Homework #3: B-splines; rational curves; surfaces; programming in OpenGL [115 points]
Due Date: Tuesday, 23 February 1999

Theory and implementation

The last two problems in this assignment (Problems 3 and 4) ask you to develop the theory of and implement in OpenGL the viewing of a NURBed torus. Problem 4 is the programming part and, as explained in the guidelines given with Homework 1, you may work on this programming part in groups of up to three students and hand in a joint write-up. You must still hand in individual solutions to problems 1, 2 and 3.

Problem 1. [15 points]

Find the periodic, uniform, C^2-continuous, cubic, polynomial spline curve with four segments that interpolates the four points $(1, 0)$, $(0, 1)$, $(-1, 0)$, and $(0, -1)$ in the plane. That is, your spline curve F should consist of four cubic polynomial segments $F([0 \,..\, 1])$, $F([1 \,..\, 2])$, $F([2 \,..\, 3])$, and $F([3 \,..\, 4])$. It should satisfy $F(0) = (1, 0)$, $F(1) = (0, 1)$, $F(2) = (-1, 0)$, $F(3) = (0, -1)$, and $F(t) = F(t + 4)$ for all t. And it should have C^2 continuity at all four joints. Draw a picture that shows the four de Boor points of the spline F and the four Bézier points of each of its four cubic segments.

Find the Cartesian coordinates of the point $F(1/2)$, one of the two points where your spline curve cuts the line $X = Y$. Is the point $F(1/2)$ inside, on, or outside of the unit circle?

Problem 2. [20 points]

We define a quadratic, rational curve segment $F([0 \,..\, 1])$ in the plane by specifying its three Bézier sites $(w; x, y)$ as follows:

$$\begin{aligned} f(0,0) &= (1; -1, -1) \\ f(0,1) &= (0; c, 0) \\ f(1,1) &= (1; 1, 1), \end{aligned}$$

where c is a nonzero, real parameter. Note that the corresponding Bézier points are the point $(-1, -1)$, the point at infinity in the horizontal direction, and the point $(1, 1)$. Give a formula for the polar value $f(t_1, t_2)$ and a formula for the diagonal value $F(t)$. Find the Cartesian coordinates of the points $F(1/2)$ and $F(\infty)$. Find the slope of the curve F at the points $F(1/2)$ and $F(\infty)$. Sketch the curve F when $c = 1/2$ and when $c = 3$.

Your sketches should include the four points $F(0)$, $F(1/2)$, $F(1)$, and $F(\infty)$, as well as the four corresponding tangent lines.

The curve traced out by F is a conic section—in fact, an ellipse. Find the implicit equation of that ellipse. (Hint: Your formula for $F(t)$ will have the form $F(t) = (w(t); x(t), y(t))$ for quadratic polynomials w, x, and y. Find the implicit equation relating the Cartesian coordinates $X = x/w$ and $Y = y/w$ by using a resultant to eliminate t from the two quadratic equations $w(t)X - x(t) = 0$ and $w(t)Y - y(t) = 0$.)

Problem 3. [20 points]

Let r and R be real numbers with $0 < r < R$. As the angles α and β vary, the varying point $V = (X, Y, Z)$ given by

$$\begin{aligned} X &:= (R + r\cos\alpha)\cos\beta \\ Y &:= (R + r\cos\alpha)\sin\beta \\ Z &:= r\sin\alpha \end{aligned}$$

traces out a *torus*, that is, the surface of an ideal bagel. The torus has rotational symmetry about the Z axis, which passes through the middle of the hole. Any plane π through the Z axis cuts the torus in two circles of radius r, whose centers lie along the line where π cuts the XY plane, R units on each side of the origin. Varying α moves the point V around one of those circles. Varying β rotates the plane π around the Z axis.

Find a biquadratic, rational parameterization of this torus. That is, express each of the homogeneous coordinates $[w; x, y, z]$ of the varying point V as a polynomial in two parameters that has degree at most 2 in each parameter when the other is held fixed. For consistency in notation, use Q and T as your two parameters, writing

$$V(Q;T) = [w(Q;T); x(Q;T), y(Q;T), z(Q;T)]$$

for certain polynomials w, x, y, and z. Hint: Let $Q := \tan(\alpha/2)$ and $T := \tan(\beta/2)$.

