

A Framework for the Synchronous Editing  
of Multiple Curve Representations

by

Matthew R. Ayers

A. B., Brown University, 1995

Sc. B., Brown University, 1995

Project Report

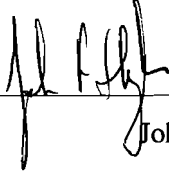
Submitted in partial fulfillment of the requirements for the  
Degree of Masters of Science in the Department of Computer Science  
at Brown University.

May 1998

Copyright  
by  
Matthew R. Ayers  
1998

This report by Matthew R. Ayers  
is accepted in its present form  
by the Department of Computer Science  
as satisfying the project requirement  
for the degree of Masters of Science.

Date 5/4/98

  
\_\_\_\_\_  
John F. Hughes

# Acknowledgments

Many thanks to John Hughes and Cindy Grimm for their insightful and patient advising. I'd like to acknowledge the support of Andy van Dam and the Brown Computer Graphics Group. Lastly, I'd like to thank the folks at Numinous Technologies for their ideas, encouragement, and senses of humor.

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>List of Figures</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Definition of Terms . . . . .	2
1.2 Roadmap . . . . .	3
<b>2 Prior Work</b>	<b>4</b>
2.1 Splines . . . . .	4
2.2 Particle Systems . . . . .	5
2.3 Multiresolution B-Splines . . . . .	5
<b>3 Curve Representations</b>	<b>7</b>
3.1 Curve Structure . . . . .	7
3.2 Samples and Splices . . . . .	7
3.3 Updating MCECurve Sub-Representations . . . . .	8
<b>4 Tools</b>	<b>9</b>
4.1 Least Squares Manipulation Tool . . . . .	9
4.2 Sketch Tool . . . . .	9
4.3 Push Tool . . . . .	10
4.4 Direct Manipulation . . . . .	11
<b>5 Constraints</b>	<b>12</b>
5.1 Constraint Description . . . . .	12
5.2 Point Constraints . . . . .	13

5.3	Segment Constraints . . . . .	13
5.4	Corner Constraints . . . . .	13
<b>6</b>	<b>Framework</b>	<b>16</b>
6.1	General Structure . . . . .	16
6.2	Curves and Representations . . . . .	17
6.3	Tools and Constraints . . . . .	18
<b>7</b>	<b>Implementation</b>	<b>19</b>
7.1	Application Framework . . . . .	19
7.1.1	Representation Factories . . . . .	19
7.1.2	The Curve Factory . . . . .	20
7.1.3	Tool and Constraint Interfaces . . . . .	20
7.2	Curves and Representations . . . . .	21
<b>8</b>	<b>Limitations and Future Work</b>	<b>22</b>
8.1	System Improvements . . . . .	22
8.1.1	Tools and Constraints . . . . .	22
8.1.2	User Interface . . . . .	23
8.1.3	Representations . . . . .	23
8.2	Other Mathematical Representations . . . . .	23
	<b>Bibliography</b>	<b>25</b>

# List of Figures

4.1	Using the Sketch Tool. . . . .	10
4.2	Using the Push Tool. . . . .	11
5.1	A 'spike' caused by a point constraint. Notice the sharp transitions indicated by the blue arrows. . . . .	14
5.2	The active corner tool and the resulting curve. . . . .	14
6.1	The Application Framework. . . . .	17
7.1	An example Application Framework. . . . .	20

## **Abstract**

Editing curves and surfaces is difficult in part because the mathematical representations rarely correspond to the mental idea most people have 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, if not impossible. A system using a single representation must therefore either limit the types of tools available or convert existing tools to a possibly unsuitable curve or surface representation.

This paper presents a framework for editing curves which supports multiple representations and ensures that they stay synchronized. As a proof of concept, I have created a curve editor which contains several tools each of which manipulate one of three different curve representations: polylines, NURBs, and multiresolution B-splines.



# Chapter 1

## Introduction

Although drawing a curve using a pen and paper is very straightforward, reproducing this simplicity with computer curves has proved an elusive goal. To date, curve manipulation has been driven by a curve's mathematical representation rather than by the tasks that users wish to perform. This results in tools and systems which, although well matched to a particular mathematical representation, are not well suited for real world use. This paper presents a system for unifying disparate curve representations as well as the tools that manipulate them.

There are two possible solutions to the problem of making different tools work together: change the tools or change the representations. Changing the tools is difficult; each tool was developed because of the ease with which it manipulates a given representation. It is usually difficult, if not impossible, to rework a tool for a mathematical representation that it was not designed to manipulate.

Alternatively, one could create a system which supports a variety of curve representations and provides a way to update them all when one changes. This allows tool designers to use the representation most amenable to their particular interaction technique. This has a limited utility, however, because updating all curve representations after every change becomes time consuming as either the number or complexity of the curve representations increases.

In the system presented here, I address both the problems of using tools designed for different mathematical representations as well as keeping the system efficient for large, complex curves. The system aggregates multiple mathematical representations into a single conceptual curve which lazily synchronizes these sub-representations. The system also allows tools to directly access the sub-representations on which they were designed to work. Thus tool designers

can design tools which work effectively on a given mathematical curve representation and integrate them into a broader system without compromising efficiency. Much of the work the work described in the paper was done jointly with Cindy Grimm, now at Microsoft Corporation.

## 1.1 Definition of Terms

Here are definitions of some of the terms used in this paper.

- **Curve** – A curve is a conceptual curve. It is the object that used to be drawn with a pencil that we are trying to recreate in the computer.
- **Representation** – A representation is a specific, mathematical implementation of a curve. Examples of representations are splines and particle systems.
- **Sub-representation** – A sub-representation is a representation contained in an MCECurve (see below). It conforms to the interface described in section 3.1.
- **MCECurve** – An MCECurve is my implementation of a conceptual curve. An MCECurve comprises one or more sub-representations which, when synchronized, have identical geometries. For example, an MCECurve may contain a b-spline and a particle system.
- **Sample** – A sample is a point on a curve represented by a  $t$  value in the interval  $[0, 1]$  and the point associated with that  $t$  value. Samples are used by sub-representations to communicate changes in their structure.
- **Splice** – A splice list consists of two parts: an interval,  $I$ , which is a subset of the interval  $[0, 1]$ , and a set of splices which represent sample points along that interval. For any  $t$  value in  $I$ , the splice list can provide a corresponding point.
- **Application Framework** - The object which contains MCECurves, tools and constraints. It also manages the interaction between these components and routes system events.
- **Tool** - An object which the user employs to directly or indirectly manipulate the mathematical structure of a representation.
- **Constraint** - An object which the user places on a part of a representation to restrict the way in which a tool may modify it.

## **1.2 Roadmap**

In chapter 2, Prior Work, I detail the structure, strengths, and weaknesses of the existing curve representations used by the system. The next chapter, Curve Representations, describes the structure of the curve representations used in this system, including how to update a representation or portion of a representation. Chapter 4 describes the various tools in the system and how they interact with a particular curve representation. Constraints, a specific type of tool, are described in chapter 5. Chapter 6 describes how the various components of the system interact with one another with the specific implementation details described in chapter 7. The final chapter of the report addresses limitations and the various possibilities for future research.

## Chapter 2

# Prior Work

This section provides a general description of the three representations used in the system, presenting the benefits and drawbacks to each, with specific implementation details described in section 3.1.

### 2.1 Splines

A spline is defined by an ordered list of control points. Depending upon the type of spline these control points are either approximated or interpolated by a smooth, piecewise polynomial curve [1]. Examples of approximating splines are NURBS and uniform B-splines. Bézier curves interpolate their end-points and use the remaining control points to specify additional curvature information. Hermite and Beta curves interpolate both 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.

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

The major limitation of traditional splines is that each control point influences a fixed region of the curve. This region may be bigger or smaller than desired. To change the amount of the curve that a control point affects, you must permanently change the knot vector or degree of the curve. Refinement [3] can be used to introduce more degrees of freedom into the curve. Unfortunately, removing excess degrees of freedom is difficult. Too many control points in a

spline often introduces unwanted wiggles during editing.

Another limitation of splines stems from the separation of their visual and mathematical representations. To manipulate a spline in a way that makes sense to a user, a tool must often make assumptions about the user's intentions and often offers little user control of these assumptions. This is the case with the Least Squares Manipulation Tool described in section 4.1.

## 2.2 Particle Systems

A particle system representation is a collection of vertices connected by straight lines. The parameterized  $t$  value for a given control point, is the sum of the length of the preceding segments divided by the total length of all the segments. The  $t$  value for a point on a segment can be approximated by the weighted average of the point's two nearest neighbors.

Particle systems have the advantage of being very simple; there is no abstraction between a particle systems visual and mathematical representation. This makes it very easy to add or remove detail and keep the changes local. The lack of abstraction also makes it very easy to directly manipulate a particle system's mathematical representation.

Unfortunately, the simplicity of particle systems gives them several undesirable properties, as well. At each vertex, a particle system representation has a discontinuity. Thus, a curve composed of particles has no derivative. Since the geometry of particle systems is composed of straight lines, you must use several vertices to approximate curves and the amount of storage needed for a particle system is directly proportional to its level of detail. In addition, detail must be recomputed when changing resolutions.

