



## *freeform*: A Tool for Teaching the Mathematics of Curves and Surfaces

James R. Miller

University of Kansas, [jrmiller@ku.edu](mailto:jrmiller@ku.edu)

### ABSTRACT

We describe the structure and use of an interactive modeling and visualization tool used to help students understand various important concepts in curve and surface design. This paper describes those aspects of the tool related to NURBS curves, but Bezier and Hermite curves are also supported as are Bezier and B-Spline surfaces. The focus of this paper is almost exclusively on those features of the tool used to visualize the mathematics of NURBS curves and surfaces, especially focusing on an integrated visualization of how weights and knots affect the blending functions and, through them, the curve. It is currently packaged as a standalone C++ program using OpenGL, running on Macintosh and linux operating systems. A port to Windows is underway, but as a part of a larger effort to redesign and better package a Java-JOGL-Swing version, access to which will be placed in the public domain using Java Web Start.

**Keywords:** CAD education, visualizing B-Spline functions, curve & surface design.

**DOI:** 10.3722/cadaps.2010.257-267

### 1 INTRODUCTION

Over the past several years, we have developed an interactive tool called *freeform* that is used to demonstrate the mathematics of curve and surface design primarily for students in our graduate Geometric Modeling course. All of the students in the course have programming expertise as well as basic knowledge of the mathematics of 3D graphics (points and vectors in affine and projective spaces, affine transformations, etc.). Most are also familiar with OpenGL, and many, if not most, have used one or more open source or commercial CAD design tools supporting the creation and manipulation of freeform curves and surfaces. The intent of this course is to give them more in-depth understanding of the underlying mathematics and algorithms that they have been employing at a user level. They demonstrate their knowledge with a combination of written homework and programming projects.

The course is taught as a graduate level computer science course, but it is also actively advertised to graduate students in other departments. It frequently draws some from mechanical engineering as well as occasionally from mathematics and aeronautical engineering. The engineering students appreciate the chance to understand the underlying mathematics and algorithms of design packages they typically use as it removes the veil of “black magic” and helps them better understand what they can and cannot do. At the same time, it helps them to become better users of CAD packages [8]. The computer science students (and, to a certain extent, the mathematics students) are exposed to a realistic vehicle for exploring numerical programming issues, mathematical function spaces, and other

such issues that they all too frequently only study in the abstract. Many are simply motivated by a desire to better understand the technology. The class is usually sufficiently small so that all the students get to know each other, and they are able to share their respective expertise.

The *freeform* program supports parametric curves and surfaces defined as blended sums of control points:

$$\text{curves: } C(u) = \sum_{i=0}^n F_i(u)P_i \quad (1)$$

$$\text{tensor product surfaces: } S(u, w) = \sum_{i=0}^n \sum_{j=0}^m F_i(u)F_j(w)P_{ij} \quad (2)$$

It also supports sketching ordered collections of point-vector pairs to generate piecewise Hermite curves.

We try to emphasize the importance of understanding how blending functions determine general properties of curves and surfaces, while the control points determine the shape of a particular instance. This intuition is sometimes easier to convey with some classes of curves and surfaces (e.g., Bezier) than it is with others (e.g., NURBS). For example, the fact that design handles such as knots and weights affect the blending functions themselves can be a stumbling block. To help with the development of this understanding, *freeform* includes an integrated and interactive visualization of how adjusting knots and weights affect the shape of the blending functions as well as that of the curve. In side-by-side windows, users watch the shape of the functions and the shape of the curve change in response to interactive knot and weight adjustments. As will be illustrated below, other features of *freeform* present visualizations of the relationship between spans and control points as determined by the knot vector.

As the scope and features of this tool have evolved, we have been using it in a more fundamental way in the course. While doing so, we have observed much greater engagement on the part of students, better performance on homework and exams, much stronger ability to handle more complex and detailed questions, and, perhaps most interestingly, much better questions raised in class.

## 2 OTHER VISUALIZATION-BASED SPLINE EDUCATIONAL TOOLS

While we know of no educational tool with the breadth of coverage of ours, others do exist which present visualizations of various sorts specific to B-Splines. Ones with which we are familiar are described below.

Lerios describes a program called *scurvey* that was written to illustrate the dependence of B-Spline curves on control point positions and knot vector sequences [5]. No attempt to visualize the basis functions is mentioned, nor does *scurvey* as described support rational B-Splines. There are also some visualization aids described that allow the user to see what portion of a B-Spline curve will be affected if a particular control point is moved.