Homogenize and polarize your parameterization. You may carry out these two steps in either order; the result will be the same. If you homogenize first, use p as the weight coordinate for the Q parameter, writing $Q = q/p$, and use s as the weight coordinate for the T parameter, writing $T = t/s$. If you polarize first, split the T parameter into two separate parameters T_1 and T_2, and split the Q parameter into Q_1 and Q_2. The homogenized, polar form v of V will have the form

$$v((p_1; q_1), (p_2; q_2); (s_1; t_1), (s_2; t_2)) = [w; x, y, z],$$

where each coordinate w, x, y, and z is a polynomial in the eight variables p_1, q_1, p_2, q_2, s_1, t_1, s_2, and t_2.

One quarter of the torus consists of points that have Y and Z positive. (If you followed the hint above, those points will correspond, under your parameterization, to parameter pairs $(Q;T)$ where both Q and T are positive.) Describe this portion of the torus as a biquadratic, rational Bézier surface patch, that is, as a rectangular, tensor-product patch of degree $(2;2)$. In particular, give the coordinates of the nine Bézier sites of this patch. (If you followed the hint above, the patch will be $V([0\mathrel{..}\infty] \times [0\mathrel{..}\infty])$.) Hint: Don't be distressed if one of the nine Bézier sites turns out to be the zero site, that is, the site all four of whose coordinates are zero.

Problem 4. [60 points]

In the previous problem, you developed a representation of a torus by using a splined surface consisting of four biquadratic, rational Bézier surface patches. In this problem you will implement a slightly more general toroidal surface, using the facilities of the OpenGL graphics library on the SGI (or possibly other) workstations. In addition to modeling the torus, you will use the OpenGL facilities to render an image of the torus on the screen and to set up a rudimentary user interface for playing with the parameters defining this generalized torus, as well as controlling the viewer's position relative to the torus. Unlike the previous theoretical problems, in this problem the emphasis is on implementing things that you already know in the context of a widely available (and very nice to use) graphics platform.

In this problem you must use the C programming language. OpenGL will give you access to a set of graphics libraries that you can use to implement modeling and rendering operations. In particular, OpenGL provides very good NURB support, so to specify your spline you need do nothing more than calculate the appropriate Bézier control sites. But since your goal is to produce and interact with an image, you must also become familiar with some of the OpenGL facilities dealing with rendering. Specifically, you will need to understand how to set up lights illuminating your model, how to define the material properties of the surface of your model, how to specify the viewpoint, and how to use z-buffering (the depth buffer) for hidden-surface elimination and double-buffering to create smooth animations. OpenGL provides excellent facilities for dealing with all these issues—each of them will require only a few additional lines of code in your program. Nevertheless, especially if you are unfamiliar with basic graphics concepts, you may want to spend a couple of hours just browsing through the rendering chapters of any standard graphics text. The OpenGL manual itself is a pretty good reference—the course reader itself contains the NURBS section of the older GL manual (the precursor to OpenGL). A more precise description of what you will need to use is given later in this handout.

In order to interact with your image, you will build a set of interactors using OpenGL and a "forms" package provided for you. You will not need anything but menus and sliders, the latter for inputting real parameters to your program. While your program is running you should be able to interactively modify the parameters defining the generalized torus, as well as the position of the view point, and immediately see the results. Documentation on the forms package is contained in the course reader. We will be using a new version of forms that has support for OpenGL canvasses, and the forms manual will be available in the class directory and in the web page of the course.

Specifying a generalized torus

As in Problem 3, we consider the torus given by the equations

$$\begin{aligned} X &:= (R + r\cos\alpha)\cos\beta \\ Y &:= (R + r\cos\alpha)\sin\beta \\ Z &:= r\sin\alpha, \end{aligned}$$

where $0 < r < R$ and α and β are free parameters. As you discovered in Problem 3, we can decompose the surface of that torus into four rational, biquadratic surface patches, according to the signs of Y and Z. The middle Bézier site of each of the resulting four patches turns out to be the zero site—the site whose four coordinates are all 0.

For this problem, we will generalize that torus in two ways. First, we remove the restriction that $r < R$; we allow both of the radii r and R to be arbitrary positive numbers. