EUROGRAPHICS '98 / N. Ferreira and M. Göbel
(Guest Editors)

Volume 17, (*1998*), *Number 3*

A Framework for Synchronized Editing of Multiple Curve Represenstations

Cindy Grimm[†] and Matthew Ayers[‡]

Department of Computer Science, Brown University

Abstract

Editing curves and surfaces is difficult in part because their mathematical representations rarely correspond to most people's idea of a curve or surface. The implementation (and hence, behavior) of most manipulation tools is intertwined with a particular curve or surface representation; this can make reimplementing the tool with a different representation problematic. A system using a single representation must therefore either limit the types of tools available or convert existing tools to work on the system's representation.
In this paper we present a framework for editing curves or surfaces which supports multiple representations and ensures that they stay synchronized. As a proof of concept, we have created a curve editor which contains several tools each of which manipulate one of three different curve representations: polylines, NURBs, and multi-resolution B-splines.

CR Categories: 13.5[Computer Graphics]:Curve, surface, solid, and object representations, Splines, Wavelets. Additional keywords: Direct manipulation, interface issues, curve manipulation.

1. Introduction

Computer representations of curves and surfaces are often complex, mathematical objects with non-intuitive controls. Significant research efforts have focused on how to make manipulation both more efficient and conceptually easier. To date, research has concentrated on taking a specific representation, such as a NURBs surface, and creating new interaction methods and tools expressly for manipulating that representation. Often the tools and techniques for one representation do not easily cross over to another. Thus, to create a system which provides an array of tools, an author would have to either support a variety of underlying representations or rework several interaction techniques to manipulate a single representation. Translating techniques can be difficult, and continuously synchronizing all representations when one of them changes can be a time-consuming process which may prevent such a system from running at interactive speeds.

We propose a framework that supports a variety of tools and multiple representations. Each tool works on a representation that is appropriate for that tool. The framework automatically keeps the different representations synchronized, updating when necessary. Since translating between complex representations can be slow, we use lazy evaluation to synchronize the various representations only when and where necessary. This framework allows us to build systems which are fast, extensible, and support a wide variety of tools. To demonstrate this, we have built a simple curve editing system which operates on polylines, NURBs, and multi-resolution B-splines.

In this paper we begin by discussing existing representations and some tools used to manipulate them. In Section 3, we examine some general problems found in curve editing systems. In section 4 we outline a framework which allows multiple curve representations to co-exist and stay up to date. We then describe how we have used this framework to create a curve editing system which addresses some of the shortcomings of existing systems. Section 5 discusses new tools that the framework allows us to use in our system and Section 6 details the implementation of the individual curve rep-

[†] Currently at Microsoft Corporation
[‡] Currently at Numinous Technologies, Inc.

© The Eurographics Association and Blackwell Publishers 1998. Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 350 Main Street, Malden, MA 02148, USA.

resentations. We conclude by discussing our results and future plans.

2. Previous work

We briefly describe three common curve representations and several tools to manipulate those representations.

2.1. Common curve representations

2.1.1. Polylines

A polyline consists of an ordered list of vertices connected together by line segments.

One advantage of polylines is that they are conceptually simple since there is no level of abstraction between their mathematical and visual representations. Other advantages include the ease of adding and deleting vertices and flexibility (they can approximate any curve within a given degree of error). Additionally, polylines are amenable to filtering.

Polylines have several disadvantages. Altering a polyline by moving individual vertices can be a slow and tedious process for the user. Maintaining visual smoothness is difficult, especially while moving vertices [1]. They are C^0 and hence do not have derivatives. They can also be expensive to store.

2.1.2. Splines

A spline is defined by an ordered list of control points and a set of basis functions. Depending upon the type of spline the control points are either approximated or interpolated by a smooth, piecewise polynomial curve. Examples of approximating splines are NURBS and Uniform splines. Bézier curves interpolate their end-points and use the remaining control points to specify additional curvature information. Hermite and Beta curves both interpolate their control points and user-supplied tangents. Beta splines [2] provide additional bias and tension controls to control the shape of the curve between control points. There are several excellent books [3] [4] on splines so we will not discuss the mechanics in further detail.

Splines are useful curve representations for several reasons: locality, arbitrary smoothness, scalability, the convex hull property (NURBS), and compactness of representation. Because they are differentiable they are also amenable to mathematical operations such as minimizing tension.

The difficulties in manipulating splines stem from two properties of the underlying mathematics. First, each control point influences a fixed region of the curve. This region may be bigger or smaller than desired. In Figure 1, the user would like to change a segment of the curve corresponding to about a quarter of its total length. On the left of the figure the region of influence is too small; in order to effect the desired change, several control points must be moved. On the right of Figure 1 the region is too large and would require

Many control points Few control points

Figure 1: *Direct manipulation with splines. The amount of curve affected depends on the number of control points.*

refinement to introduce enough control points to provide the desired degrees of freedom.

Introducing additional degrees of freedom into a curve is easily accomplished using refinement. Unfortunately, the reverse operation, removing unnecessary degrees of freedom, is more difficult. Excess control points in a spline often produce unwanted “wiggles” while editing.

2.1.3. Multiresolution wavelet B-splines

A B-spline wavelet [5] [6] is a B-spline stored as a base curve and a series of wavelet detail coefficients. This representation allows for the reconstruction of the curve with an arbitrary number of control points, detail changes, and broader, global edits.

B-spline wavelets can generate representations with varying amounts of detail, allowing the user to interactively select the amount of the curve to edit. The more detail present in a curve, the smaller the area that a single control point affects. Another desirable property of wavelet B-splines is that they allow users to change the overall sweep of the curve without losing the fine details. A final benefit of B-spline wavelets is that they permit the transfer of detail from one curve to another, allowing users to add detail, such as jagged lines or bumps, to a curve without affecting its overall sweep.

The drawbacks to B-spline wavelets lie mostly in their global nature; fine detail in one part of the curve, even if it is not used elsewhere, requires an even and dense parameterization of the entire curve. Also, B-spline wavelets are a relatively new type of curve and do not have the range of tools available to other types of curves.

2.2. Common curve manipulation tools

2.2.1. Directly changing the controls of a representation

In this approach, the user manipulates the controlling functions of the representation directly; a very general (and powerful) tool but also a difficult one to use. It is also simple to

© The Eurographics Association and Blackwell Publishers 1998.

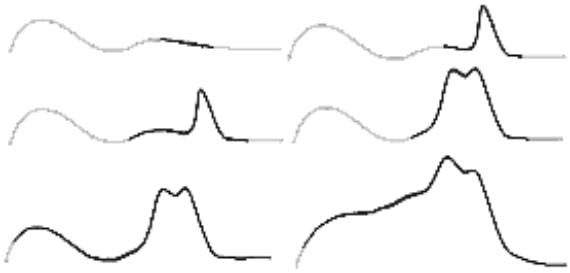

Figure 2: *Direct manipulation with wavelets. The amount of curve affected depends on the chosen level of detail coefficients. The dark area denotes the region affected by the edit.*

implement; the user picks a control point (spline representation) or vertex (polyline representation) and moves it.

2.2.2. Direct manipulation

The direct manipulation method allows the user to grab a point on the curve and move it to a new location [7] [8]. For splines and wavelets this tool is usually implemented using the least-squares technique [9] which moves the control points the smallest amount possible while ensuring that the curve passes through the desired point. In splines, the amount of curve affected is fixed and determined by the parameterization. With wavelets, however, the amount of curve affected can vary. Because of the way wavelet coefficients are organized we can choose to move the coefficients of a given level and propagate the effects to the other levels, changing the width of the effect (see Figure 2).

2.2.3. Sketching

Sketching is tool that mimics the way people draw curves on paper by allowing the user to sketch, or sketch over, a curve with a mouse or tablet pen. A curve is then fit to the resulting data points. Some additional processing may be required to remove noise in the input device and to recognize discontinuities (i.e., corners). Converting this set of data points to a spline is relatively straightforward [10] [11] and is a variation of the technique used in direct manipulation. Wavelet curves are easily fit to data points [6] and smoothing can be accomplished automatically by removing the lowest level of detail coefficients.

Prior research has extended the basic sketching technique in two ways. First, instead of taking a single set of data points, the user sketches the desired curve several times and the curve is then fit to an average of the sketches. This lets the user "home in" on a desired curve. Second, the user can sketch over a piece of the curve [12], replacing part of the curve with the new sketch. The implementation for this is somewhat more difficult because the new piece must blend in nicely with the old curve. Baudel [12] implemented this technique on piecewise Béziers; extending this technique to splines or wavelets is difficult both because of the blending problem and because the new segment may be substantially "shorter" or "longer" than the original.

2.2.4. Wavelet detail copying

Since wavelet B-spline curves are stored as a base curve with additional detail information, the detail information can be copied from one curve to another. This operation transfers the high-frequency detail without changing the overall shape of the curve [5] [6].

