window.

- SculptureProgram.instance(geometry, color = Vector(0.5, 0.5, 0.5))

  Instantiate a geometric object with a particular surface color.

- SculptureProgram.light(position, color, intensity)

  Add a point light to the scene with a particular position, color, and intensity.

- SculptureProgram.update(params)

  A pure virtual function that must be overridden in any subclass. *params* is a Python dict mapping names (as in the parameter methods above) to the values of the parameters set by the GUI elements. This method should use calls to instance(), light(), and set_background_color() to set up the scene.

- SculptureProgram.begin_main_loop()

  Begin the program loop, initializing windows and GUI elements. It calls update() whenever changes are made to parameters via the GUI.

- sweep(profileCurve, pathCurve)

  Returns a geometric object representing a sweep with a Curve *profileCurve* swept along a Curve *pathCurve*.

- hermite_ribbed_surface(curve0, interval0, curve1, interval1, ribParamFunc, numRibs, ribCross)

  Returns a list of geometric objects representing the (cubic Hermite) ribs of a ribbed surface.

Appendix B

Python Module Code Sample

```
import math
import pyribbed
from pyribbed import Vector, mag, mag_squared, normalize, dot, cross
from pyribbed import sweep, hermite_ribbed_surface

def circle(self, xVec, yVec, startAngle, endAngle, samples):
    '''
    A helper function to create circular polylines.
    '''
    pl = pyribbed.Polyline()
    for i in xrange(samples):
        t = float(i) / float(samples - 1)
        angle = t * (endAngle - startAngle) + startAngle
        sample = math.cos(angle) * xVec + math.sin(angle) * yVec
        pl.push_back(sample)
    return pl

NUM_RAIL_SAMPLES = 50
class Harmony(pyribbed.SculptureProgram):
    def __init__(self):
        '''
        Add sliders and a color chooser for the sculpture
        parameters. Sets the label text min, max, step,
        and default value for each GUI element.
        '''
        ### First, call parent class' constructor.
        pyribbed.SculptureProgram.__init__(self)

        self.param_slider_int("Num. Ribs", 20, 100, 4, 50)
        self.param_slider_float("Rib Radius", 0.05, 2.0, 0.05, 0.25)
        self.param_slider_float("Rail Radius", 0.05, 2.0, 0.05, 0.3)
        self.param_color_chooser("Background Color", (1, 1, 1))
```

```
def update(self, params):
    '''
    The update() function constructs the
    sculpture geometry, creates the lights, and sets the background
    color. The params argument is a dict of parameters from the GUI
    elements created in the constructor. The key is the GUI label (a
    string), and the value type depends on the type of the GUI element.
    This method should use the self.instance() method, which takes an
    object representing a piece of geometry, an optional surface
    description object (material), and an optional transformation as
    arguments, to instante geometric objects. The self.light method
    adds a light to the scene, and the self.set_background_color()
    method sets the clear color.
    '''

    ### Set the background color to the parameter value given for it.
    self.set_background_color(params["Background Color"])

    ### Construct two polylines for the rails.
    circleYZ = circle(Vector(0, 20, 0), Vector(0, 0, 20),
                              0.0, math.pi, NUM_RAIL_SAMPLES)
    circleXZ = circle(Vector(20, 0, 0), Vector(0, 0, -20),
                              0.0, math.pi, NUM_RAIL_SAMPLES)

    ### Construct two polylines for the rib and rail cross-sections.
    ribRad = params["Rib Radius"]
    ribCross = circle(Vector(ribRad, 0, 0),
                              Vector(0, ribRad, 0),
                              0.0, 2.0 * math.pi, 8)
    railRad = params["Rail Radius"]
    railCross = circle(Vector(railRad, 0, 0),
                               Vector(0, railRad, 0),