## 2.3 Multiresolution B-Splines

Like splines, multiresolution B-splines [4] [5] are a set of functions controlled by a set of control points. Unlike standard B-splines, multiresolution B-splines are represented at various levels of detail by wavelet coefficients. By interpolating between these fixed levels of detail, multiresolution B-splines whose most detailed representation contains  $n$  control points can be represented by curves with between 4 and  $n$  control points. Thus, moving a single control point can affect anywhere from all of the curve to  $(\frac{1}{n-3} * 100)\%$  of the curve.

The major advantage of multiresolution B-splines over traditional splines is the ability to

dynamically reparameterize the curve without permanently changing the underlying mathematical representation. Thus, you can make both broad sweeping changes as well as fine detail manipulations. In addition, because the curves are stored as a base curve with a set of detail coefficients, you can copy high-frequency detail from one curve to another, without affecting the overall sweep.

Other benefits of multiresolution B-splines are similar to splines: they have  $C^2$  continuity and are resolution independent. multiresolution B-splines are also extremely compact, requiring only slightly more storage than traditional B-splines.

Like all representations, multiresolution B-splines are imperfect. Many of the operations affecting detail are global in nature and thus, do not integrate easily into a task-oriented editing system. Also global in nature is the storage required for the control points. Detail is uniform along the entire curve, so the portion of the curve with the finest detail determines the number of control points present in the entire curve.

Another drawback to multiresolution B-splines is that the underlying mathematical representation is hard to visually relate to the position of the control points. This makes direct manipulation of the control points both difficult and unpredictable. Most interaction techniques, therefore require the computer to make some assumptions about the user's intentions. Finally, the multiresolution B-spline is a new representation and tool designers have not had an opportunity to fully explore its potential. Thus, the number of available tools is still fairly limited.

## Chapter 3

# Curve Representations

In this chapter, I will give the details of the interfaces and restrictions imposed on representations used in this system and the methods by which curves remain synchronized.

### 3.1 Curve Structure

Although the system supports any type of curve representation, every representation in the system must have the following properties:

- The representation must be parameterized on the interval  $[0, 1] \rightarrow \mathfrak{R}^n$ .
- Given a value of  $t$  in the interval  $[0, 1]$ , the representation must be able to provide the corresponding point on the curve.
- Given an interval  $I \subset [0, 1]$ , the representation must be able to provide a splice containing enough samples for other representations to reconstruct the interval.
- Given a splice (see below) over the interval,  $I$ , the curve must be able to alter itself such that for any  $t$  value,  $t$ , in  $I$ ,  $curve(t) = splice(t)$  and that the curve is unchanged outside of  $I$ . This ability is encapsulated in the representation function *UpdateInterval()*.

### 3.2 Samples and Splices

A sample represents a point on a representation and consists of a parameterized  $t$  value,  $t$ , and a corresponding point  $p$ . Samples are contained in splices which are similar to particle systems, except that they cover an interval  $[t_0, t_1] \subset [0, 1]$ . Curves use splices to convey the changes made

over an interval. Like particle systems, if a splice covering an interval,  $I$ , does not contain a sample for a given  $t$  value  $t$ , it will linearly interpolate between corresponding points for the two  $t$  values which bracket  $t$ .

### 3.3 Updating MCECurve Sub-Representations

MCECurve sub-representations use splices to communicate changes to one another. When a sub-representation in an MCECurve is altered, it broadcasts the interval containing the changes to the other sub-representations in the MCECurve. Each sub-representation keeps its own update list of disjoint intervals between  $[0, 1]$  which are out of date (note that these intervals are just markers, a sub-representation uses a splice to actually get samples). If another sub-representation broadcasts a change interval which overlaps an interval already in a sub-representation's update list, the union of the two intervals is stored in the update list. To ensure that at least one sub-representation in a curve is up to date at all times, a sub-representation can not be modified until its update list is empty.

When a sub-representation needs to update itself, it iterates over its outdated intervals and, for each disjoint interval, does the following:

- The sub-representation requests a splice from an up-to-date sub-representation
- The sub-representation uses the *UpdateInterval()* function to change its own geometry over the interval.
- The sub-representation removes the newly validated interval from its update list.

Note that no matter how many times the points in a given interval change, sub-representations only keep track of the interval until it is time to update; splices are only generated for the *UpdateInterval* and are not automatically broadcast when a representation changes.