3. Problems with existing representations and tools

While several curve editing tools have been developed in the past, there are still several problems with existing techniques. Most of these problems stem from approaching curve manipulation by asking how *can* we manipulate curves, rather than how would we *like* to manipulate curves. An example of this is direct manipulation, which allows the user to pick a new location for a point but does not allow the user to say how the rest of the curve should accomplish this change. The method by which the curve changes is determined entirely by the underlying representation. This problem is very apparent in the spline case but also shows up to a lesser degree in wavelets. As an illustration, refer to Figure 1 where the same movement has been applied to two splines with differing numbers and locations of control points. One manipulation results in a sharp change, the other a more gradual one. This points out the need to be able to indicate how much of a curve to change as well as how to change it.

Another problem with existing editing techniques is that repeated editing of a single curve usually results in a curve with unnecessary wiggling. This happens because most editing techniques introduce more degrees of freedom to accomplish their task. Over time, these extra degrees of freedom manifest themselves as wiggles, i.e., the curve loses its initial, coherent look. There are two possible approaches to this problem; one is to design tools that do not add unnecessary degrees of freedom, the other is to provide a mechanism for removing the wiggles while keeping the desired detail. This is a user-interface problem because desired detail is often mathematically indistinguishable from noise.

Related to this problem is the one of making global adjustments to detail – for example, changing the general sweep of the curve without losing small detail on that sweep. Wavelets support this but the definition of the detail versus the sweep is still left to the representation, not the user.

We now describe an example editing session, using our system, in order to both illustrate the problems presented above and to illustrate the need for a variety of tools and representations. In this example we will use the common tools

© The Eurographics Association and Blackwell Publishers 1998.

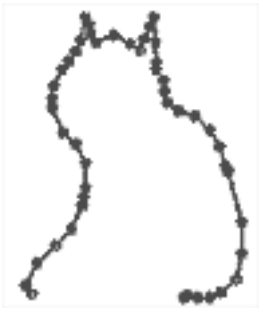

Figure 3: *Sketch tool and polyline representation.*

described above (Section 2.2) and some new tools described below in Section 5. The system automatically switches between representations as needed. We describe how this happens in detail in Section 4.

3.1. An example editing session

Problems with curve manipulation techniques generally arise when the user is trying to create a specific curve (or curves), not when they are just "trying out" a tool. To illustrate the aforementioned problems and to demonstrate our tools, we will now describe an editing session in which we try drawing a cat. This editing session uses four tools and three representations.

Refer to the sequence of Figures 3 through 7 for the following discussion. First, we sketch the basic outline of the cat using the sketching tool, sketching over the back to make the first outline. Next we use the wavelet direct manipulation tool with different widths to alter the contour. To add a tail we use the spatula tool which works on the polyline representation and easily adds more length to the curve. We use the corner tool with the spline representation to make explicit points on the ears. Finally, we draw all three representations simultaneously.

Note that we do not necessarily need to draw only the representation the tool modifies, but doing so makes the system faster. To draw a representation that is not being directly manipulated by a tool requires the framework to synchronize representations with every change to correctly render the curve.

4. An introduction to the multi-representation

No single curve representation is ideal for every situation. For example, sketching over is simple to implement with polylines but more difficult with splines. The reverse is true for the spline-based technique of least-squares manipulation – getting the same look and feel with polylines would be difficult. Since no single representation is perfect, our solution is to define a mechanism that lets the user use different

Figure 4: *Wavelet direct manipulation tool and wavelet representation*

Figure 5: *Spatula tool and polyline representation.*

Figure 6: *Corner tool and spline representation*

Figure 7: *All representations.*

© The Eurographics Association and Blackwell Publishers 1998.

representations and switch easily between them. Once this mechanism is in place the tool designer can easily add both new representations and new tools to the existing tool set and, more importantly, pick the representation that is appropriate for the tool they are designing. This mechanism must address the following concerns:

- Accuracy. How do we insure fidelity between the different representations to some given degree of accuracy?
- Speed. Interactive curve manipulation requires interactive speed; to maintain this speed we need to keep the number of conversions to a minimum.
- Ease of implementation. Adding a new tool should consist of just that; it should not require changing the other tools. Adding a new representation should be as straightforward as possible and require no knowledge of the current, existing representations.
- Change propagation and update order. When a tool changes a representation, these changes must be propagated to the other representations. This is complicated by the existence of constraints; for example, maintaining a point constraint while moving the curve. The tool moving the curve may operate on a different representation than the tool maintaining the point constraint.

