Teaching B-splines Is Not Difficult!*

John Lowther and Ching-Kuang Shene
Department of Computer Science
Michigan Technological University
Houghton, MI 49931–1295
Email: {john,shene}@mtu.edu

Abstract

This paper describes the authors' approach of introducing important concepts and algorithms of B-splines to junior computer science students with the help of a pedagogical tool DesignMentor. This approach is non-mathematical and intuitive, and has been used and refined in the past six years.

Categories & Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; K.3.2 [**Computers and Education**]: Computer and Information Science Education–*Computer Science Education*

General Terms

Design

Keywords

Curves, B-splines, knot insertion, de Boor's algorithm

1 Introduction

After 30 years of intensive development, B-splines have become standard tools in computer graphics, geometric modeling, computer-aided design, and many other interdisciplinary areas [1, 4, 5]. While Foley et al [2] may not be the first computer graphics textbook to include a significant portion dedicated to splines, many recently published textbooks cover B-splines [6]. However, teaching and learning B-splines is a very challenging and demanding task to both instructors and students because of the involved mathematics. In the past six years, we developed an elective course **Introduction to Computing Geometry** for introducing the skills of handling geometric problems to our junior students [3]. Due to its importance, B-splines is a major topic. We use a non-mathematical and intuitive approach with the help of a pedagogical tool DesignMentor which is used to minimize the need of lengthy and tedious mathematical discussion. DesignMentor supports Bézier, rational Bézier, B-spline and NURBS curves and surfaces, and cross-sectional surface design [7, 8, 9], and is used world-wide at many universities in various courses. It provides an environment for students to visualize and verify important properties and algorithms.

Due to the use of DesignMentor, the coverage of B-splines is more extensive than the coverage in many undergraduate level textbooks. Our course is taught in a computer-equipped classroom. The B-splines unit takes six hours lectures. There are six steps in this unit. We start with a motivation telling students the reason for using B-splines, followed by the definition and exploration of some important properties such as the *local modification property*. Then, we continue with three extremely important topics: knot insertion, de Boor's algorithm, and curve subdivision. If time permits, we also discuss degree elevation. Knot insertion, curve subdivision and degree elevation are useful in cross-sectional surface design and curve and surface interpolation and approximation. This paper describes the content of these six steps and the way of using DesignMentor in a classroom setting that reduces the need of involved mathematics by using a learning-by-doing, intuitive approach.

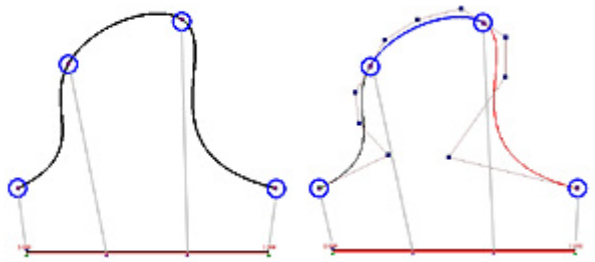
2 Step One: Motivation

Bézier curves are covered extensively before reaching the B-splines unit. As a motivation, we start with a disadvantage of Bézier curves, namely: the shape of a Bézier curve changes globally when a control point is modified. To overcome this problem, we need a curve whose shape only changes *locally* when a control point is modified. One solution is to connect a number of Bézier curves together and force them to act as a single one. Hopefully, any change made to a control point would only affect some neighboring curve segments. Since this composite curve is defined on a domain (*e.g.*, [0,1]), the domain is also divided into sub-intervals, each of which becomes the domain of a Bézier curve segment (Figure 1(a)). The composite curve is a B-spline curve, and the division points in [0,1] are its *knots*. A different subdivision yields a different B-spline curve.

How can we "blend" the Bézier curve segments together and find a new set of control points that defines the composite curve? This is *the* question we need to answer in the B-splines unit. Figure 1(b) shows the control points of three Bézier curves and the control points of the B-spline curve.

*This work was partially supported by the National Science Foundation under grants DUE-9653244, DUE-9952621 and DUE-0127401, and by a grant from the Michigan Research Excellence Fund 1998–1999.

SIGCSE'03, February 19-23, 2003, Reno, Nevada, USA.



(a) (b)

Figure 1: Knots of a B-spline Curve

We show the students this example and indicate that the answer is in the discussion of curve subdivision (Section 7).

