Teaching Surface Design Made Easy[*]

Yan Zhou, Yuan Zhao, John L. Lowther and Ching-Kuang Shene[†]
Department of Computer Science
Michigan Technological University
Houghton, MI 49931–1295
Email: `[yzhou|yzhao|john|shene]@mtu.edu`

1 Introduction

All popular computer graphics textbooks have chapters on curves and surfaces [1, 5, 6, 7]. Teaching these chapters could be one of the most challenging tasks in a computer graphics course because of the involved mathematics and the difficulty of visualizing the anticipated effects of the theoretical results. We certainly can skip these chapters and only rely on the polyhedron world, because Gouraud's or Phong's shading algorithms could make polyhedra objects appear as realistic curvilinear ones. Unfortunately, real world applications such as ship hull and car body design are usually curvilinear and may only be approximated using polyhedron models. Thus, to address this problem and to make teaching surface design easier, we have designed a pedagogical tool as part of our NSF supported project [8]. One of our goals is to provide the students with an interactive environment which is used to visualize, experiment and verify important and fundamental concepts and algorithms.

While there are good textbooks on curve and surface design [3, 4, 9], we can only find two pedagogical aids. Yen's video program [11] only provides one way information flow. Rockwood and Chambers [10] describe a multimedia tutorial on computer aided geometric design which runs on Windows. It is basically a tutorial that introduces concepts using some animation but lacks a fully interactive environment. Moreover, its surface capability is limited. To remedy this situation, we presented a curve design system last year [12]. The surface system presented here works with our curve system to provide a reasonably complete support to many basic concepts in surface design.

In what follows, Section 2 addresses the system design issues, Section 3 introduces the user interface, Section 4 discusses basic features, Section 5 presents some advanced techniques, Section 6 focuses on cross-section design, and, finally, Section 7 has our conclusion.

2 System Issues

The goal of this system is to provide a software tool for students to learn and understand the fundamentals of surface design. Hence, its emphasis is to help students to visualize, experiment and verify important elements in surface design. Currently, this system supports all commonly seen surface types, including Bézier, rational Bézier, B-spline and NURBS (Non-Uniform Rational B-Spline) surfaces. It can also display algebraic surfaces using Bloomenthal's implicit surface polygonizer [2]. The details can be found in [3, 4, 9] and will not be repeated here.

Since we hope this system can be distributed widely, it has to be highly portable. We chose OpenGL for graphics programming and GLUT (OpenGL Utility Toolkit) for the system interface. Both are available on virtually every platform, including Unix, Linux and Windows 95/NT. If OpenGL is not available, then a free OpenGL clone, Mesa, can be used. We have successfully tested our systems using Mesa on SunOS, Solaris, and Linux. Windows 95 and NT versions are planned but not available yet. However, users should keep in mind that without a hardware OpenGL accelerator, response time could be slow. A Pentium 200MHz machine usually delivers satisfactory performance if geometric transformations are not used frequently.

[*]This work was partially supported by the National Science Foundation under grant DUE-9653244. The fourth author was also partially supported by the National Science Foundation under grant DUE-9752244 and CCR-9696084 (formerly CCR-9410707).

[†]Communicating author

3 The User Interface

Our system uses of five major windows, the *drawing window*, *tracing window*, *control point window*, *partition of unity window* and *control panel*. There are other windows; but, they are less important to this paper. All windows are grouped together and shown in Figure 1.

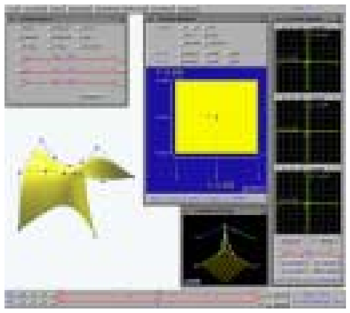


Figure 1: The User Interface

In the figure, the background is the drawing window. The top row of this window provides several menu items for the user to perform various tasks such as loading and saving a scene, defining a surface, selecting the method for displaying the surfaces in various styles, and performing advanced surface design tasks. On the bottom there are two slides for the user to translate and rotate the selected surface and to perform scene zooming.

The top left smaller window is the control panel used to change the colors of various displayable components (*i.e.*, background color, surface color and so on). The window in the middle is the tracing window. The large square at the center of this window is the domain of the surface (*i.e.*, $[0,1] \times [0,1]$). A small disk, the *position indicator*, is shown in this square indicating a point (u,v) in the domain. As the position indicator moves in the domain, its corresponding point $\mathcal{S}(u,v)$ moves on the surface. The horizontal and vertical directions of this square are the u- and v-direction. The values of knots and their multiplicities are also shown. The little window below the tracing window is the partition of unity window in which the coefficient of the selected control point is shown.

The window at the right is the control point window. After the user selects a control point with the right mouse button, this window appears automatically. Then, the user can drag a small disk in one of the three subwindows to change the position of the selected control point. These subwindows are used for changing the coordinates in the xy-, xz- and yz-plane. The bottom part of the control point window has a slide for the user to modify the weight of the selected control point, if the current surface type is rational Bézier or NURBS.

4 Basic Features

4.1 Moving Control Points

The user must tell the system (1) the number of rows and the number of columns, (2) the degrees in the u- and v-directions, and (3) the type of the surface (*i.e.*, Bézier, rational Bézier, B-spline and NURBS). Then, the system generates a set of planar control points which defines a flat surface (Figure 2(a)). Control points can be moved in all directions until its shape is satisfied (Figure 2(b)). The user can ask the system to display the control points, control net, IDs of control points and other information. The generated surface can be translated and rotated, shaded or left as a wireframe. The scene can also be zoomed in and out to fit the surface in the drawing window.

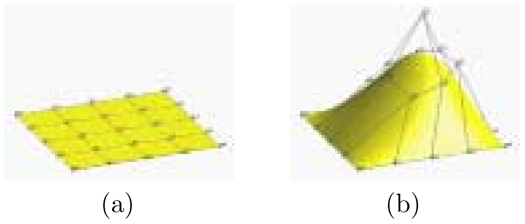


Figure 2: Shaping a Surface

4.2 Changing Weights

To modify the shape of a NURBS surface, in addition to moving its control points, one can also change the weights of the control points. In general, increasing (*resp.*, decreasing) the weight of a selected control point pulls (*resp.*, pushes) the surface toward (*resp.*, away from) that control point.

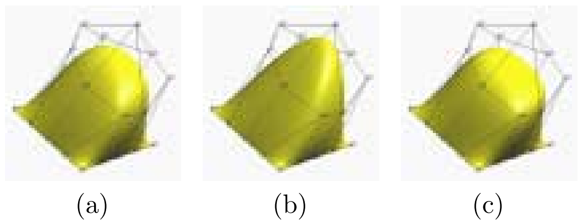


Figure 3: Modifying Weights

Figure 3(a) is a degree $(3,3)$ NURBS surface defined by a 5×5 control point grid with all weights being 1s. This is actually a B-spline surface. Suppose control point $\mathbf{P}_{22}$ is selected, which is the one at the peak of the control net. If w_{22} is increased to 5, the resulting surface is shown in (b). It is clear that the surface is pulled toward $\mathbf{P}_{22}$. If w_{22} is decreased to 0.3, we have the result in (c) in which the surface is pushed away from $\mathbf{P}_{22}$. While in real applications weights are positive, our system supports negative weights. However, in this case, the convex hull property no longer holds. If the weight of a control point is zero, this control point does not affect the generation of the surface.

4.3 De Boor's Algorithm

De Boor's algorithm provides a simple and easily understood way for computing a point $\mathcal{S}(u,v)$ on the surface and is a fundamental algorithm in NURBS theory. In fact, de Boor's algorithm for surfaces is a multiple application of de Boor's curve algorithm. Suppose the rows of control points correspond to the u-direction. Since each row defines a NURBS curve in u, applying de Boor's algorithm to row i at u yields a point $\mathbf{P}_i(u)$ on the NURBS curve. As a result, we have a set of new control points $\mathbf{P}_1(u)$, ..., $\mathbf{P}_m(u)$, which in turn defines a new NURBS curve. Applying de Boor's algorithm to this set of new control points at v yields a point and this is the point $\mathcal{S}(u,v)$ on the NURBS surface.

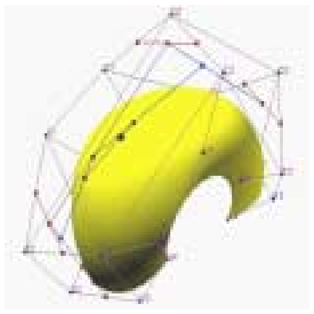


Figure 4: De Boor's Algorithm

Our system can display all intermediate calculations of de Boor's algorithm. Figure 4 is an example, where $(u,v) = (0.3, 0.6)$. In the figure, control points on row i (*i.e.*, $\mathbf{P}_{i,0}$, $\mathbf{P}_{i,1}$, ..., $\mathbf{P}_{i,4}$) define a new point. These points, in turn, are used to compute the corresponding point on the surface. The point on the surface is shown as a small red sphere.

5 Advanced Algorithms

5.1 Knots and Knot Curves

A knot vector is required for each of the u- and v-direction. For a fixed knot, say u_i, $\mathcal{S}(u_i,v)$ is a curve on the surface $\mathcal{S}(u,v)$. This is called a *knot curve*. Thus, for all knots in the u- and v-direction, a knot curve can be drawn on the surface. All knot curves are isoparametric curves because they are defined by fixing one of the two parameters. Figure 5(a) shows a NURBS surface with its knot curves. The knot vector in both directions is 0*4, 0.5, 1*4. Figure 5(b) shows another NURBS surface whose knot vector in the u-direction is 0*4, 0.5, 0.75, 1*4 and the knot vector in the v-direction is the same as that of (a). Therefore, we see two knot curves in the u-direction.

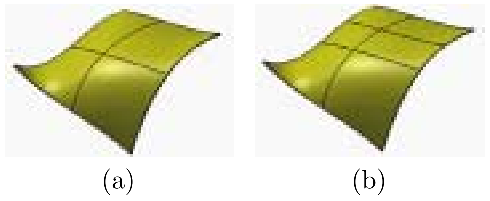


Figure 5: Knot Curves and Knot Insertion

Our system allows the user to drag a knot. Changing a knot affects the shape of the surface. However, since there is no satisfactory relationship between the change of the knots and the change of the shape of the resulting surface, it is not recommended to change the shape of a NURBS surface by modifying knots.

5.2 Knot Insertion

The idea of knot insertion is to insert a new knot into an existing knot vector while keeping the shape of the surface *unchanged*. The same knot can be inserted multiple times, making it a knot of multiplicity larger than 1. Figure 5(b) is obtained by inserting a new knot 0.75 into the u-direction of the surface in (a). Note that the shapes of the surfaces are the same.

Our system allows the user to drag the position indicator (u,v) in the tracing window and to insert a new knot into the knot vector of the u- or v-direction. For example, if the position indicator has coordinate $(0.35, 0.7)$ and the user inserts a new knot in the v-direction, then the knot vector in the v-direction will have a new knot at 0.7.

5.3 Degree Elevation

Degree elevation is the technique of increasing the degree of a curve or a surface without changing its shape. The number of control points is also increased. Degree elevation and knot insertion are important to curve and surface design, since in many applications (*e.g.*, cross-section design) two curves are required to be *compatible*. Two curves are compatible if they have the same degree, the same knot vector and the same number of control points. Thus, degree elevation can bring two curves to the same degree, while knot insertion can combine two different knot vectors into a common one. Both techniques do not change the shape of the curve. Our system can increase the degree of a NURBS surface by one in the selected direction.

5.4 Surface Subdivision

Surface subdivision allows the user to subdivide a NURBS surface into subpatches so that he/she could concentrate on some subpatches that require further work while keep the other "good" subpatches unchanged. Each subpatch is a NURBS surface and has its own knot vector and control points. But, the degree of each such patch is the same as the original. When the user moves the control points of a subpatch, continuity along patch boundaries could be destroyed. Our system allows the user to specify if this level of continuity should be preserved along patch boundaries.

Figure 6(a) is the original surface, (b) is obtained by subdividing the original along the isoparametric curve of $u = 0.75$, and (c) is obtained by subdividing the left subpatch in (b) along the isoparametric curve $v = 0.5$. In our system, these new patches do not have their own domains. Instead, they share the original domain so that when the position indicator in the tracing window moves into the domain of a subpatch, that subpatch is activated and becomes the current patch (Figure 6(d)).

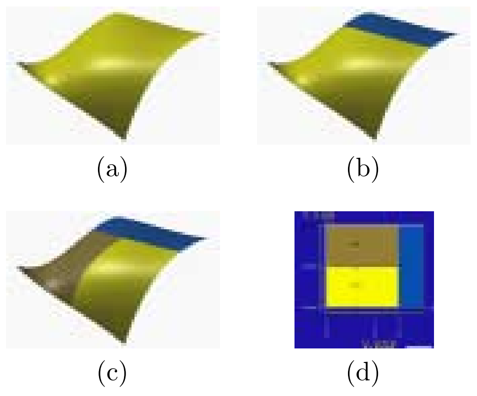


Figure 6: Surface Subdivision

6 Cross-Section Design

Cross-section design is a commonly used surface design technique. In general, cross-section design requires one or more *profile curves* that define the surface profile, and a *trajectory curve* that describes the way of moving and perhaps scaling of the profile curves to generate the surface. In this way, the design of surfaces reduces to the design of curves. The cross-section design module of our system supports ruled surfaces, surfaces of revolution, swung surfaces, swept surfaces and skinned surfaces.

This system is designed to be used with our curve system [12]. The user chooses cross-section design in the surface system, which forks a new process to run the curve system. Then, the user designs profile and trajectory curves and generates the desired surface with the curve system. All data (*i.e.*, control points, degrees and knot vectors) are transfered to the surface system, which will, in turn, display the resulting surface. Due to page limits, we only discuss ruled surfaces, surfaces of revolution and skinned surfaces.

6.1 Ruled Surfaces

The simplest surface that can be obtained with cross-section design is a ruled surface. A *ruled surface* requires two compatible NURBS curves, and is generated by connecting the corresponding points with a line segment. As u moves from 0 to 1, the locus of this segment generates a ruled surface. Figure 7(a) shows two circles with the bottom one rotated by an angle and (b) shows the generated ruled surface which is a hyperboloid of one sheet.

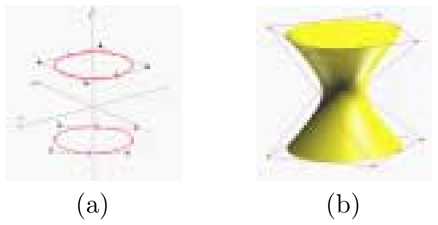
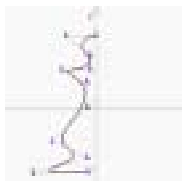


Figure 7: A Hyperboloid of One Sheet

6.2 Surface of Revolution

A *surface of revolution* is generated by revolving a profile curve about an axis. In our system, the axis of revolution is always the z-axis. Figure 8(a) is a NURBS profile curve defined by 11 control points, degree 3, and a knot vector of 15 uniformly spaced knots with both ends "clamped" so that the curve is tangent to the first and the last leg of the control polygon. Figure 8(b) is the generated surface of revolution.


(a)

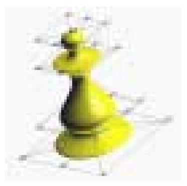
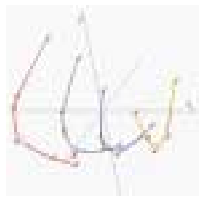

(b)

Figure 8: A Surface of Revolution

6.3 Skinned Surface

Cross-section design has a very important application in car body design. Designers create a series of key curves; but they do not know the surface and want to find one that can contain all of these key curves. Hopefully, the resulting surface will follow their design. The given curves are compatible profile curves and the resulting surface is a *skinned surface*. In fact, the skinned surface "interpolates" the profile curves. In Figure 9, (a) shows four degree 3 NURBS curves to be used as profile curves, and (b) is the generated skinned surface.


(a)

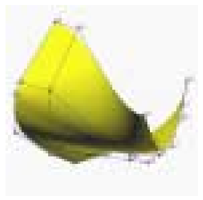

(b)

Figure 9: A Skinned Surface

7 Conclusion

We have presented important features of our surface design tool. This tool has been used in our *Introduction to Computing with Geometry* course in the Winter of 1998 and will be used again in the coming winter. Both the curve and surface systems are available at `http://www.cs.mtu.edu/~shene`. Also available are a set of online course notes, online user guides of both systems and other course materials. Although we have tried our best to incorporate many features into our tools, there are other important topics and powerful algorithms not included. These include blossoming principle, triangular patches, more sophisticated surface interpolation schemes, wavelets, surface tessellation and others. We hope that these could be added in the future so that our systems will not only provide elementary pedagogical aids, but also serve as learning environments for advanced courses such as computer aided design, geometric modeling, advanced computer graphics and visualization.

References

[1] Edward Angel, *Interactive Computer Graphics: A Top-down Approach with OpenGL*, Addison-Wesley, 1997.

[2] Jules Bloomenthal, An Implicit Surface Polygonizer, in *Graphics Gems IV*, edited by Paul S. Heckbert, Academic Press, 1994, pp. 324–349.

[3] Gerald Farin, *NURB Curves and Surfaces*, A K Peters, 1995.

[4] Gerald Farin, *Curves and Surfaces for CAGD: A Practical Guide*, forth edition, Academic Press, 1997.

[5] James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes, *Computer Graphics: Principles and Practice*, second edition, Addison-Wesley, 1990.

[6] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes and Richard L. Phillips, *Introduction to Computer Graphics*, Addison-Wesley, 1994.

[7] Donald Hearn and M. Pauline Baker, *Computer Graphics*, second edition, Prentice Hall, 1994.

[8] John L. Lowther and Ching-Kuang Shene, Geometric Computing in the Undergraduate Computer Science Curricula, *The Journal of Computing in Small Colleges*, Vol. 13 (1997), No. 2 (November), pp. 50–61.

[9] Les Piegl and Wayne Tiller, *The NURBS Book*, Springer-Verlag, 1995.

[10] Alyn Rockwood and Peter Chambers, *Interactive Curves and Surfaces: A Multimedia Tutorial on CAGD*, Morgan Kaufmann, 1996.

[11] Jonathan Yen, *Knotty: A B-Spline Visualization Program*, Part I and II, Morgan Kaufmann, 1993.

[12] Yuan Zhao, John Lowther and Ching-Kuang Shene, A Tool for Teaching Curve Design, *The Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, February 25 - March 1, 1998, Atlanta, Georgia, 1998, pp. 97–101.