Chang developed a tool using Mathematica that allows a single non-rational B-Spline curve to be manipulated [1]. A button is used to toggle between showing the blending functions and showing the curve. Sliders can be used to adjust the knot values.

Demidov offers web pages with Java applets that demonstrate qualitatively how knot placement affects B-Spline basis function shapes [2]. No quantitative knot values are shown, and it is difficult to experiment with knots of multiplicity greater than 1. It is also difficult to relate blending functions to control points or positions on curves to points on the blending functions or even spans in parameter space.

Unlike the previous two examples, Fisher describes a tool that focuses on control point weights in NURBS curves [4]. They present weighted control points as points in projective space and illustrate how the curve is generated by projecting the projective space curve points back to the affine plane. There is little or no mention of knots, visualization of basis functions, or ability to alter knot spacing for their curves.

By and large, all these tools allow only a single planar curve and permit visualizations of limited aspects of the curves and/or basis functions. Our goal was to develop a comprehensive set of

visualizations that allowed the user to understand deeply the relationship between knots, basis functions, control points, weights, and curve shape. Moreover, we also wanted to be able to illustrate common algorithms in a context sufficiently rich so that not only their operation, but also their application could be appreciated. This requires the ability to store and manipulate several 3D curve and surface instances at once.

### 3 THE DESIGN OF *FREEFORM*

The *freeform* program was designed first and foremost as a teaching tool, not as a production curve and surface design tool. Most notably, this means certain features are very limited or missing altogether. For example, there are relatively few *ab initio* curve and surface creation mechanisms. There is no support for matching curvatures at specific positions or other similar precision design tools. There are only limited sketching input facilities.

Supported curve creation operations include:

- Interpolation:
  - Create a degree  $n$  Bezier curve exactly interpolating an ordered collection of  $n+1$  points. (A few options are provided for selecting how parameter values are associated with the points.)
  - Create a piecewise cubic Hermite curve interpolating an ordered set of  $n+1$  point-vector pairs.
- Least squares approximation: Create a degree  $m$  Bezier curve which is the least squares approximation to a collection of  $n$  points, where  $m < n$ . (A few options are provided for selecting how parameter values are associated with the points.)
- Shape approximation: Use an ordered set of  $n+1$  points as the control polygon for a degree  $n$  Bezier or Rational Bezier curve; alternatively, use the ordered set as the control points for a degree  $p$  B-Spline, or rational B-Spline (NURBS) curve,  $p \leq n$ .

The points and/or point-vector pairs can be digitized or read from a file.

Supported surface creation operations include:

- Generate a regular  $n+1 \times m+1$  array of points on a plane to be used as control points for a Bezier or NURBS surface.
- Create a NURBS surface as, for example, a ruled surface, sum surface, or revolution surface.

There is a simple text file format that can be read by *freeform* to import geometry. Geometry can be saved by *freeform* in this format for later re-loading. Its format is also sufficiently simple so as to be readily generated by other applications to allow geometry from those sources to be imported into *freeform*.

Several features are included in *freeform* that would likely not be visible at the user interface in production design systems, or at least not as directly as they are here. Some of these features are listed below. We will discuss these in detail in the next section.

- spinner controls for setting the knot values of the current B-Spline, including allowing knots to assume any multiplicity;
- spinner controls for adjusting weights of the current B-Spline;
- separate side-by-side displays of normalized and unnormalized B-Spline basis functions that, along with the separate display of the corresponding curve, are dynamically updated as weights and/or knots are adjusted;
- several passive highlighting techniques discussed below that are designed to help the student visualize several important concepts including the spans affected by a given control point as well as the set of control points affecting a given span;

The rest of the paper focuses almost exclusively on NURBS curves and associated visualizations from the perspective of a student trying to master the relationships among the blending functions, control points, weights, and knot vectors in order to better understand the underlying NURBS representations and algorithms.

## 4 USING *FREEFORM* TO VISUALIZE NURBS CURVES

Our general approach when teaching this material is to cover the detailed mathematics and proofs in parallel with the more intuitive and interactive visualizations presented in this section. The goal is to interpret the visual properties we discover as geometric manifestations of corresponding mathematical properties of the blending functions and curves that we derive. Eventually we try to build the intuition that allows students to look at a visualization of some aspect of a curve's geometry and be able to describe it in terms of what the knots and weights (and hence blending functions) must be; and vice versa: be able to see some mathematical result or expression and be able to predict what the visual manifestation of it will be.