3 Step Two: The Basics

To "blend" a number of Bézier curves into a single one, we use B-spline *blending* or *basis* functions. Given a degree p, a knot sequence $0 = u_0 \leq u_1 \leq \cdots \leq u_m = 1$, and a value $u \in [0,1]$, the i-th B-spline basis function of degree p, $N_{i,p}(u)$, is defined recursively:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u \in [u_i, u_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

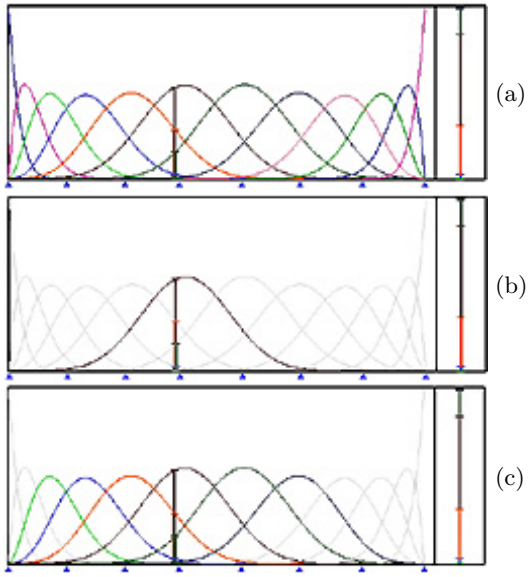
$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

Each $N_{i,p}(u)$ is computed from two B-spline basis functions of degree $p-1$, each of which is computed from two B-spline basis functions of degree $p-2$. Hence, $N_{i,p}(u)$ is recursively built from basis functions of degree 0. Note that $N_{i,p}(u) \geq 0$ for all i, and the sum of all $N_{i,p}(u)$'s is 1. Therefore, the basis functions at u "partition" the unit interval [0,1]. This is the well-known *partition of unity* property.

Two fundamentally important properties are discussed in class. Given $u \in [0,1]$, one can find a $[u_k, u_{k+1})$ that contains u. $N_{k,0}(u)$ is non-zero on $[u_k, u_{k+1})$ from the first formula, and $N_{k,1}(u)$ and $N_{k-1,1}(u)$ are non-zero because both use $N_{k,0}(u)$. Similarly, $N_{k,2}(u)$, $N_{k-1,2}(u)$ and $N_{k-2,2}(u)$ are non-zero because they use $N_{k,1}(u)$ and $N_{k-1,1}(u)$. Repeating this argument, we have **there are at most** $p+1$ **non-zero basis functions of degree** p **on** $[u_k, u_{k+1})$**:** $N_{k-p,p}(u)$**,** $N_{k-p+1,p}(u)$**,** $\cdots$**,** $N_{k,p}(u)$. On the other hand, since $N_{k,p}(u)$ uses $N_{k,p-1}(u)$ and $N_{k+1,p-1}(u)$, the former uses $N_{k,p-2}(u)$ and $N_{k+1,p-2}(u)$, and the latter uses $N_{k+1,p-2}(u)$ and $N_{k+2,p-2}(u)$. Hence, $N_{k,p}(u)$ uses $N_{k,0}(u)$, $N_{k+1,0}(u)$, $\ldots$, $N_{k+p,0}(u)$. Since $N_{j,0}(u)$ is non-zero on $[u_j, u_{j+1})$ for all j, $N_{k,p}(u)$ **is non-zero on** $[u_k, u_{k+p+1})$. We never emphasize the mathematical derivation in class. In fact, students have no difficulty in understanding the meaning and discussion of these two fundamental facts.

4 Step Three: Exploring the Basics

The next step is to explore the basics and justify the claim of "localization." Figure 2 shows screen-shots of DesignMentor's Partition of Unity Window. The curves are B-spline basis functions of degree 5, the middle vertical line marks the current value of u, and small triangles below the horizontal axis indicate knot positions. Students can display *all* basis functions (Figure 2(a)), a selected one (Figure 2(b)), or all non-zero basis functions on $[u_k, u_{k+1})$ that contains the current value of u. All zero basis functions on $[u_k, u_{k+1})$ are shown in light color. The partition of unity at u is shown by the right-most vertical bar of the window. Moreover, when the value of u moves passing u_{k+1} from $[u_k, u_{k+1})$ to $[u_{k+1}, u_{k+2})$, students can see one non-zero basis function is replaced by another. Hence, they can easily verify the two important properties discussed earlier.



(a) (b) (c)

Figure 2: Various B-spline Basis Functions

The B-spline curve $\mathbf{C}(u)$ of degree p defined by $n+1$ control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\ldots$, $\mathbf{P}_n$ is:

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i$$

where $m+1$ is the number of knots and $n = m+p+1$. Hence, $\mathbf{C}(u)$ is the weighted sum of the defining control points. Suppose we change the position of control point $\mathbf{P}_i$. The change made to $\mathbf{P}_i$ alters the term $N_{i,p}(u)\mathbf{P}_i$ only. Since $N_{i,p}(u)$ is zero outside of $[u_i, u_{i+p+1})$, the effect of changing $N_{i,p}(u)\mathbf{P}_i$ does not propagate outside of $[u_i, u_{i+p+1})$. Hence, if $\mathbf{P}_i$ is modified, the curve segment on $[u_i, u_{i+p+1})$ changes and the segments on $[0, u_i)$ and $[u_{i+p+1}, 1]$ do not. This justifies the claim that the modification made to a control point is localized. This is referred to as the *local modification* property.

DesignMentor helps visualize the local modification property vividly. Figure 3 shows two B-spline curves of degree 5 defined by 16 control points (*i.e.*, $n = 15$). The original curve is in dark color with control point $\mathbf{P}_9$ marked by a rectangle. If $\mathbf{P}_9$ is moved to a new location marked by a circle, the new curve is shown in light color. It is clear that only a portion of the original curve changes and the beginning and ending curve segments are the same. In fact, a simple calculation can reveal more. Suppose $\mathbf{P}_i$ is moved to $\mathbf{P}_i + \mathbf{v}$. The new curve is $\mathbf{D}(u) =$ "other terms" $+ N_{i,p}(u)(\mathbf{P}_i + \mathbf{v}) =$ ["other terms" $+ N_{i,p}(u)\mathbf{P}_i] + N_{i,p}(u)\mathbf{v} = \mathbf{C}(u) + N_{i,p}(u)\mathbf{v}$. Thus, the new curve is the sum of the original and a "shift" $N_{i,p}(u)\mathbf{v}$. Since $N_{i,p}(u)$ is non-zero on $[u_i, u_{i+p+1})$, the original curve segment on $[u_i, u_{i+p+1})$ is shifted in the direction of $\mathbf{v}$ as shown in Figure 3.

Figure 3: The Impact of Changing a Control Point

How about knots? They only appear implicitly in the definition of a B-spline curve. Normally, all knots are equally spaced, and changing the position of a knot may produce an unpredictable change to the shape of the curve. If the first $p+1$ knots are set to 0 and the last $p+1$ knots are set to 1, where p is the degree of a B-spline curve, the resulting curve is tangent to both ends of the control polygon (Figure 4). This produces a *clamped* curve and is commonly used. Otherwise, the curve may not be tangent to the control polygon and we have an *open* curve. By replicating some knots and/or control points, we can force the curve to become *closed*.

(a) Clamped (b) Open (c) Closed

Figure 4: Three Types of Knots

5 Step Four: Knot Insertion

After exploring the basics and other concepts, we move forward and discuss *knot insertion*, a fundamentally important concept. Knot insertion means adding one more knot into the knot sequence *without* changing the shape of the curve. We do not derive the knot insertion algorithm because it is a tedious and perhaps not an enlightening process. Instead, we convert the computation algorithm to a visualization. Suppose $t \in (0,1)$ is to be inserted. The knot insertion algorithm has the following steps: **(1)** find a $[u_k, u_{k+1})$ that contains t, **(2)** retrieve the $p+1$ control points $\mathbf{P}_{k-p}$, $\mathbf{P}_{k-p+1}$, ..., $\mathbf{P}_{k-1}$, $\mathbf{P}_k$, **(3)** find new points $\mathbf{Q}_{k-p+1} \in \overline{\mathbf{P}_{k-p}\mathbf{P}_{k-p+1}}$, $\mathbf{Q}_{k-p+2} \in \overline{\mathbf{P}_{k-p+1}\mathbf{P}_{k-p+2}}$, ..., $\mathbf{Q}_k \in \overline{\mathbf{P}_{k-1}\mathbf{P}_k}$, and **(4)** replace the original control points $\mathbf{P}_{k-p+1}$, ..., $\mathbf{P}_{k-1}$ by $\mathbf{Q}_{k-p+1}$, $\mathbf{Q}_{k-p+2}$, ..., $\mathbf{Q}_{k-1}$, $\mathbf{Q}_k$ (Figure 5), and the original knot sequence with $0 = u_0 \le \cdots u_k \le t < u_{k+1} \le \cdots \le u_m = 1$. Now, we have $m+2$ knots and $n+2$ control points, and the identity $(m+1) = (n+1)+p+1$ still holds. In this way, the corner at $\mathbf{P}_{k-j}$ is cut by $\overline{\mathbf{Q}_{k-j}\mathbf{Q}_{k-j+1}}$.

For $k-p+1 \le i \le k$, $\mathbf{Q}_i$ is computed as $\mathbf{Q}_i = (1-a_i)\mathbf{P}_i + a_i\mathbf{P}_{i+1}$, where $a_i = (t-u_i)/(u_{i+p}-u_i)$. Instead of proving this formula, we teach our students how to interpret it in a visual way. We ask them to stack the following intervals together: $[u_k, u_{k+p})$, $[u_{k-1}, u_{k+p-1})$, ..., $[u_{k-p+1}, u_{k+1})$ as in Figure 6. The vertical line at t intersects each interval at a point. The distance from this point to its left end u_i is $t-u_i$, and the length of this interval is $u_{i+p}-u_i$. Therefore, a_i is the ratio of the position of t in $[u_i, u_{i+p})$. In our six-year experience, almost all students in our class can quickly recall and use this relation to insert a knot. A simple modification to this procedure will insert a knot multiple times.

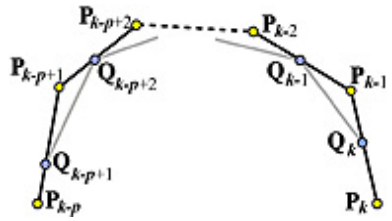


Figure 5: Knot Insertion via Corner Cutting

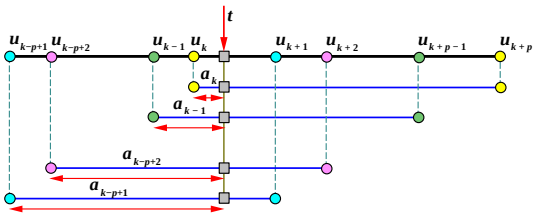
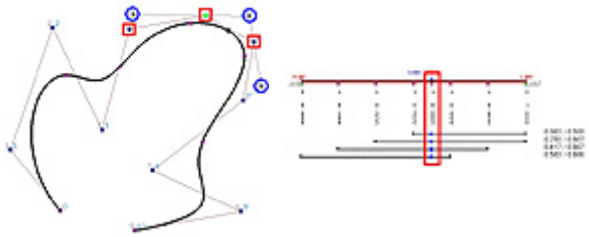


Figure 6: Computing $\mathbf{Q}_j$'s

When the Knot Insertion Window is activated, DesignMentor displays all control points and the new ones as if u has been inserted. In Figure 7(a), the new control points are marked with squares and the old ones that will be removed by corner cutting are marked with circles. The diagram for computing the a_i's is shown in Figure 7(b). The value of u can be changed and the display is updated on-the-fly. Students can do, re-do and reset multiple knot insertion operations. In this way, they will learn knot insertion without any difficulty.



(a) (b)

Figure 7: Knot Insertion Window

6 Step Five: De Boor's Algorithm

While it is possible to compute $\mathbf{C}(u)$ for a given u with the recurrence relation discussed in Section 3, this is not recommended because it is tedious and not robust. With knot insertion, we can discuss de Boor's algorithm for computing $\mathbf{C}(u)$ in a geometric way. De Boor's algorithm states that

for a given u, inserting it as a new knot p times yields $\mathbf{C}(u)$. More precisely, after the p-th insertion, one control point remains which is $\mathbf{C}(u)$. In class, we ask students to activate the display of de Boor's algorithm. DesignMentor displays all intermediate control polygons computed in the process of multiple knot insertion (*i.e.*, the de Boor net) as shown in Figure 8(a). As the value of u is dragged for tracing the curve, the de Boor net is updated on-the-fly.

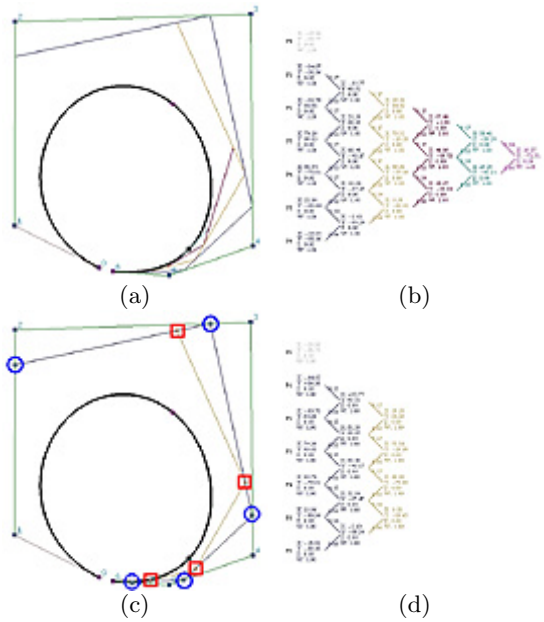


Figure 8: De Boor's Algorithm

To help students understand this multiple insertion process, DesignMentor has a Stepwise Computation Window. A click on the `Step` button shows the next level of knot insertion. Figure 8(c) and (d) show the second insertion and its actual computation, respectively. After performing the necessary number of insertions to yield a single point, the computation process has a triangular shape similar to that of de Casteljau's algorithm (Figure 8(b)).

7 Step Six: Curve Subdivision

The next application of knot insertion is curve subdivision. In a design process, one part of a curve may have the desired shape. Thus, this curve may be divided into two so that we can ignore the "good" segment and work on the other. Note that subdividing a B-spline curve cannot change its shape and each segment is a new B-spline curve defined by a set of new control points and knot sequence. Curve subdivision is easy. An application of de Boor's algorithm at u yields a de Boor net. Then, starting with $\mathbf{P}_0$ and moving toward $\mathbf{P}_n$ on the de Boor control net, select each encountered control point until $\mathbf{C}(u)$ is reached. These control points define the first curve segment. Then, starting with $\mathbf{C}(u)$ and selecting each encountered control points yields the second curve segment. The knot sequence of the first (*resp.*, second) curve consists of all knots from $u_0 = 0$ to u (*resp.*, from u to $u_m = 1$) with u duplicated $p + 1$ times.

DesignMentor's Subdivision Window displays a de Boor net as shown in Figure 9(a) with the original and the two new sets of control points, the "selection path" marked in different colors. A student can drag the value of u and divide the curve into two as shown in Figure 9(b).

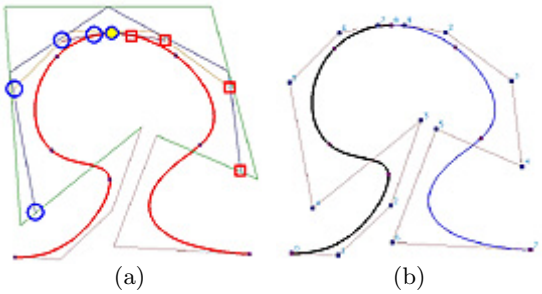


Figure 9: Curve Subdivision

With curve subdivision, we can finally reveal the link between B-spline and Bézier curves as discussed in Section 2. If a given B-spline curve is divided at its knots, each curve segment is tangent to the left and right ends of its control polygon and there is no internal knot. Hence, each segment is a Bézier curve, and the relationship between the control points of the B-spline and those of the Bézier curve segment is established by curve subdivision! Figure 10 has a B-spline curve of degree 5 defined by 7 control points (*i.e.*, $n = 6$) and knot sequence $0, 0, 0, 0, 0, 0, 0.5, 1, 1, 1, 1, 1, 1$. Since this is a *clamped* curve, the first and last knots are repeated $p+1 = 6$ times. The only internal knot is 0.5. If the curve is divided at $u = 0.5$ as shown in the figure, we have two new control polygons each of which defines a B-spline curve. However, since these two B-spline curves have no internal knots, they are actually Bézier curves.

