# ASTERISK*: AN EXTENSIBLE TESTBED FOR SPLINE DEVELOPMENT

*Jonathan R. Gross* †
*Tony D. DeRose* ‡
*Brian A. Barsky*
Berkeley Computer Graphics Laboratory
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA. 94720
U.S.A.

*Abstract*

*Asterisk** is a testbed system designed to support the development of new kinds of splines. The key concept is the integration of *symbolic computation facilities* with tools for *interactively modifying and comparing* different splines. By modeling a spline as a list of attributes, *Asterisk** can be used to create and manipulate almost any spline, without making assumptions about the future directions of spline research.

*Résumé*

*Asterisk** est un système d'étude créé pour appuyer le développement de nouvelles sortes de courbes à base de splines. Le concept clef est l'intégration d'*outils interactifs pour modifier et comparer* différents courbes, avec un *système de calcul symbolique*. En décrivant une courbe par une liste d'attributs, le système *Asterisk** peut être utilisé pour créer et manipuler presque n'importe courbe à base de splines sans présumer les directions de cette recherche à l'avenir.

*KEYWORDS:* geometric modeling, software, symbolic computation, parametric curves.

## 1. Introduction and Motivation

A *spline* is a mathematical formulation of a curve used for a wide variety of modeling applications. Unlike polygonal representations, splines provide compact and resolution-independent descriptions of complex objects. Many splines combine a set of *control vertices* with a set of *blending functions* to determine the path of a curve through space. Different splines have different properties, and the choice of which spline to use depends on the problem at hand. For example, some splines *interpolate* (pass through) the control vertices, while others *approximate* (pass near) them.

There is no single type of spline that is ideal for all applications; each spline is a tool best suited to some particular set of tasks. Current spline research involves developing new splines with specific properties as solutions to different problems. To create a spline representation with a desired set of properties, the blending functions must satisfy a set of *constraints*. Sometimes this involves combining constraints from existing splines to arrive at a new formulation [4]; in other cases it requires substantial trial and error to find a suitable set of constraints that produces the desired properties.

Most splines can be described mathematically in simple terms. In spite of this, establishing the constraints that correspond to desired properties, and then solving for a set of blending functions that satisfy those constraints is a task that can easily become overwhelming. Computer algebra systems such as *Vaxima* [5] can perform error-

* A Simple Testbed for the Evaluation and Rendering (Interactively) of Splines of many Kinds.

† Author's current address: Vertigo Systems International, Suite 221-119 West Pender Street, Vancouver, B.C. Canada V6B 1S5.

‡ Author's current address: Department of Computer Science, FR-35, University of Washington, Seattle, WA. 98195, U.S.A.

This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00039-82-C-0235, the National Science Foundation under grant number ECS-8204381, the Natural Sciences and Engineering Research Council of Canada, the State of California under a Microelectronics Innovation and Computer Research Opportunities grant, and a Shell Doctoral Fellowship.

free symbolic (as opposed to numeric) computations of complexity too great for humans to handle. Vaxima takes a description of the constraints and produces a *symbolic representation* of the blending functions. Recent efforts in spline development have taken advantage of such capabilities [1,4].

To understand the behaviour of new splines, it is useful to draw them, preferably in an interactive setting. This generally requires an *evaluation routine* to compute points on the curve, and a collection of routines to graphically display and modify the curve. The coding of the evaluation routine requires tedious translation from the symbolic representation into the implementation language.

Intuition gained from visual feedback, as well as comparison with other spline representations, often leads to modification of the constraints that define the spline. This in turn changes the symbolic representation, necessitating a recoding of the evaluation routine.

We would like to tighten this specification/visualization loop and automate the translation step. Feng & Riesenfeld [6] describe just such a system for the development of *Boolean sum surfaces*. Although Feng & Riesenfeld had access to the symbolic algebra system *REDUCE* [7], they chose not to use it for two reasons: they were interested primarily in simple operations on rational bivariate polynomials, and they felt that they could not provide a sufficiently high level of user interaction by running a large system like REDUCE. Instead, Feng & Riesenfeld implemented a small, specialized algebra system capable of performing operations such as operator composition, evaluation, and symbolic differentiation of bivariate polynomials.