# Chapter 4

## Tools

This chapter describes the tools used to manipulate the various representations in the system.

### 4.1 Least Squares Manipulation Tool

The least squares manipulation tool [6] [7] allows a user to grab a point on a spline curve and move it to a new location. The tool manipulates the underlying control points to create the change. Since there are four control points affecting any one point on the curve, this is an under-constrained problem. The additional constraint of minimizing the distance that the control points travel is added to define the exact manner in which the control points move.

When working with a multiresolution B-spline, the user can change the area that is affected by moving a single point. This is done by using the wavelet detail coefficients to create a temporary b-spline with the appropriate number of control points which can be edited and then reconverted into wavelet coefficients. This makes the least squares manipulation tool much more powerful, allowing users to change the overall sweep as well as fine details of a curve.

### 4.2 Sketch Tool

The sketch tool [8] is a digital pencil which works with a particle system. The user sketches a new line,  $L$ , over the part of the curve that he would like to replace. The system finds the  $t$  values on the curve closest to the endpoints of  $L$ ,  $t_1$  and  $t_2$ , and then splices  $L$  into the curve, replacing the interval  $[t_1, t_2]$ . If the closest point to one of the endpoints is also an endpoint of the curve, the system assumes that the user wishes to extend the curve and splices onto the end of the curve, as appropriate.

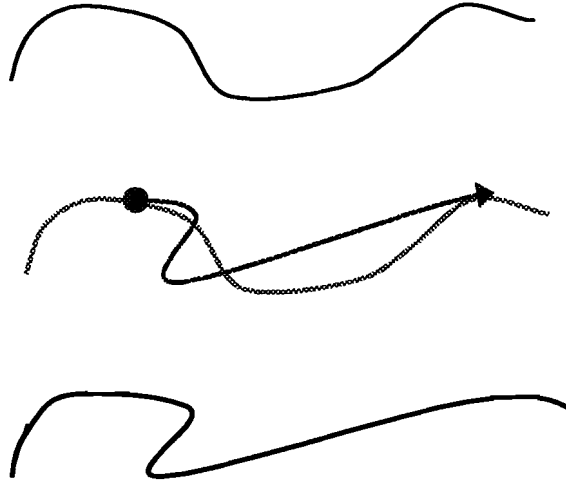


Figure 4.1: Using the Sketch Tool.

### 4.3 Push Tool

Like the sketch tool, the push tool works on particle systems. The effect is similar to pushing a mold up against an elastic band, except that when the mold is removed, the curve retains its shape. A push tool can be any shape with a well defined interior and exterior. As the user moves the tool, the tool checks to see if any particle in the curve has crossed the tool's boundary and lies inside the tool. If this happens, the push tool finds the point on its boundary closest to the particle and moves the particle to that location.

After moving a particle, the push tool checks the line segment that it forms with each of its neighboring particles. If the midpoint of the line segment lies within the tool's boundary, a new particle is added to the particle system at the midpoint and moved outside the tool. The tool also checks the newly moved particle to see if it lies in a straight line with its neighbors. If it does, the newly moved particle is removed from the particle system. In practice, for efficiency, these tests are not exact; if a point inside the boundary lies within a certain distance  $\epsilon$  of the boundary, it is considered to be outside. This method works well for small tool motions; large motions make the tool appear to push through the curve. Thus, in this system, the application framework breaks a large mouse motion into several small ones.

For this project, I have created a circular push tool, because it is simple to test if a point has crossed the boundary. Any shape with a defined interior and exterior will work, however,

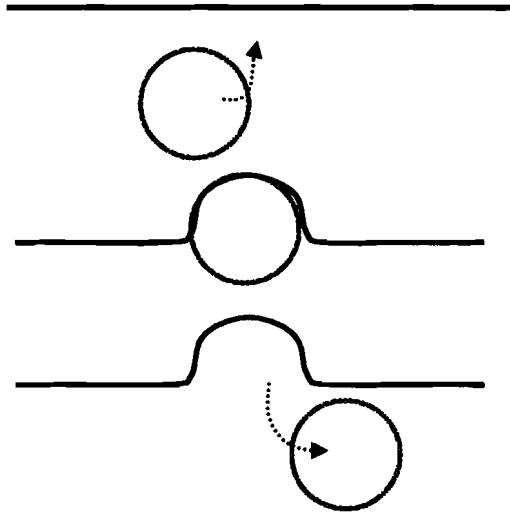


Figure 4.2: Using the Push Tool.

as long as you can find the point on its boundary closest to any arbitrary point.

#### 4.4 Direct Manipulation

Some would argue that the most efficient way to manipulate any curve is to directly alter the mathematical structure. While this offers the greatest degree of control for expert users, it is often extremely non-intuitive for the novice. Since the focus of this system is on the task, not the mathematics, I chose not to implement this tool in favor of tools which are more intuitive for the non-mathematical user.

# Chapter 5

## Constraints

In this chapter, I will describe constraints that can be placed on curves to restrict or guide the effect of the various tools.

### 5.1 Constraint Description

A constraint is a way for a user to restrict the way that a tool modifies a given interval. Constraints are similar to tools, except that a constraint's primary method of communicating with a curve is through a splice list. A constraint can access any specific sub-representation for information about a curve, but needs to be able to convey its restrictions to any arbitrary sub-representation via a splice list.

After a tool modifies a sub-representation, the system allows each constraint to examine the modified interval. If the new curve violates a constraint, the constraint will correct the curve by generating a splice list, over the violating interval, containing a valid set of splices. The sub-representation then incorporates the splice lists generated by the constraints using the *UpdateInterval()* function and finally broadcasts the changed interval to the other sub-representations.

It is possible for two separate constraints which influence the same point or interval to provide conflicting information when enforcing their constraints. To avoid this, the system imposes the restriction that only one constraint can affect a given interval.

## 5.2 Point Constraints

A point constraint is like a thumbtack; it pins a point on the curve in place. Each point constraint contains a  $t$  value,  $t_c$ , and a point,  $p_c$ . To validate a representation, the constraint asks the representation for the point at  $t_c$ . If the resulting point  $p_r$  is not within some  $\epsilon$  of  $p_c$ , the constraint generates a splice using the pair  $t_c$  and  $t_p$ .

This implementation of point constraints leaves the position of the neighboring points up to the representation. Applied to a particle system, this constraint usually creates a spike (figure 5.1), because it fixes one particle in place. With splines, the transition to the constrained point is more gradual. Note that the other sub-representations, when they update, will mimic the behavior of the constrained representation, but the shape is governed by the sub-representations on which the constraint was initially placed. To ensure a more consistent transition over sub-representations, the constraint could return a splice containing not only the pair  $(t_c, p_c)$ , but surrounding  $t$  values and points as well.

## 5.3 Segment Constraints

Segment constraints, generated by the freeze tool, are similar to point constraints except that they freeze an entire interval in place, rather than just one point. A segment constraint consists of an interval  $I_c$ , and a list of  $n$  representative  $[t \text{ value}, \text{point}]$  pairs. Similar to point lists, when validating a curve, a segment constraint checks if the modified interval,  $I_m$  overlaps with  $I_c$ . If so, the tool takes the interval spanning the intersection of  $I_m$  and  $I_c$  and creates a splice list from the  $t$  value, point pairs in its splice list that lie in that interval.

As with point constraints, just supplying a splice list to replace the frozen interval can result in widely varying transitions between frozen and unfrozen parts of the representation. A blending function in the constraint tool can make this more consistent.

The inverse of the freeze tool is the affected region tool which allows the user to mark the active interval,  $I_a$  that a tool affects. It is implemented by applying segment constraints to the compliment of  $I_a$ .

## 5.4 Corner Constraints

Corner tools (figure 5.2) allow users to put a sharp corner, or discontinuity, into an otherwise smoothly changing curve. To create a corner, the user sets the location of the corner which

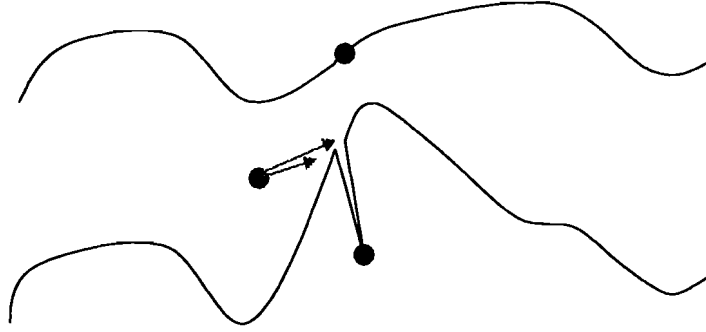


Figure 5.1: A 'spike' caused by a point constraint. Notice the sharp transitions indicated by the blue arrows.