                               0.0, 2.0 * math.pi, 8)

    ###
    ### Construct the rail sweeps.
    ### sweep() takes two parameters: a cross-sectional curve and a path curve.
    ### A curve can be a Polyline, BezierCurve, or BSplineCurve.
    ###
    yzSweep = sweep(railCross, circleYZ)
    xzSweep = sweep(railCross, circleXZ)

    ###
    ### Set up the intervals of the rail curves for the four
    ### ribbed surface sweeps.
    ###
    ribsPerSurf = params["Num. Ribs"] / 4
    numRibs = 4 * ribsPerSurf
```

```
endOffset = 0.01
ribSep = (1.0 - 2.0 * endOffset) / (numRibs - 1)
ivals = [0, 0, 0, 0]
surfParamWidth = ribSep * (ribsPerSurf - 1)
for i in xrange(4):
    start = endOffset + float(i) * (surfParamWidth + ribSep)
    ivals[i] = (start, start + surfParamWidth)

reverse = lambda ival: (ival[1], ival[0])
###
### This function is passed to the ribbed surface generation function.
### It takes the rib end-points, frenet frames along the rail curves,
### and overall sweep parameter s, and returns the parameters for the
### rib curves. For Hermite ribs, the function must return a 2-tuple
### of the tangent vectors at each rib end-point.
###
def ribParams(end0, frenet0, end1, frenet1, s):
    return (-10.0 * frenet0.normal,
            -10.0 * frenet1.normal)

###
### The hermite_ribbed_surface function returns a list of geometric
### objects representing a ribbed surface with cubic hermite spline
### ribs. The first argument is the first rail curve, the second
### argument is a parameter interval for that curve (e.g. (0.0, 1.0)).
### The third and fourth arguments are similar but for the second rail
### curve. The fifth parameter is the rib parameter function. Several
### default functions are provided in the package. The sixth argument
### is the number of ribs to create, and the seventh is the rib
### cross-section.
###
ribsRails = pyribbed.hermite_ribbed_surface(
    circleYZ, ivals[0],
    circleXZ, ivals[2],
    ribParams, ribsPerSurf, ribCross)

ribsRails += pyribbed.hermite_ribbed_surface(
    circleYZ, ivals[1],
    circleXZ, reverse(ivals[0]),
    ribParams, ribsPerSurf, ribCross)

ribsRails += pyribbed.hermite_ribbed_surface(
    circleYZ, ivals[2],
    circleXZ, reverse(ivals[3]),
    ribParams, ribsPerSurf, ribCross)

ribsRails += pyribbed.hermite_ribbed_surface(
    circleYZ, ivals[3],
```

```
            circleXZ, ivals[1],
            ribParams, ribsPerSurf, ribCross)

        ###
        ### Instance all of the geometry.
        ###
        for rib in ribsRails:
            self.instance(rib)
        self.instance(yzSweep)
        self.instance(xzSweep)

        ###
        ### Create two lights. The first argument specifies the position,
        ### the second specifies color, and the last the intensity.
        ###
        self.light(Vector(50, 50, 50), Vector(1, 1, 1), 1)
        self.light(Vector(-50, -50, -50), Vector(0.4, 0.1, 1), 0.1)

###
### Instantiate the sculpture class.
###
s = Harmony()

###
### Start the program.
###
s.begin_main_loop()
```

Bibliography

[1] Boost C++ Libraries. `http://www.boost.org/`.

[2] Okan Arikan. Pixie: Open source RenderMan. `http://www.renderpixie.com/`, Last accessed: 2009.

[3] James Hamlin and Carlo Séquin. Ribbed surfaces for art, architecture, and visualization. 2009.

[4] Charles Perry. Ribbed sculptures. `http://www.charlesperry.com/Ribbed.html`, Last accessed: 2009.

[5] Autodesk Maya Press. *Learning Autodesk Maya 2009 Foundation: Official Autodesk Training Guide*. SYBEX Inc., Alameda, CA, USA, 2008.

[6] Ken Shoemake. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the conference on Graphics interface '92*, pages 151–156, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[7] Jordan Smith. SLIDE Educational Rendering System for 3D Interactive Dynamic Environments, 2003.

[8] Jun-Hai Yong and Fuhua Cheng. Geometric hermite curves with minimum strain energy. *Comput. Aided Geom. Des.*, 21(3):281–301, 2004.