There are two parts to our solution; the overall application framework for tools and representations and the exact mechanism for converting between representations. We will begin with some basic definitions and then describe the various parts of our system in more detail.

Each specific curve type (wavelet, polyline, NURBs, etc.) is called a *representation*. A collection of representations, or the conceptual curve, is referred to as the *multi-representation*. For the moment, we will treat the multi-representation as a black box which handles the synchronization and updating of the representaions, and that we can withdraw exactly one representation, edit it, and then return it. The multi-representation is contained inside of the *application framework* which also houses tools and constraints. The application framework is responsible for managing communication between these three system components. For simplicities sake we will assume there is a single curve, or multi-representation, in the following discussion.

Although we will go into more detail later, it is important to know that it is the multi-representation's responsibility to synchronize the representations present within it. At least one representation in the multi-representation is kept up-to-date at all times. Representations need not know about each other but must be parameterized on the same interval, $T \subset \Re$.

Communication between representations is done by marking the portions of the interval T which are invalid for each representation. Each connected, invalid region is called a *splice*. We can accumulate a set of these splices together to form a *splice list*. Each representation keeps its own splice list to indicate its currently invalid sections. An up-to-date curve, therefore, has an empty splice list. We formally define splices and sampling of splices in Section 4.3.

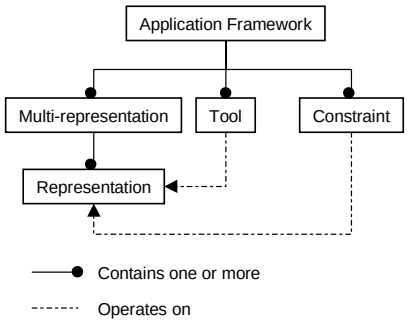

Figure 8: *A block diagram of the overall application framework. Note that the various curve representations do not communicate directly with one another.*

Updating invalid representations is accomplished by *sampling*. When a representation needs to update itself, it requests a set of sample points for its splice list from the multi-representation. The multi-representation passes this splice list to the currently valid representation, which then fills in sample points for the given intervals. The multi-representation then hands these sample points back to the updating representation.

4.1. Tools and constraints in the application framework

Unlike both the application framework and the multi-representation, which treat the different representations as black boxes, tools have full knowledge of at least one representation. When a tool is selected it first asks the multi-representation for a valid copy of the appropriate representation. The tool makes a change to the representation and then passes this change back to the multi-representation via a splice list. The multi-representation then broadcasts this splice list to the other representations.

Constraints complicate this feedback loop slightly. Constraints are implemented as tools which are always active. After a tool changes a portion of a representation it passes the changed splice to the multi-representation, which then applies any relevant constraint tools. Each constraint tool examines the affected area and, if necessary, requests samples from the current representation. The constraint can then make changes to the sample list and re-apply them to the representation. After all constraints have had a chance to re-establish themselves, and all constraint conditions are met, the tool regains control of the representation. Note that while re-establishing a constraint, a constraint tool may request an updated copy of a *different* representation than the currently

© The Eurographics Association and Blackwell Publishers 1998.

active one. For example, the corner constraint uses the B-spline representation to check that a given change is not violating an existing point and derivative constraint, and to re-establish the constraints if necessary.

One potential danger of this approach is the problem of conflicting constraints which would result in endless cycling between two (or more) constraint tools. One solution to this is to require each constraint tool to define a region over which it wants control in order to re-establish itself. For example, a point constraint at $t = t_0$ might need control over the portion of the curve $[t_0 - 0.05, t_0 + 0.05]$. The framework would therefore only allow multiple point constraints that were separated by 0.1 units in parameter space. Another solution is to define constraints of a given type in one tool, e.g., putting all of the point constraints into a single tool so they can be re-established en-masse.

Although this works for explicit constraints such as freezing parts of a curve and introducing corner discontinuities, we may need to add a constraint priority hierarchy for systems using implicit constraints such as a series of relative positional constraints.

4.2. Synchronizing representations in the framework

Our two main concerns here are speed and accuracy. One obvious approach is to maintain a base representation and switch back and forth between it and the currently desired representation. A major drawback to this approach is accuracy; even if every conversion is required to be accurate within a fixed error, over time this error accumulates. Also, rarely does a tool change the entire curve – usually only a small portion of the curve is affected. Rather than constantly update the entire representation, we would prefer to buffer the changes and then splice them in when the representation is needed. This splicing has two benefits; first, the portion of the representation outside of the changed area remains the same, and second, we can increase application efficiency by accumulating changes and delaying their application until needed. In our system representations are updated when they are rendered or just before they are modified by a tool.