This paper describes a testbed system called *Asterisk** designed to integrate the power of symbolic computation with convenient, extensible, interactive tools for modifying and comparing different splines [8]. *Asterisk** is intended for use by those doing research in spline techniques rather than those wishing to design specific objects using splines. Our system is similar to that of Feng & Riesenfeld, but differs in that the emphasis is on *generality*. Since future spline representations may not conform to all the paradigms of current spline techniques, one of our primary goals was to avoid making too many assumptions about the future directions of spline research.

*Asterisk** supports generality in three ways: the use of a complete symbolic algebra system (Vaxima), an extensible model of a spline (a list of attributes), and an extensible set of interactive spline management and graphics routines (written in Lisp). The blending functions may be developed and verified within Vaxima, and then used directly to draw or plot the resulting curves. As new algorithms are developed to manipulate the spline representations, routines to implement these algorithms can be written quickly, thus extending the basic building blocks that *Asterisk** provides. In this way, spline researchers can adapt the interactive testbed to their particular requirements and preferences.

## 2. The Asterisk* Solution

### 2.1. The Asterisk* Model of a Spline

Thoughout this paper we use the term spline to mean a piecewise parametric function of the form

$$\mathbf{Q}(u) = \sum_{i=0}^{m} \mathbf{V}_i \, B_i(u) \qquad (2.1)$$

where the $\mathbf{V}_i$ represent a set of $m+1$ *control vertices* and the $B_i(u)$ represent a set of *blending functions*.

*Asterisk** provides a uniform definition protocol for spline specification by modeling a spline as a list of *attributes*. Each attribute is a <name,value> pair. Table 1 summarizes some common attributes. In Lisp, these are stored as *property lists* of the spline's name. For instance, the attribute list for a spline named "ucb_spline" is stored on the property list for the symbol *ucb_spline*. The attribute list in Figure 1 was used to define the spline illustrated in Colour Plate 1.

```
(splinename ucb_spline
 polycolor GREEN
 polystyle DASHED
 polydisplay ON
 curvecolor WHITE
 curvestyle SOLID
 curvedisplay ON
 controlpolygon ((.42 .41) (.41 .73) (.55 .86)
                 (.86 .76) (.86 .59) (.61 .58)
                 (.61 .38) (.83 .35))
 evalroutine ucb_eval
 parameterstep 0.0625)
```

**Figure 1:** *The attribute list for ucb_spline.*

### 2.2. System Architecture

Conceptually, the *Asterisk** testbed consists of three logical parts:

- A *Vaxima to Lisp translator* for automatically generating evaluation routines from Vaxima descriptions. This is the transition from the analytic expressions for the blending functions (the symbolic representation), to the procedures that evaluate the blending functions (the numeric representation required for graphical display).
- *Interface routines* for support of graphical interactions such as geometric input or display of a curve. This module provides a device independent graphical interface to the rest of the system.

| Attribute Name | Value Type | Purpose |
|---|---|---|
| controlpolygon | List of vertices $\mathbf{V}_i$ | Rough specification of curve path. |
| evalroutine | Name of a function | Names the function which evaluates points on the curve. |
| parameterstep | Real number | The amount the domain parameter $u$ is varied between consecutive calls to the evaluation routine. |
| polystyle (curvestyle) | DOTTED DASHED SOLID | Style used to draw control polygon (curve) |
| polycolor (curvecolor) | Colour name | Colour used to draw control polygon (curve) |
| polydisplay (curvedisplay) | ON OFF | Should polygon (curve) be displayed? |
| degree | Integer | The polynomial degree of the curve. |
| knotvector | List of real numbers $u_i$ | Specifies a set of parameter values associated with the break-points between curve segments. This attribute is useful for B-spline curves and cubic interpolatory splines (cf. Bartels *et al*[3]). |
| beta1 (beta2) | Real number | Shape parameters for geometrically continuous techniques such as Beta-splines [1,2]. |

**Table 1:** *Common Attributes*

- *Spline management routines* for manipulating attribute lists. The spline management routines facilitate convenient modification of the spline attributes. These routines invoke the interface routines to interact with the user and invoke the evaluation routines to generate curve segments for each spline.

When a *draw curve* operation is initiated, the spline management routines repeatedly invoke the evaluation routine for the spline, passing in a value of the parameter. The parameter step attribute determines which, and how many, parameter values are passed in. It is the responsibility of the evaluation routine to return the point on the curve corresponding to each given parameter value. This is done by referring to the specific attributes that are needed — attributes which do not affect the computation are ignored. The spline management routines invoke the interface routines to connect the points on the curve into a piecewise linear approximation of the curve.

Ideally, we would like the spline management and interface routines running within the Vaxima environment. This approach was taken in an early implementation; unfortunately, the code space requirements of Vaxima caused excessive page faulting, resulting in poor performance. This problem can be avoided by executing Vaxima and the Vaxima to Lisp translator in one process, and the spline management and interface routines in another. In the current implementation these processes run on separate machines and communicate via disk files transmitted over a local area network. An advantage of the two-process structure is the ability to substitute an alternate symbolic computation system, such as REDUCE, by supplying an appropriate translation routine.

### 2.3. Extension Features

*Asterisk** supports extensibility in two ways. Both the set of functions that manipulate splines and the data structures used to represent splines can be extended as needed.

Functional extension is a byproduct of the Lisp environment. Extension of spline representations follows from the fact that each evaluation routine ignores attributes which do not affect its computation. Thus, not all splines require all attributes, and new attributes can easily be added for evaluation routines which require them. More importantly, these additions are completely transparent to other evaluation routines, meaning that existing code need not be modified.

## 3. Applications

### 3.1. Comparisons of Spline Curves

It is often desirable to compare and contrast the behaviour of various curves. For instance, one might want to determine the effect of changing the control polygon, the shape parameters, or the polynomial degree of a curve. One may also wish to compare the shapes of two curves of different types defined by the same control polygon. In *Asterisk**, all of these comparisons can be accomplished using the following three steps:

1. Define the curve that is to be the basis of the comparison.
2. Make a copy of the curve by copying its attribute list.

3. Change one (or more) of the new curve's attributes.

To visually distinguish between the two curves it is often useful to change display attributes such as the line style or colour of the control polygon or curve. Colour Plates 1 through 4 illustrate such comparisons.

### 3.2. The Development of a New Curve Type

As mentioned in Section 1, one of the main motivations for *Asterisk** was the ability to easily define and assess new curve types in an interactive setting. The process of designing new curve types typically involves repeated iteration through the following steps:

1. Determine the desired properties for the curve.
2. State these properties as mathematical constraints.
3. Use vaxima to solve the constraints.
4. Interactively assess and experiment with the curve.

As an example, Table 2 describes a new curve that was constructed and tested using *Asterisk**. The three columns of the table contain the desired properties for the curve (step 1), the corresponding mathematical constraints (step 2), and the associated Vaxima expressions (step 3). In the bottom right section of the table, the lengthy derivation of the Vaxima expressions describing the convex hull property has been omitted for the sake of brevity. Figure 2 contains the Lisp evaluation routine produced by the Vaxima to Lisp translator.

To interactively assess the new technique (step 4), we create a new spline attribute list and graphically specify a control polygon to which the evaluation routine is applied (see Colour Plate 3). The techniques of Section 3.1 can then be used to observe the behaviour of the curve under various conditions.

### 3.3. Interactive Testing of Algorithms

To this point, we have concentrated on the specification and comparison of splines themselves. As mentioned above, this involves creation and modification of spline attribute lists, using various operations which *Asterisk** provides. Another aspect of current spline research is the development of algorithms that operate on splines. To test interactively such an algorithm without the benefit of a testbed system such as *Asterisk**, one would be forced to write a specialized program to manage data structures and user-interaction, in addition to performing the required computation.

With *Asterisk**, one is able to concentrate on writing a Lisp function to implement the new algorithm alone; data structure manipulation is handled by the spline

| Property | Constraints | Vaxima Expression |
|---|---|---|
| Cubic polynomial curve defined by 4 control vertices | $\mathbf{Q}(u) = \sum_{i=0}^{3} \mathbf{V}_i B_i(u)$ where $B_i(u) = \sum_{j=0}^{3} k_{ij} u^j.$ | `poly(a,b,c,d) := a+b*u+c*u^2+d^3;`<br>`b0(u) := ''(poly(k00,k01,k02,k03));`<br>`b1(u) := ''(poly(k10,k11,k12,k13));`<br>`b2(u) := ''(poly(k20,k21,k22,k23));`<br>`b3(u) := ''(poly(k30,k31,k32,k33));` |
| *Symmetry:* reversing the control points reverses the curve. | $B_0(u) = B_3(1-u)$<br>$B_1(u) = B_2(1-u)$ | `b2(u) := ''(b1(1-u));`<br>`b3(u) := ''(b0(1-u));`<br>`unknowns : [k00,k01,k02,k03,`<br>`k10,k11,k12,k13];` |
| Interpolation of first and last control vertices and the midpoint of the center leg of the control polygon. | $\mathbf{Q}(0) = \mathbf{V}_0$<br>$\mathbf{Q}(\frac{1}{2}) = \frac{\mathbf{V}_1 + \mathbf{V}_2}{2}$<br>$\mathbf{Q}(1) = \mathbf{V}_3$ | `e0:b0(0)=1; e1:b0(1/2)=0; e2:b0(1)=0;`<br>`e3:b1(0)=0; e4:b1(1/2)=1/2; e5:b1(1)=0;` |
| *Convex hull:* the curve should lie entirely within the smallest convex region containing the control polygon. | $\sum_{i=0}^{3} B_i(u) = 1$<br>$B_i(u) \geq 0, u \in [0,1]$ | `b0d1(u) := ''(diff(b0(u),u));`<br>`b1d1(u) := ''(diff(b1(u),u));`<br>`e6:b0d1(0) = -1; e7: b1d1(0)=0;`<br>`equations : [e0,e1,e2,e3,e4,e5,e6,e7];`<br>`answer : linsolve(equations, unknowns);` |

Table 2: *Definition of a New Curve Type*

```
(def b0 (lambda (u)
     (+ 1.0 (*.  -5.0 u)
            (*.  8.0 (expt u 2.0))
            (*.  -4.0 (expt u 3.0)))))
(def b1 (lambda (u))
     (+ (*.  4.0 u)
        (*.  -8.0 (expt u 2.0))
        (*.  4.0 (expt u 3.0)))))
(def b2 (lambda (u)
     (+ (*.  4.0 (expt u 2.0))
        (*.  -4.0 (expt u 3.0)))))
(def b3 (lambda (u)
     (+ u (*.  -4.0 (expt u 2.0))
          (*.  4.0 (expt u 3.0)))))
```

**Figure 2:** *Code produced by the vaxima to lisp translator for the example of Table 2.*

management routines and user interaction is handled by the interface routines. If the operation provided by the algorithm is of lasting interest, it is a simple matter to include it on one of the *Asterisk** menus, thereby allowing the user to graphically invoke the algorithm by selecting the menu item and the spline that is to be operated on.

As a specific example, consider an algorithm to perform linear subdivision of polynomial splines. The subdivision process takes as input the control polygon describing one segment of a curve, and returns a control polygon that generates only a portion of the original curve (see Colour Plate 4). This operation was added to *Asterisk** by writing a Lisp function that requests a copy of a spline's attribute list, and replaces the *controlpolygon* attribute with a subdivision polygon. The new operation can now be applied interactively and the subdivided curves can be directly compared to the original ones.

## 4. Summary

Symbolic algebra systems have allowed increasingly complex spline formulations to be developed. However, interactive visual feedback is needed to understand the behaviour of the resulting curves. Intuition thus gained often leads to reformulation of the underlying mathematics.

*Asterisk** has been developed to tighten this design loop by automating the conversion between symbolic and graphical representations. As such, it provides a framework for the interactive development, manipulation, and comparison of new splines; it is not merely a program with a fixed set of built-in splines. *Asterisk** also provides facilities for quickly implementing and exercising new algorithms that operate on spline representations. In addition, because *Asterisk** allows one to concentrate on the mathematics and implementation of computational algorithms without worrying about user interaction and graphical display, we believe that it will prove useful as a tool for teaching basic curve and surface techniques.

The *Asterisk** model of a spline as a list of attributes does not restrict us to dealing with two-dimensional curves. Modifying the data structures and graphical display routines to handle three-dimensional curves and surfaces is an obvious extension.

## References


1. Brian A. Barsky, *The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures*, Ph.D. Thesis, University of Utah, Salt Lake City, Utah (December, 1981).

2. Brian A. Barsky, *Computer Graphics and Geometric Modelling Using Beta-splines*, Springer-Verlag, Tokyo (1986).

3. Richard H. Bartels, John C. Beatty, and Brian A. Barsky, *An Introduction to the Use of Splines in Computer Graphics*, UCB/CSD 83/136, Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, USA (August, 1983). Also Tech. Report No. CS-83-9, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

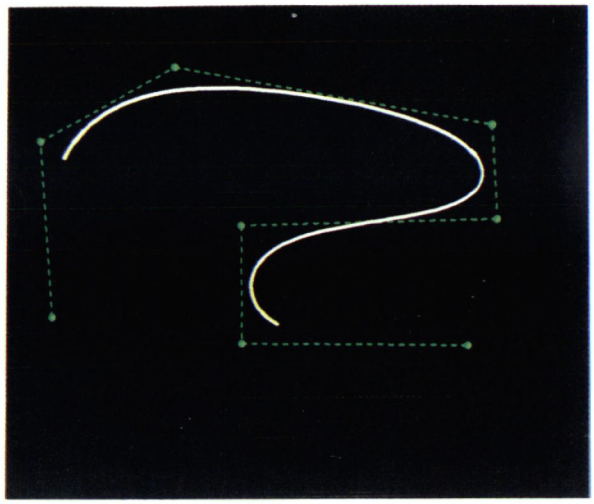
4. Tony D. DeRose and Brian A. Barsky, "Geometric Continuity and Shape Parameters for Catmull-Rom Splines (Extended Abstract)," pp. 57-62 in *Proceedings of Graphics Interface '84*, Ottawa (27 May - 1 June, 1984).

5. Richard J. Fateman, *Addendum to the MACSYMA Reference Manual for the VAX*, Computer Science Division, University of California, Berkeley (1982).

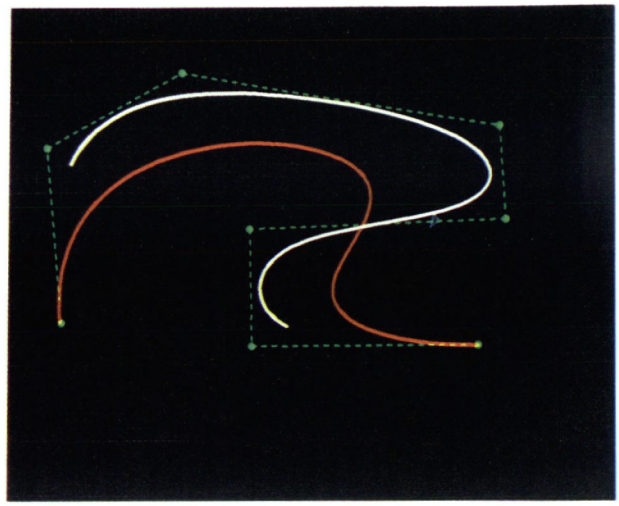
6. David Y. Feng and Richard F. Riesenfeld, "A Symbolic System for Computer-Aided Development of Surface Interpolants," *Software — Practice and Experience*, Vol. 8, No. 4, July-August, 1978, pp. 461-481.

7. Martin L. Griss, *A REDUCE Symbolic-Numeric Tutorial*, Utah Symbolic Computation Group Operating Note, Technical Report No. UCP-32, Department of Computer Science, University of Utah (October, 1977).

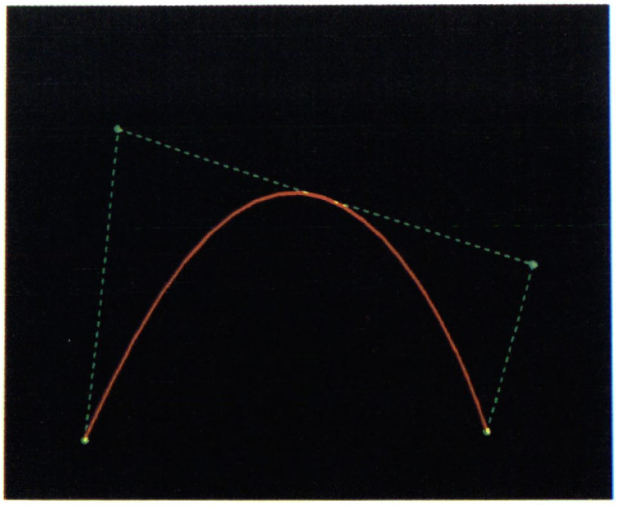
8. Jonathan R. Gross, *Software Tools for Computer Graphics Research and Development*, Masters Report, University of California, Berkeley (May, 1984).



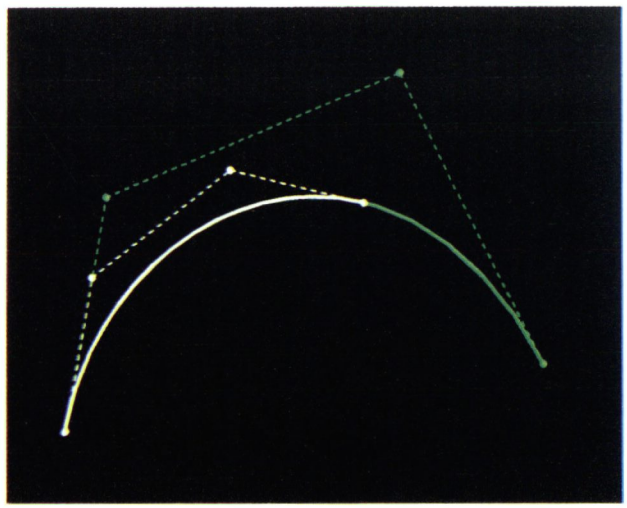
**Colour Plate 1:** *A uniform cubic B-spline. The control polygon is coloured green and the curve is coloured white.*



**Colour Plate 2:** *Two splines that share a common control polygon (shown in green), but have different evaluation routine attributes. The white curve is a uniform cubic B-spline, and the red curve is a seventh degree Bézier curve.*



**Colour Plate 3:** *This plate illustrates a midpoint curve (shown in red), together with its control polygon (shown in green). As required by the representation, the curve interpolates the first and last control vertices, as well as the midpoint of the middle leg of the control polygon.*



**Colour Plate 4:** *This plate demonstrates the behaviour of a subdivision algorithm for a cubic Bézier curve. The original control polygon (shown in green) generates the green curve when its parameter varies on the interval* $[0,1]$. *The subdivision polygon (shown in white) generates the white curve when its parameter varies on the interval* $[0,1]$. *The subdivision polygon is constructed so that the white curve is identical to the portion of green curve corresponding to the interval* $[0,2/3]$.