### 4.1 Preliminaries: Creating a Curve; Basic Control Point - Knot - Blending Function Associations

A top-level dialog window allowing the student to create instances of curves and specify how certain manipulations are to be interpreted is shown in Fig. 1.

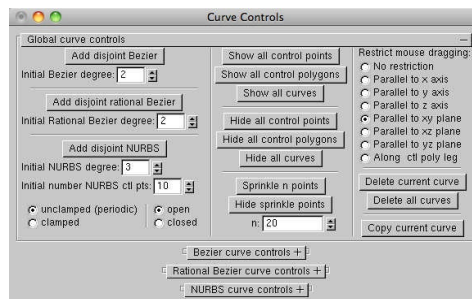


Fig. 1: Basic curve control window.

When a student clicks the “Add disjoint NURBS” button, a NURBS curve with the indicated properties is created and displayed. Using the properties settings as displayed in the Curve Controls window of Fig. 1., an open, periodic, cubic B-Spline with ten control points will be created, all of whose weights are 1. As illustrated in Fig. 2(a)-(c)., side-by-side windows are then presented so the student can see the curve, the knot vector and weights, and the basis functions.

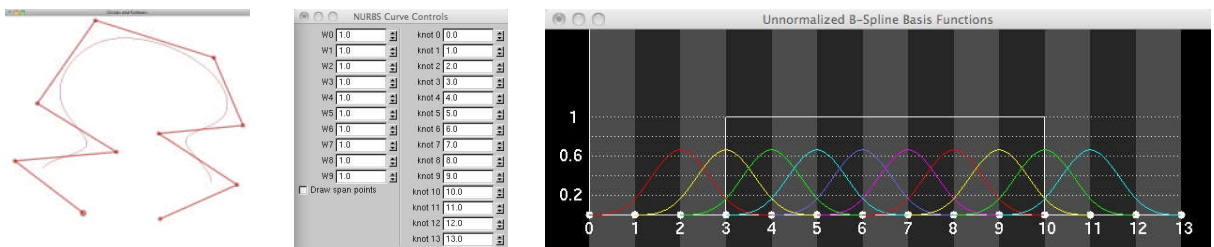


Fig. 2: After creation of a cubic NURBS curve, side-by-side windows present (a) the curve, (b) the weight and knot controls, and (c) the Basis Functions.

We shall use the following notation and conventions when discussing NURBS curves and the various visualizations supported by *freeform*: the curve is of degree  $p$ , order  $k=p+1$ ; it employs  $n+1$  control points and utilizes a knot vector,  $\mathbf{U} = (u_0, \dots, u_{n+k})$ .

In the basis function window, a solid white rectangle outlines the parametric domain. The alternating light and dark gray vertical bands emphasize the parametric spans. In Fig. 2(c) this is  $u_p = u_3 = 3 \leq u \leq u_{n+1} = u_{10} = 10$ . This parametric domain generally makes sense to the student at this point because we have already derived the proof that the blending functions  $N_{j,k}$ ,  $i - k + 1 \leq j \leq i$  are

the only non-zero order  $k$  blending functions defined over the parametric interval  $u_i \leq u < u_{i+1}$ , and that these  $k$  functions sum to 1. Between the proof and this visualization, students come to understand the idea that every span of an order  $k$  B-Spline curve has exactly  $k$  non-zero blending functions active, and hence the shape of each span is determined solely by the corresponding set of  $k$  control points. Moreover, it is then clear to the student that the parametric domain must start at knot  $u_p$  because that is when we first have  $k$  active blending functions. It ends at knot  $u_{n+1}$  because we do not create an  $(n+1)$ -th blending function (there is no  $(n+1)$ -th control point for it to blend), hence we no longer have  $k$  non-zero blending functions for  $u > u_{n+1}$ .

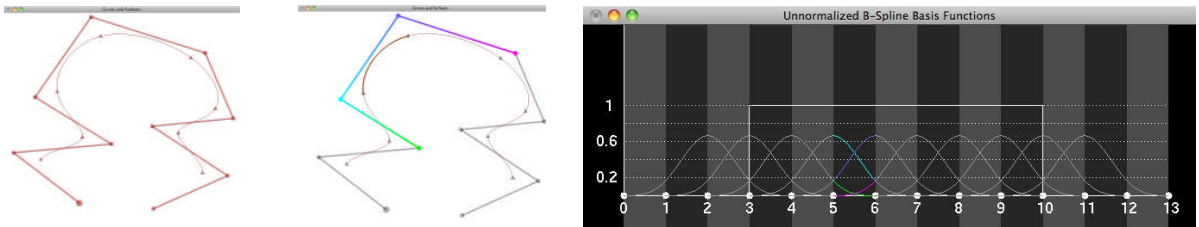


Fig. 3: Some passive highlighting in the side-by-side windows: (a) the curve with its break points, (b) a segment highlighted in brown with control points colored to match their blending functions, and (c) the active blending functions colored to match the control point colors.

To make this connection more concrete as well as to better illustrate exactly what control points affect a given span, we employ a passive highlighting scheme operating in unison across the curve geometry and blending function windows. As a prelude, we request that span break points be displayed. (See the check box in Fig. 2(b)) The student then sees the image of Fig. 3(a). As the student passively moves the cursor over a span, that span is highlighted in brown, and the portion of the control polygon defined by the  $k$  control points corresponding to the  $k$  non-zero blending functions is highlighted using a color scheme tied to that of the blending functions (Fig. 3(b)). We can also see in Fig. 3(b) that all other control polygon legs in the curve geometry window have been grayed out; similarly, all portions of all blending functions in the blending function window outside of the corresponding span have been grayed (Fig. 3(c)) - all to emphasize the connection between the parameter and coordinate spaces.

One use of the display in the curve geometry window is to make clear to the student that any change to the highlighted control points will change the shape of the highlighted span; conversely, in order to adjust the shape of that highlighted section, it is necessary to adjust the position of one or more of the highlighted control points.

The closely related concept is that all control points (other than the first  $p$  and the last  $p$ ) have influence over  $k$  consecutive spans. Additional passive highlighting schemes are used to convey this idea as well as to visualize how much of the curve is affected by the first  $p$  and last  $p$  control points.

In Fig. 4(a), the student has positioned the cursor over control point  $P_5$ . All elements of the display have been grayed out except the indicated control point and the  $k$  spans it affects. Meanwhile, the blending functions in the adjacent blending function window have all been grayed out except for the corresponding blending function. The function and the point have been assigned the same color to make the connection obvious. Fig. 4(c)-(d) illustrates the same idea for an initial control point whose influence extends over only 2 spans since its blending function starts two spans before the start of the active parametric domain.

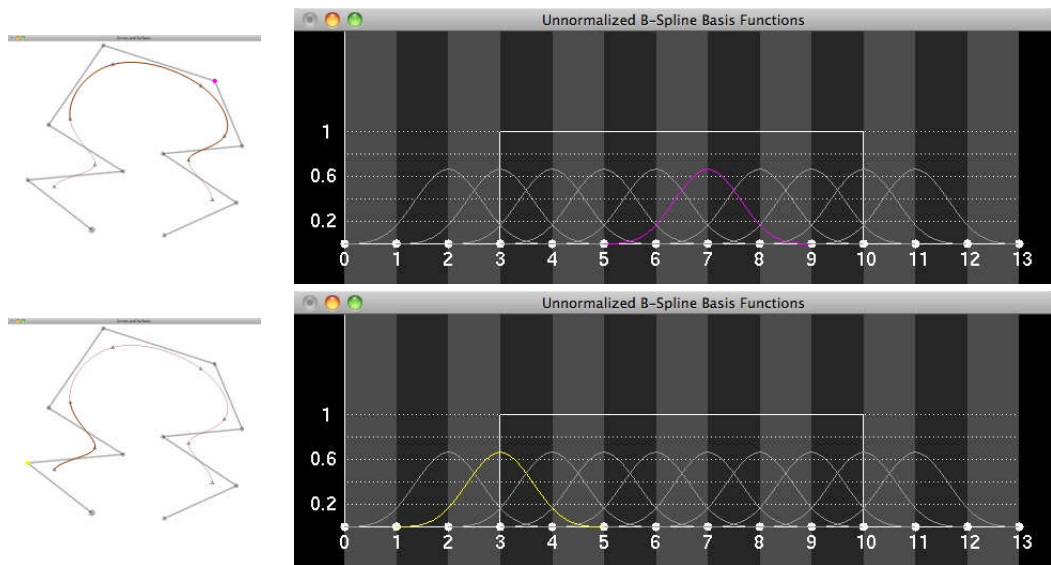


Fig. 4: Passive highlighting in the side-by-side windows to illustrate the connection between control points and affected spans: (a) an interior control point affecting all  $k$  spans, (b) its blending function whose domain is entirely within the curve parametric domain, (c) one of the first  $p$  control points with the two spans it influences highlighted, (d) its blending function whose domain includes two spans outside the parametric domain of the curve.

## 4.2 Closed Curves

To this point, students have seen only uniform knot spacing and the periodic blending functions they generate. It generally only takes a little explanation and a simple demonstration of moving the positions of the final  $p$  control points on top of the positions of the first  $p$  to show them how a closed  $C^{p-1}$  curve can be created. Selecting the “closed” radio button shown in Fig. 1 and creating a new curve then illustrates how this can be done automatically.

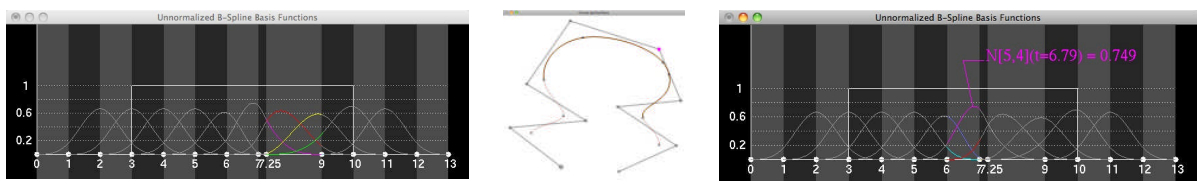


Fig. 5: Illustrating the effect of knot adjustment: (a) knot  $u_8$  is decreased in magnitude, increasing the maximum value of  $N_5$  while decreasing the maximum value of nearby blending functions, (b) the curve draws closer to control point  $P_5$  as a result, (c) using the blending function probe to examine the new maximum value of blending function  $N_5$ .

## 4.3 Nonuniform Knot Spacing and Clamped Curves

As a first step towards exploring nonuniform spacing, the slider of an interior knot,  $u_8$ , is adjusted. The students watch as various curve spans grow and shrink in arc length; simultaneously they can see

how the spans in the blending function window grow and shrink (Fig. 5(a)-(b)). More significantly, the relative influence of various control points in the adjacent spans of the curve rise and fall at the same time. Fig. 5(c), for example, shows how control point  $P_5$  has more influence in those spans after the knot adjustment since its blending function ( $N_{5,k}$ ) grows while others (e.g.,  $N_{4,k}$ ) diminish. Students observe the curve drawing closer to  $P_5$  as a result.

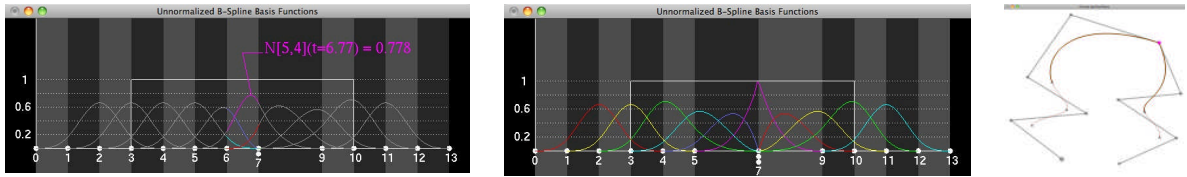


Fig. 6: Illustrating the effect of knot multiplicity: (a)  $u_7 = u_8$ , (b)  $u_6 = u_7 = u_8$ , (c) control point  $P_5$  is interpolated as a result.

We then drop  $u_8$  even more until it equals the value of  $u_7$  (Fig. 6(a)). A zero-length span has been introduced, and even though the passive control point span highlighting (e.g., Fig. 4(a)) appears to indicate that control point  $P_4$  now only influences three spans, it is obvious to the students that it still affects four because they have watched one of the spans gradually shrink until its length is zero. We also point out to the students that this knot vector configuration has caused the curve to become tangent to the  $P_5 - P_6$  control polygon leg - this result being the necessary geometric consequence of the fact that only  $N_5$  and  $N_6$  are non-zero at the duplicated knot.

We then demonstrate increasing the multiplicity to  $p = k - 1 = 3$ , this time interactively increasing  $u_6$  until  $u_6 = u_7 = u_8$ . The students watch as blending function  $N_5$  spikes to a value of 1 at the repeated knot while the curve pulls into and interpolates  $P_5$ . (See Fig. 6(b)-(c).)

It is natural to ask - and students typically do at this point - what happens if the knot is repeated again (and again and again...). After trying to extend their understanding to guess the answer, we demonstrate the effect by setting  $u_5$  equal to the knot whose multiplicity was previously three to yield the displays of Fig. 7. Students see blending function  $N_4$  spike to 1 at  $u = u_5$ , then immediately drop to 0; meanwhile,  $N_5$  starts at 1 at  $u = u_5$  and then gradually tapers down to 0. The resulting broken curve is then intuitive.

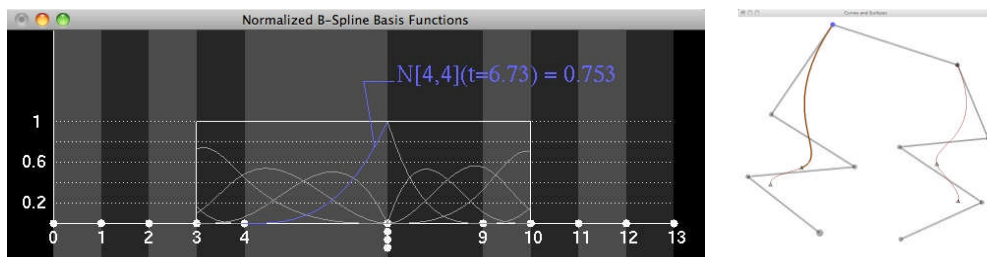


Fig. 7: Creating an interior knot with multiplicity  $k$ , breaking the curve: (a) two blending functions spike to 1 at the repeated knot, (b) the curve is broken into two pieces.

Describing how to generate B-Splines that interpolate their initial and final control points is fairly straightforward at this point. Combining an understanding of how duplicated knots allow interpolation of arbitrary control points with the fact that the parameter range of the curve starts at the  $k$ -th knot and ends at the  $(n+1)$ -th knot, it is not surprising that  $k$  initial and  $k$  final duplicate knots lead to endpoint interpolation, and we introduce the common term “clamped B-Spline” to describe B-Splines with this property. We generally start with an unclamped B-Spline and interactively adjust the initial and final knots to have multiplicity  $k$  so that students can watch the curve gradually draw near the first and last control points, ultimately interpolating them.

One slightly confusing issue that arises at this point relates to the specific control point that gets interpolated when knots are repeated the appropriate number of times. When first introducing the idea in the context of repeated interior knots, we discover the rule that says: if  $u_i$  is the first of  $p = k - 1$  repeated knots, then control point  $P_{i-1}$  is interpolated at parameter value  $u = u_i$ . But things are a little different at the start since we have  $u_0 = L = u_{k-1}$ , but control point  $P_0$  is interpolated. There are various ways to explain this, but we have found it convenient to explain it in the context of another mystery: exactly how many knots are really needed to describe an order  $k$  B-Spline with  $n+1$  control points?

Students are frequently assigned readings from different sources, and the “how many knots are required?” question often arises in that context. For example, a student will read in Farin’s book [3] that an order  $k$  B-Spline with  $n+1$  control points requires  $n+k-1$  knots. They then read in Piegl & Tiller [6] that there are  $n+k+1$  knots! Is there a typographical error? Is one of them wrong? We demonstrate that neither is really wrong by showing that the actual value of the first and last knot are irrelevant to the shape of the curve. We demonstrate this fact in *freeform* by observing that, as we drag the first and last knot around, no part of the blending functions inside the parametric domain change; similarly, the curve shape is completely unaffected as we modify those two knot values. The geometry of Fig. 8. illustrates a clamped order 5 B-Spline along with its blending functions. Fig. 8(a). shows the curve, and Fig. 8(b). shows the corresponding blending functions as determined by the knot vector whose first and last knots have multiplicity  $k=5$ . As we interactively decrease the first knot and/or increase the last knot, students see that we modify the initial (final) span of the corresponding blending functions, but we do not modify any blending function in any way within the defined parametric domain of the curve. It is then no surprise when they also observe that the shape of the curve remains unaltered as those two knots are modified.



Fig. 8: The actual value of the first and last of the  $n+k+1$  knots of a B-Spline do not affect the shape of the B-Spline curve: (a) blending functions for an order 5 clamped B-Spline, (b) the corresponding curve, (c) the blending functions after decreasing the first and increasing the last knot.

In summary, strict interpretation of the recursive definition of the basis functions most naturally leads to the use of  $n+k+1$  knots. On the other hand, typical implementations of evaluation algorithms that start by locating the only non-zero order  $l$  basis function at a given parameter value and then work up to the appropriate order  $k$  function will never actually use knot 0 or knot  $n+k$ . This observation, coupled with the fact that the curve shape is unaffected by the first and last knot provide the justification for saying only  $n+k-1$  knots are required. Of course care must be taken when interfacing with programming APIs like OpenGL or geometric data base representations (IGES, STEP; commercial ones like ACIS; etc.) to make sure that relevant conventions are observed. Some of these expect only  $n+k-1$  knots; others require  $n+k+1$ .



### 4.4 Weights and Rational B-Splines

The B-Spline controls shown in Fig. 2(b). show spinners allowing weights of the control points to be modified. This intuition is easy to convey: increasing weights draws the curve projectively towards the given control point. We introduce the standard curve representation:

$$C(u) = \frac{\sum_{i=0}^n N_{i,k}(u) w_i P_i}{\sum_{i=0}^n N_{i,k}(u) w_i} \tag{3}$$

Particular values of the weights in conjunction with specific knot vector and control point placement allow the conic curves to be exactly represented. Fig. 9 illustrates matching ninety degree conic section arcs with the indicated weight assignments. (These are actually rational Bezier curves, but students understand by this time how to generate a rational B-Spline representation of a rational Bezier curve.)

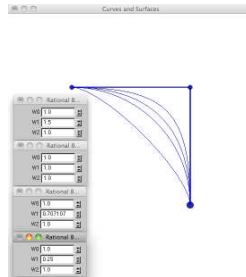


Fig. 9: Matching a hyperbola, parabola, circle, and ellipse with rational curves.

The main additional intuition at this point is to understand the basis functions. We introduce two standard equivalent mathematical representations for the NURBS curve. The first employs what we call “unnormalized blending functions” (the conventional blending functions multiplied by the weights):

$$N_{i,k}^U(u) = w_i N_{i,k}(u) \tag{4}$$

The curve representation remains defined in affine space and becomes:

$$C(u) = \frac{\sum_{i=0}^n N_{i,k}^U(u) P_i}{\sum_{i=0}^n N_{i,k}^U(u)} \tag{5}$$

The second representation employs what we call “normalized blending functions” and are only applied to control points embedded in projective space using their weights:

$$P_i^P = (w_i x_i, w_i y_i, w_i z_i, w_i) \tag{6}$$

$$N_{i,k}^N(u) = \frac{w_i N_{i,k}(u)}{\sum_{j=0}^n w_j N_{j,k}(u)} \tag{7}$$

Using Eqn. (6) and Eqn. (7), the NURBS curve is defined in projective space as:

$$C^P(u) = \sum_{i=0}^n N_{i,k}^N(u) P_i^P \tag{8}$$

In *freeform*, we allocate one display window for the collection of  $N_{i,k}^U$  functions, and one for the set of  $N_{i,k}^N$ . These windows, displayed in Fig. 10, are titled “Unnormalized B-Spline Basis Functions” and “Normalized B-Spline Basis Functions”, respectively.