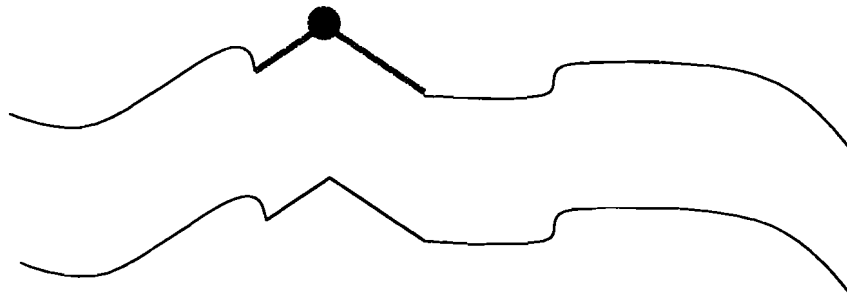


Figure 5.2: The active corner tool and the resulting curve.

places a vertex and two arms at a point on the curve. The user can rotate these arms like the hands on a clock to specify the angle of the corner as well as the length that it affects on each side.

Internally, the corner tool is similar to the freeze tool in that it affects a range of the curve. It consists of an interval,  $I_c$ , a hinge point,  $p_h$ , and two endpoints. Each of these points has an associated  $t$  value.

When a representation changes, the corner tool first calculates the distance between  $p_h$  and each of the endpoints,  $\Delta_0$  and  $\Delta_1$ . The constraint then asks the representation for the new point at the  $t$  value associated with  $p_h$ . This then becomes the new hinge point,  $p_h$ . The endpoints are then calculated as  $p_h + \Delta_0$  and  $p_h + \Delta_1$ . Thus the corner retains its magnitude, but its position changes.

This is not the only way to implement the corner constraint. Other possibilities are to,

rather than keeping the  $t$  value of the hinge fixed, try to move the hinge point as little as possible, allowing the  $t$  value to vary. Alternatively, the user may want to pin the end points of the corner, and allow the angle to change. All these are valid constraints, which implement a corner. The method used in this system seemed, to the author, to offer the best tradeoffs between utility and complexity.

## Chapter 6

# Framework

The framework is a general term to describe the mechanisms used to control and synchronize the communication between MCECurves, tools, and constraints. This section describes the structure and communication used by these objects as well as the communication between the various sub-representations, governed by the MCECurve.

### 6.1 General Structure

The flow of control in this system loosely mirrors the Model-View-Controller (MVC) structure found in Smalltalk [9]. All of the data is contained in the model object known as the *application framework* (figure 6.1). The application framework contains a list of MCECurves, a set of tools and a list of active constraints. It is the application framework which facilitates communication between these three major components. For each user generated event, the application framework does the following.

- The application framework passes the event to the currently active tool
- The currently active tool modifies a sub-representation and reports changed intervals to the MCECurve
- Before the MCECurve passes the changed intervals to the other sub-representations, the application framework checks them against the current constraints.
- If any constraints generate a splice list, pass that along to the recently changed sub-representation



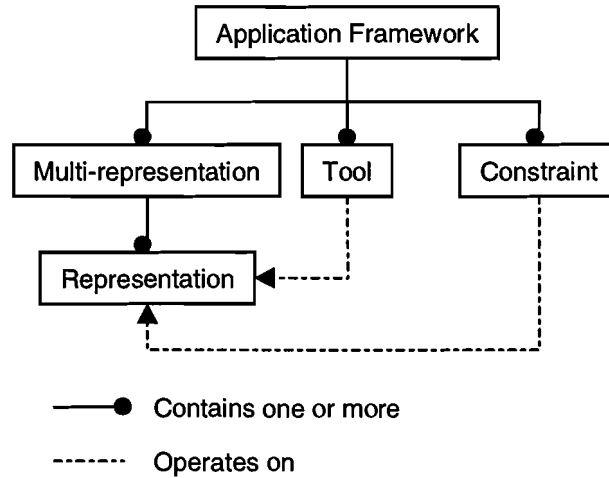


Figure 6.1: The Application Framework.

- Have the MCECurve broadcast the changes to the other sub-representations

## 6.2 Curves and Representations

As defined in the introduction, an MCECurve is an implementation of a curve comprised of one or more sub-representations. It is the MCECurve's job to ensure that all the sub-representations stay synchronized. Although tools and constraints can access sub-representations directly, they must do so through the MCECurve which contains them. When an MCECurve receives a request for one of its sub-representations, it checks if the sub-representation is up to date. If the sub-representation contains invalid intervals, the MCECurve finds a sub-representation which is up to date, and generates splice lists for the outdated sub-representation. The outdated sub-representation updates itself via the splice lists. Finally, the MCECurve returns the up-to-date sub-representation to the requesting object. Requests most frequently occur when a tool wishes to modify a sub-representation, or when the application framework needs to draw a sub-representation.