As discussed before, changed intervals are represented by a splice list and the geometric changes propagated by using samples. These samples approximate the changed portion of the curve to a given error and are each associated with a parameterized t-value along the curve. To add a new curve representation to our framework, we only need to add routines for creating samples for splice lists and for splicing sampled intervals in. To maintain fidelity between the representations we require the following:

- All representations are defined on a given parameter space $T \subset \Re$, for example, $T = [0,1]$.
- Representations will be within ε of each other, i.e., if S_1 and S_2 are two representations defined on T then $||S_1(t) - S_2(t)|| < \varepsilon$ for all $t \in T$.

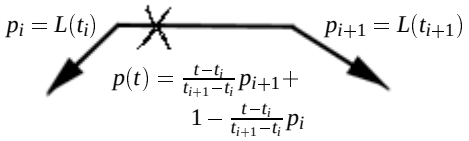

Figure 9: *Evaluating the splice sample at a given t value.*

We can maintain the above by requiring that the representation produce samples for a splice which together are within $\varepsilon/2$ of the changed area of the curve. When splicing in a change, the updated representation must be within $\varepsilon/2$ of the sampled splice. Thus, the total error for the generation and application of a splice list is $\varepsilon/2 + \varepsilon/2$ and remains less than ε We now formally define splices.

4.3. Splice implementation

We call a single, connected interval (with or without sample points) a splice. Splices are used to mark invalid intervals on the curve. A splice list is a collection of splices, each of which is disjoint from the other splices in the list. We maintain one splice list for each curve representation. Note that we delay filling in sample points for a splice until necessary.

The sampled splice is a parameterized polyline approximation that is always within an epsilon distance from the sampled curve. More formally, a sampled splice consists of an interval $[a,b] \subset T$ and a list of samples of the form $< t \subset [a,b], p \subset \Re^n >$, sorted by t. There must be at least two sample points, one with $t = a$ and one with $t = b$. Let $L : [a,b] \rightarrow \Re^n$ be defined as the linear combination of adjacent samples in the splice (see Figure 9). When a tool is applied to a representation S, producing a new representation S', the difference between S and S' is represented by a splice list made up of one or more splices. The splice list must satisfy the following:

- If $S(t) \neq S'(t)$ then t must be contained in some splice in the splice list.
- For each connected interval $[a,b]$ where $S(t) \neq S'(t)$ construct a splice. Construct the samples $< t \subset [a,b], p \subset \Re^n >$ by sampling S'. The sampling must satisfy $||S(t) - S'(t)|| < \varepsilon/2$ for all $t \in [a,b]$.

When updating a representation from a splice list we require that the resulting updated representation remain the same outside of the splice list intervals and be within $\varepsilon/2$ in the intervals. Let R be the representation before updating and let R' be the representation after updating:

- For all $t \in [a,b]$ for some splice in the splice list, $||R'(t) - L(t)|| < \varepsilon/2$.
- For all t not in some splice, $||R(t) - R'(t)|| = 0$.

© The Eurographics Association and Blackwell Publishers 1998.

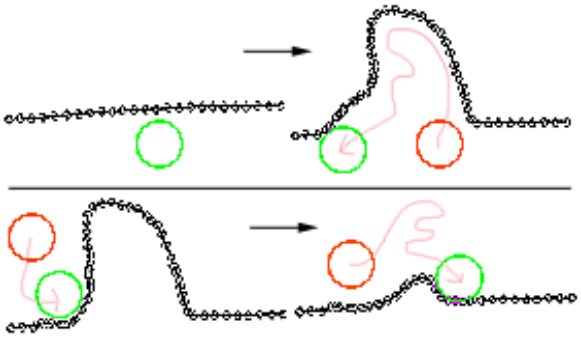

Figure 10: *Pushing the polyline around with a spatula. Path of the spatula is shown by the grey line.*

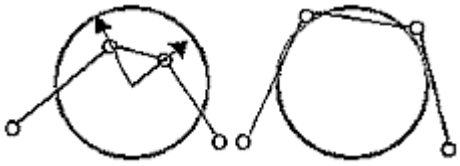

Figure 11: *Pushing the vertices of the polyline out of the circle.*

5. New tools

Here we define the three new tools mentioned in the Section 3.1, the corner tool, the spatula tool, and the freezing tool.