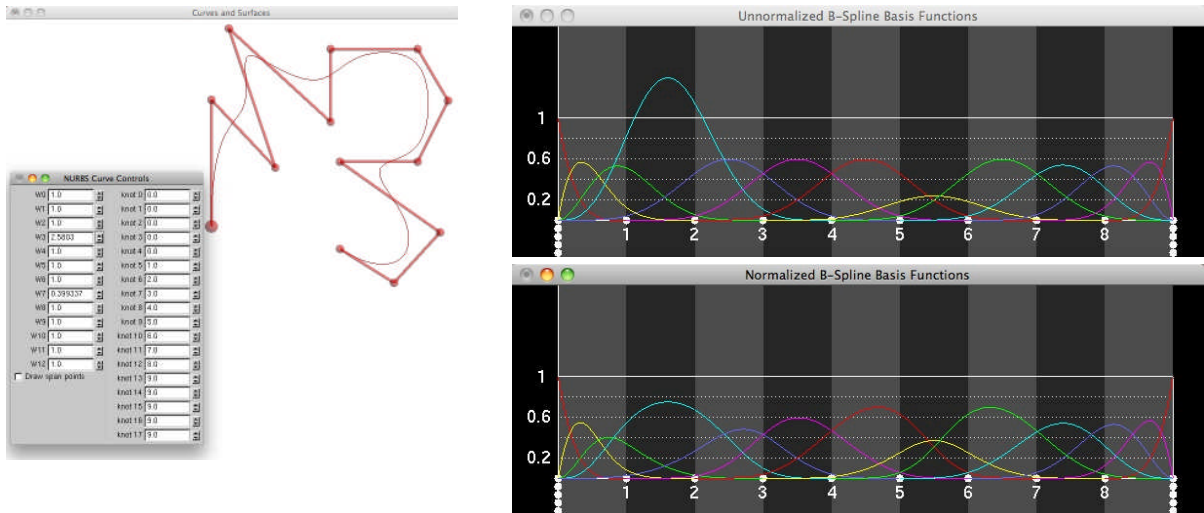


Fig. 10: The geometry of Fig. 8 after increasing the weight of  $P_3$  and decreasing the weight of  $P_7$ .

From Eqn. (4) it is clear that the  $N_{i,k}^U$  functions will not in general sum to 1. In fact, individual functions may exceed 1 as can be seen in Fig. 10. Modifying weight  $w_i$  affects only  $N_{i,k}^U$ ; all other functions remain unchanged. By contrast, the normalized  $N_{i,k}^N$  functions allow students to see exactly how the influence of other points decrease (increase) as  $w_i$  increases (decreases). In the example of Fig. 10., students can observe how continuous increases to  $w_3$  cause corresponding continuous decreases in all blending functions that are non-zero over a parametric interval which overlaps that of  $N_{3,k}^N$ . Therefore, the influence of the corresponding control points on the curve decreases, something that is readily observed by watching the curve as the weight  $w_3$  increases.

#### 4.5 Other Features

Several other features of *freeform* permit students to see the operation of important algorithms like knot insertion, conversion of a B-Spline to an equivalent piecewise Bezier, and others. There are also various interactive tools for restricting mouse-based dragging of points to given axes, coordinate planes, or control polygon legs. A marker can be moved through the Basis function window while the corresponding B-Spline point is traced along the curve. While doing so, successive spans along with the control points determining their shape are highlighted as in earlier examples.

### 5 CURRENT STATUS AND PLANS

The *freeform* program is used to a limited extent late in our undergraduate senior-level graphics class in which the focus is less on the mathematics and more instead on intuition. The goal is to teach the basics of curve and surface modeling at a user level and possibly interest some students in taking the graduate Geometric Modeling class in which the program is used extensively. In that course, the goal is to understand the mathematics and algorithms much more deeply.

The current implementation has evolved over several years and is written in C++ using OpenGL and the GLUT for basic display, and the `glui` [7] for most user interface widgets. An effort is currently underway to clean up the user interface, make some things more consistent and move the higher level code to Java, Java OpenGL (JOGL), and Swing. The goal is to then deliver the application over the internet via Java Web Start.

## 6 SUMMARY

We have described the use of a portion of *freeform*, a curve and surface modeling tool developed first and foremost as an educational tool to help students understand the mathematics of curves and surfaces, especially B-Splines and rational B-Splines. Reaction from students in various classes has been positive, and performance on homeworks, projects, and exams has exhibited significant improvement as the tool has evolved and been used more extensively.

## REFERENCES

- [1] Chang, Y.-S.: B-Spline Curve With Knots, <http://demonstrations.wolfram.com/BSplineCurveWithKnots/>.
- [2] Demidov, E.: An Interactive Introduction to Splines, <http://www.ibiblio.org/e-notes/Splines/Intro.htm>.
- [3] Farin, G.: Curves and Surfaces for CAGD: A Practical Guide, fifth edition, Morgan Kaufmann Publishers, San Diego, CA, 2002.
- [4] Fisher, J.; Lowther, J.; Shene, C.-K.: If You Know B-Splines Well, You Also Know NURBS!, Proceedings SIGCSE 2004, March 3-7, 2004, Norfolk, VA, pp. 343-347.
- [5] Leros, A.: B-Spline Curve Visualization, <http://graphics.stanford.edu/courses/cs348c-95-fall/software/scurvy/>.
- [6] Piegl, L.; Tiller, W.: The NURBS Book, second edition, Springer, New York, 1997.
- [7] Rademacher, P., GLUI User Interface Library, <http://www.cs.unc.edu/~rademach/glui/>.
- [8] Saakes, D.: Hit and Render: Teaching CAD Visualization to Product Designers, Computer-Aided Design & Applications, Vol. 3, Nos. 1-4, 2006, pp. 315-322.