### 6.3 Tools and Constraints

Although they are contained by the application framework, tools operate directly on the sub-representations contained in an MCECurve. The application framework keeps track of the currently active tool and when it receives an event, calls the appropriate event handler on the currently active tool. The event handler on a tool requests an appropriate sub-representation from the currently active MCECurve and makes the necessary modifications. It then informs the MCECurve which interval has changed and on which sub-representation.

When it receives the message that a tool has modified a sub-representation  $R$  over the interval  $I$ , the parent MCECurve requests a list of constraints from the application framework and uses them to validate  $R$ , as described in chapter 5. After  $R$  passes all constraints, the MCECurve invalidates  $I$  for all sub-representations other than  $R$ .

## Chapter 7

# Implementation

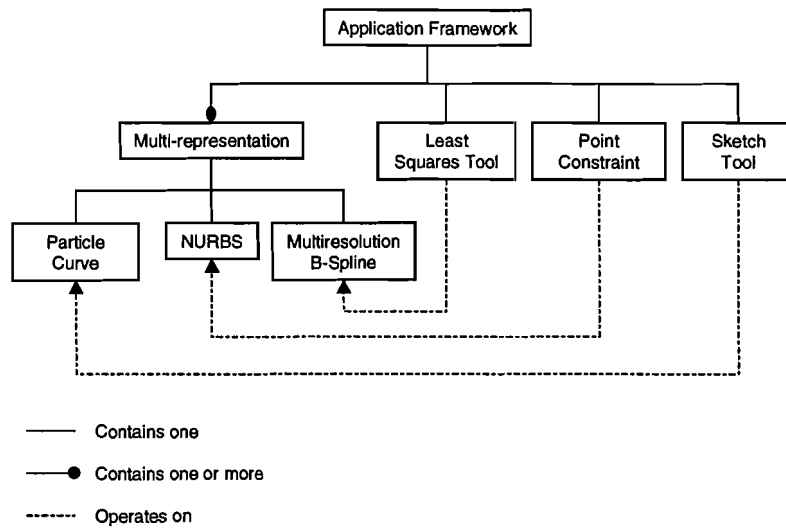
The current design for the system was not born in a vacuum. The first attempt at unifying curves was clumsy at best. Due to various vagaries of the interaction framework used, the curve inherited from all associated representations and used all the methods found therein. This made the interface large, slowed the system, and made it difficult to understand, much less extend. In this section, I will discuss the specific interfaces required to add a new sub-representation to the system as well as the objects which use this interface.

### 7.1 Application Framework

One of the design goals for this system was extensibility. In the interest of working with several curve representations, it should be easy to add new representations to an existing system. To facilitate this, the system uses factories [10] to generate new MCECurves and sub-representations. Note that the prefix MCE indicates a class implemented in the system.

#### 7.1.1 Representation Factories

A *representation factory* is an object which generates a specific curve representation. For each representation in the system, there is exactly one MCERepresentationFactory subclass which will create both an empty representation and will create a representation from a list of samples. At system initialization, each sub-representation type registers itself with the *curve factory* (see below) by passing it an ID and a pointer to the appropriate representation factory. This ID is also used by the MCECurve to provide access to an instantiation of a sub-representation (when requested by a tool, for example).



Before After

Figure 7.1: An example Application Framework.

### 7.1.2 The Curve Factory

The curve factory is a singleton object [10] which uses representation factories to create new MCECurves. The Curve Factory contains a list of Representation Factories and their associated IDs. Rather than calling *operator new()* on a subclass of MCECurve, the application framework calls the *CreateCurve()* method on the static MCECurveFactory object. The Curve Factory creates a new MCECurve, iterates over the registered Representation Factories and requests a new sub-representation from each which it places in the newly created MCECurve. The Curve Factory then returns the new MCECurve.

The alternative is to use a subclass of MCECurve for each combination of sub-representations and changing the source code everywhere that the system wants to create a curve. With this design, the system only has to register the new representation and provide an appropriate factory. This localizes creation of MCECurves and sub-representations and frees the application framework to work with an abstract MCECurve rather than knowing about a specific subclass.

### 7.1.3 Tool and Constraint Interfaces

To interact with the application framework, an MCETool must be able to handle system events, activate and deactivate itself and render itself. MCETools are created with a representation ID

as a parameter. This should be provided by any subclass of MCETool so that the application framework can ensure that the sub-representation has registered itself with the MCECurveFactory. Constraints have the same interface, but with an additional method which validates a sub-representation. This method takes an MCECurve and returns a splice (since constraints are associated with a sub-representation, they can request that sub-representation from the MCECurve that is passed in).

## **7.2 Curves and Representations**

The system implements three types of sub-representations, none of which were developed specifically for this system: traditional NURBS splines, Particle Systems, and multiresolution B-splines. The NURBS and Particle representations were salvaged from the previous implementation of this system, and the multiresolution B-splines implementation is used under license from Numinous Technologies. I decided to use existing representations for two reasons. First, it is much easier not to reinvent the wheel. Secondly, and most importantly, none of the implementations were modified to fit the system. Shell classes or subclasses were used to provide the specific interfaces required by the application framework, but no major modifications were required to integrate these representations into the system.

## Chapter 8

# Limitations and Future Work

Although the system presented here is a good introduction to synchronized curve editing, it is by no means perfect. This chapter discusses possible system improvements as well as extensions to the basic ideas contained in the report.

### 8.1 System Improvements

#### 8.1.1 Tools and Constraints

The most obvious improvements to the system deal with expanding the types and power of the tools available to the user. As far as tools go, I would like to experiment with different shapes for the push tool ranging from simple lines to complex, user-defined tools. I would also like to experiment with the least squares tool and change the way that it solves the under-constrained equation which manipulates the control points. Expanding the variety of push tool shapes in the system would also make it more powerful.

As to constraints, I would like to implement a variety of blending functions which provide a more graceful fall-off for the point and segment constraints. Once again a user-controlled blending function for these constraints would be ideal. I would also like to implement linking constraints, similar to those pioneered by Michael Gleicher [11], to pin curves to one another rather than a fixed point in space. Finally, the system needs a graceful arbitration routine to deal with sets of conflicting constraints. The current method of making constraints work only on non-overlapping intervals is very limiting.

### **8.1.2 User Interface**

The most interesting user interface improvements to this system involve providing more control over the assumptions that the system makes about the user's intention. For example, with the corner constraint described in section 5.4, the user should be able to not only specify the desired corner variety, but easily iterate over the possible alternatives to the one she has chosen. To give more power to the user, I would like to implement interactive control over the blending functions used in the point and interval constraints. I would also be interested in non-tradition curve manipulation techniques involving gesture recognition and sketching.

### **8.1.3 Representations**

The current multiresolution B-spline implementation is fairly primitive. It would be nice to implement a sparse version of this representation so that areas with fine detail did not govern the storage needed everywhere along the curve. It would be a nice (although challenging) addition to incorporate visual smoothing as described in [5]. This problematic under the current system, however, because a visual change in the multiresolution B-spline may result in a permanent change in the other representations.

## **8.2 Other Mathematical Representations**

Although the current system uses only two-dimensional curves, the ideas extend into higher dimensional curves and surfaces. Efficiently synchronizing multiple mathematical representations becomes especially important with 3-dimensional patches because translating entire patch representations is both difficult mathematically and time consuming. I would also like to extend this system to incorporate curves usable by current commercial modeling systems. This would leverage off of a large body of tools and possibly allow users to merge two entire commercial systems.

# Bibliography

- [1] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics : Principles and Practice*. Addison-Wesley, (1990).
- [2] B. Barsky, *Computer Graphics and Geometric Modeling Using Beta-Splines*. Springer-Verlag, (1988).
- [3] 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).
- [4] E. Stollnitz, T. DeRose, and D. Salesin, *Wavelets for Computer Graphics: theory and applications*. Morgan Kaufmann, (1996).
- [5] A. Finkelstein and D. Salesin, “Multiresolution curves”, *Computer Graphics*, **28**(2), pp. 261–268 (1994).
- [6] B. Fowler and R. H. Bartels, “Constraint based curve manipulation”, *Siggraph course notes 25*, (1991).
- [7] G. Strang, *Linear Algebra and its Applications*. Harcourt Brace Jovanovich, (1988).
- [8] M. J. Banks and E. Cohen, “Realtime spline curves from interactively sketched data”, *Computer Graphics*, **24**(2), pp. 99–107 (1990).
- [9] A. Goldberg and D. Robson, *Smalltalk 80 The Language*. Addison-Wesley, (1989).
- [10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patters: Elements of Reusable Object-Oriented Software*. Addison-Wesley, (1995).
- [11] M. Gleicher and A. Witkin, “Snap together mathematics”, *Proceedings of the 1990 Eurographics Workshop on Object Oriented Graphics*, (1991).



- [12] T. Baudel, "A mark-based interaction paradigm for freehand drawing", *UIST '94 proceedings*, pp. 185–192 (1994).
- [13] P. H. Schneider, "An algorithm for automatically fitting digitized curves", *Graphics Gems*, (1990).