5.1. Pushing particles with a spatula

This is a technique for directly shaping a curve. Imagine a curve consisting of sand that can be pushed around with a round spatula. As the spatula pushes through the sand, more sand is added and as the spatula pushes sand back on itself, excess sand is removed (see Figure 10).

To implement this technique we use the polyline representation and a circle of radius r for the spatula (other shapes are possible). The length of the polyline segments where the tool contacts should be less than $r/4$ (we split the polyline segments if need be).

As the circle is moved, all points that lie in the circle are moved to lie on the closest point on the boundary of the circle (see Figure 11). If necessary, we prevent the circle from "jumping" across the polyline by sub-sampling the motion vector. If any moved segments are now longer than $r/4$ we split them in the middle. Finally, we merge any two segments whose summed length is less than $r/8$. These last two operations correspond to adding sand and removing sand.

5.1.1. Explicit corners

The corner tool is an interface for introducing a discontinuity into a spline. The user can create, change, and remove corners. The corner tool consists of a base point and two tangents that are manipulated by grabbing and pulling. To create a corner, the user clicks on a point on the curve. To remove a corner, the user double clicks on the base point of an existing corner. Single clicking selects the corner. The curve is required to pass through the base point, entering and leaving at the given tangents.

More formally, the base point constraint consists of a parameter value $t \in T$ and a point p. The spline S is required to satisfy $S(t) = p$. The derivative constraints consist of a parameter value t, a left derivative $\vec{d}_l$, and a right derivative $\vec{d}_r$. The spline is required to satisfy $\lim s \to t^- \frac{S(s)-S(t)}{s-t} = \vec{d}_l$ and $\lim s \to t^+ \frac{S(s)-S(t)}{s-t} = \vec{d}_r$.

We use approximate derivatives (within a given ε) for those representations that do not support true mathematical discontinuities.

5.2. Marking and freezing

Marking selects a region of the curve to which subsequent operations are then applied. We currently use marking to select sections of the curve to be smoothed or frozen. The marking tool consists of two marks, a left and a right one (see Figure 12). The user creates or moves marks by clicking down and moving in the direction of the desired selected area. Double clicking removes marks. All or parts of a curve can be frozen. Selection is performed by using the mark tool or by "painting" over the desired region. Frozen regions of the curve cannot be changed. The freezing tool is implemented as a constraint tool. After the current editing tool has introduced a change, the freezing tool looks at the affected region and if any of the points in the sample list are in the frozen region, it reverts them to their previous values before returning them to the multi-representation. The implementation of the marking tool is the reverse of the freezing tool; everything but the marked region is frozen.

6. Representation implementation

In this section we describe our implementation for three curve representation types; polylines, splines and wavelets. For each type we define the representation, its controls and how much of the curve they affect, how to construct a splice list for a given change, and how to incorporate the changes in a splice list.

6.1. Polylines

A polyline is an ordered list of points from 0 to $n-1$ connected together by line segments. We augment this structure

© The Eurographics Association and Blackwell Publishers 1998.

Figure 12: *Marking of a segment of a curve.*

by assigning a parameter value $t \subset T = [A,B]$ to each point. The ts must be in ascending order, starting at A and ending at B.

There are three ways to change a polyline: moving a point, adding a point, and deleting a point. When adding a point the user specifies a unique t value to indicate the position of the new point in the list. To construct a splice list, we need to know the parametric interval that is affected for each of these operations. Let i be either the index of the moved point, the index of the newly added point, or the index of the point to be deleted. Then the affected interval is $[t_{\max(i-1,0)}, t_{\min(i+1,n-1)}]$.

To generate the point list for a given splice interval we regularly sample the line segments. To guarantee that the samples are within $\varepsilon/2$ of the line segments, we sample the line segment i with any spacing $\Delta s < \varepsilon/2 \frac{t_{i+1}-t_i}{||p_{i+1}-p_i||}$.

To incorporate a splice on the interval $[a,b]$, we introduce two new points at a and b and then replace all of the points between those two points with the points in the splice list. We can optionally replace any adjacent, nearly linear splice segments with a single segment, provided that the removed point is within $\varepsilon/2$ of the new, single segment.

6.2. Splines

A non-uniform spline is defined by a degree, o, a monotonically non-decreasing knot vector $k_i, i \in [0, n-1]$ and $n-o-2$ control points. We used a degree one spline for the figures and video but the implementation supports any degree spline. Let $T = [A,B]$ be the interval the spline is defined on, i.e., $k_{o+1} = A$ and $k_{n-o-1} = B$.

There are four ways to change a spline; change the knot vector, move a point, add a control point to an end, or remove a control point from an end. Adding control points in the middle of the spline can be accomplished by *refining* the curve [13]; this changes the spline's representation but not its image. When a point is added at an end, the knot vector must be adjusted by moving the last $o+3$ knots so that the spline is still defined on $[A,B]$. Similarly, when a point is removed from an end the last $o+2$ knots must be adjusted.

Moving the control point i affects the interval delimited by the support of the corresponding basis function, i.e., $[k_i, k_{[}i+o+2]$. Adding or removing a control point affects the resulting first (or last) segment, i.e., the interval $[a, k_{o+3}]$ or $[k_{n-o-2}, b]$. Changing the knot i changes the interval $[k_{i-1}, k_{i+1}]$.

Creating a point list for a splice involves sampling the spline at fine enough intervals. We use an approximation of the arc length to guarantee that the arc length distance between two samples is less than $\varepsilon/2$.

To add in a splice we essentially add (or remove) control points in the interval and then data fit in that interval. The difficulties are guaranteeing that the spline is not affected outside of the interval and that the approximation is close enough. The data fitting routine we use is a variation of the least-squares method that includes absolute constraints – pin the curve in the adjacent, unchanged regions and fit the data points in the changed region as best as possible. This is a description of the altered least-squares method:

- $Qx = y$ where Q is an $n \times m$ matrix with $n < m$ representing the absolute constraints (the "pins").
- $Ax = b$ where A is an $l \times m$ matrix representing the desired constraints (fit the splice points).

The method returns x such that the absolute constraints are satisfied and the desired constraints are minimized in the least-squared sense [9].

We first refine the curve at the start and stop parameter points of the splice. To ensure that the spline remains unchanged outside of the interval, we must guarantee that the polynomials of the segments outside of the interval remain unchanged. This can be accomplished by requiring, for each segment, that the spline pass through $o+1$ of the original points of that segment at their original parameter values. These constraints make up the Q matrix. For each point in the splice list we add a desired constraint into the A matrix. We now add (or remove) control points into the interval based on the number of data points in the splice (roughly one control point for every twenty samples). Note that we can remove control points from the interval provided that their support is entirely contained in that interval. We next alter the knot vector within the interval so that it reflects the spacing in the splice list. We now perform the following iteration:

- Data fit
- If the data fitting routine was unable to find a solution (i.e., it could not satisfy the absolute constraints) refine at the end segments of the interval and continue.
- Check that the distance to the curve for each sample point is less than $\varepsilon/2$.
- Refine segments for which the above did not hold and continue.
- If all distances were within the bounds, we are finished.

© The Eurographics Association and Blackwell Publishers 1998.

6.3. Wavelets

We use the B-spline wavelet representation described in [6]. This representation uses a four control-point B-spline as its base curve and a set of wavelet detail coefficients to describe additional levels of detail. The detail coefficients are arranged into levels, each level having twice as many coefficients as the level below it. For each level l of detail we can construct a B-spline which has $2^l + 3$ control points. As the number of control points in the B-spline increases (i.e., as we add more levels of detail) the amount of the curve affected by a single control point decreases. Since the number of control points doubles at each level the affect of a detail coefficient at level l is $1/2^l$ of the total parameter space. For a more precise definition and information on editing fractional levels, see [6 5].

There are two ways to change a B-spline wavelet; one, add another layer of detail coefficients, and two, change a detail coefficient. Changing a detail coefficient at level l and position i affects the interval $[i\frac{1}{2^l}, (i+1)\frac{1}{2^l}]$ (assuming $T = [0,1]$). Adding more detail coefficients does not change the image of the curve as long as the new detail coefficients are zero.

In general, B-spline wavelets behave best if they are evenly parameterized, i.e., equal steps in parameter space result in equal steps along the image of the curve. When the B-spline is initially fit to some data, and whenever we splice in a change, we try to maintain this even parameterization. In addition to keeping the behavior of the curve consistent, this also simplifies sampling.

To create a sample list we sample the B-spline evenly in parameter space. We must chose a sampling rate, Δt, so that the samples are spaced less than $\varepsilon/2$ apart. Let p_i and p_{i+1} be the closest-spaced control points on the interval in question (using the B-spline created using all of the detail coefficients) and let the wavelet have l levels of detail. Then a conservative estimate for Δt is $\frac{10*2||p_{i+1}-p_i||}{\varepsilon} * \frac{1}{2^l}$ (assuming again that $T = [0,1]$).