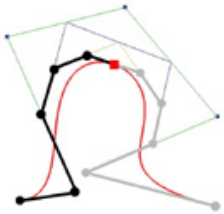


Figure 10: Dividing a B-spline into Bézier Segments

8 Assignments

In addition to several hand-calculation problems such as computing $N_{i,p}(u)$ for a given u, inserting a knot multiple times, and performing curve subdivision, the B-spline curve unit includes a programming assignment. Students are given a scaled-down environment of DesignMentor and are asked to (1) read in the control points, knots and degree, (2) display the B-spline curve using OpenGL (actually with one OpenGL function call), and (3) trace the curve using de Boor's algorithm. The programming skill for this assignment is important, because it is required for implementing de Boor's algorithm for B-spline surfaces and interpolation and approximation.

9 Findings

Contrary to the belief that B-splines are too difficult to be taught at undergraduate level, our six-year experience shows that virtually all students did the programming assignment

correctly except for occasional minor programming errors. We believe that this success is due to the use of a non-mathematical and intuitive approach and the learning-by-doing style supported by DesignMentor.

This course is evaluated with pre- and post- tests and an attitudinal survey. Of the 22 questions in the pre- and post-tests, two are directly related to the B-splines unit. One asks students to comment on their skill in B-spline curves and the other on B-spline surfaces. The scale is from 1 (none) to 5 (excellent). The means and standard deviations of the answers to these two questions in the pre-test (*resp.*, post-test) are 1 and 0 (*resp.*, 4 and 1), respectively. The means (*resp.*, standard deviations) of the differences between pre- and post- tests are 2.57 and 2.43 (*resp.*, 1.18 and 1.24), respectively. Therefore, students enter this course virtually have no knowledge in B-splines, and, at the end, they have a quite solid gain. We also ask if students like the B-splines topic in an anonymous attitudinal survey. The mean and standard deviation of B-spline curve (*resp.*, surface) are 4.13 and 0.96 (*resp.*, 4.27 and 0.44), respectively. This shows that students really like this seemingly "difficult" topic. This pattern occurs in all pre- and post- tests and attitudinal surveys of the past six years. Therefore, we can safely conclude that the fundamentals of B-splines are not difficult to teach if we handle it properly with the help of DesignMentor.

10 Conclusions

We have presented our way of introducing the fundamentals of B-splines to junior students. Since the use of B-splines has become a basic design tool in many graphics systems (*e.g.*, trueSpace, LightWave 3D, 3D Studio and Maya) and is widely used in many interdisciplinary areas, it is the time for computer science educators to seriously consider the way of incorporating this important topic into a typical curriculum. We hope this paper may serve as a starting point. We are continually developing DesignMentor to support more features. Interested readers may find more about our work, our web-based textbook, user guides, DesignMentor and other tools, and future announcements, at the following site:

`http://www.cs.mtu.edu/~shene/NSF-2`

Acknowledgment

The first version of DesignMentor was implemented by Yuan Zhao and Yan Zhou, our former graduate students. Former graduate student Budirijanto Purnumo started the implementation of Version 2, and John Fisher, currently an undergraduate student, contributes significantly to this new system that will be released in the future.

References

[1] E. Cohen, R. F. Riesenfeld and G. Elber, *Geometric Modeling with Splines: An Introduction*, A K Peters, 2001.

[2] J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Second Edition, Addison-Wesley, 1990.

[3] J. L. Lowther and C.-K. Shene, Computing with Geometry as an Undergraduate Course: A Three-Year Experience, *The Thirty-second SIGCSE Technical Symposium*, 2001, pp. 119–123.

[4] L. Piegl and W. Tiller, *The NURBS Book*, Springer-Verlag, 1995.

[5] D. F. Rogers, *An Introduction to NURBS with Historical Perspective*, Academic Press, 2001.

[6] P. Shirley, *Fundamentals of Computer Graphics*, A K Peters, 2002.

[7] Y. Zhao, J. L. Lowther and C.-K. Shene, A Tool for Teaching Curve Design, *The Twenty-ninth SIGCSE Technical Symposium*, 1998, pp. 97–101.

[8] Y. Zhao, Y. Zhou, J. L. Lowther and C.-K. Shene, Cross-Sectional Design: A Tool for Computer Graphics and Computer-Aided Design Courses, *ASEE/IEEE Frontiers in Education*, Vol. II (1999), pp. (12b3-1)–(12b3-6).

[9] Y. Zhou, Y. Zhao, J. L. Lowther and C.-K. Shene, Teaching Surface Design Made Easy, *The Thirtieth SIGCSE Technical Symposium*, 1999, pp. 222–226.