To splice in a change, we first reparameterize the interval in question. If we cannot fit the curve to within $\varepsilon/2$ in the given interval we add another layer of detail coefficients to the entire curve and try again until successful. Since we only move the new detail coefficients in the splice interval we will not affect the curve outside of the interval. Note that adding another layer of detail coefficients results in double the number of coefficients, most of which will be zero. This is where a sparse representation would be useful.

7. Results and future work

We have demonstrated the synchronized, lazy evaluation editing of several curve representations. The interface runs in real-time on an Intel Pentium Pro 200. In general, interaction times are not affected if the display representation is the same as the representation being edited. Updating while editing is slightly slower. Curve sizes range from 5 to 50 control points, or on the order of 100 vertices.

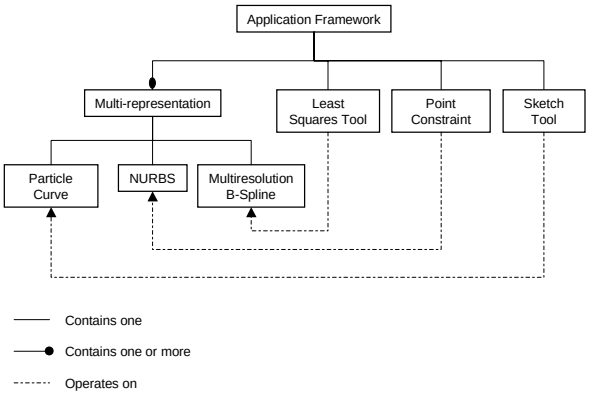

Figure 13: *A block diagram of our implementation.*

Currently, the acceptable approximation error (ε) is fixed; this sometimes results in lost detail or an excessive number of points. It should be possible for the user to set this value.

We would like to use the system to explore more user interface techniques, especially ones which focus on supplying better selection and editing control to the user. Other tools, especially controllable smoothing techniques, are needed. We would like to experiment with gestural selection and application of tools.

The framework as described extends directly to surfaces with one major difficulty; unlike curves, there is no single parameter space (or any parameter space in some cases) shared by all surface representations. One possible solution to the parameterization problem is to use an abstract manifold; representations would then need to provide their own mapping to the manifold *and* be able to change the manifold to reflect topological changes. Fortunately, almost all surface representations are capable of producing a polygonal approximation; this polygonization could be altered to produce a manifold.

References

1. C. Grimm, D. Pugmire, M. Bloomenthal, J. Hughes, and E. Cohen, "Visual interfaces for solids modeling", *UIST '95*, pp. 51–61 (1995).
2. B. Barsky, "The beta-spline: A curve and surface represenation for computer graphics and computer aided geometric designs", *In a book by the International Summer Institute Springer-Verlag, NY*, (1896).
3. R. Bartels, J. Beatty, and B. Barsky, *An Introduction to*

© The Eurographics Association and Blackwell Publishers 1998.

Splines for Use in Computer Graphics and Geometric Modeling. Morgan Kaufmann, (1987).

4. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, (1988).
5. E. Stollnitz, T. DeRose, and D. Salesin, *Wavelets for Computer Graphics: theory and applications*. Morgan Kaufmann, (1996).
6. A. Finkelstein and D. Salesin, "Multiresolution curves", *Computer Graphics*, **28**(2), pp. 261–268 (1994).
7. B. Fowler and R. H. Bartels, "Constraint based curve manipulation", *Siggraph course notes 25*, (1991).
8. B. Fowler, "Geometric manipulation of tensor product surfaces", *Symposium on interactive 3D graphics*, **25**(2), pp. 101–108 (1992).
9. Strang, *Linear Algebra and its Application*. HBJ, (1988).
10. M. J. Banks and E. Cohen, "realtime spline curves from interactively sketched data", *Computer Graphics*, **24**(2), pp. 99–107 (1990).
11. P. H. Schneider, "an algorithm for automatically fitting digitized curves", *Graphics Gems*, (1990).
12. T. Baudel, "A mark-based interaction paradigm for freehand drawing", *UIST '94 proceedings*, pp. 185–192 (1994).
13. E. Cohen, T. Lyche, and R. Riesenfeld, "Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics", *Computer Gr. Image Process.*, **14**, pp. 87–111 (1989).
14. D. Forsey and R. Bartels, "Hierarchical b-spline refinement", *Computer Graphics*, **22**(2), pp. 205–212 (1988). Proceedings of SIGGRAPH '88.

© The Eurographics Association and Blackwell Publishers 1998.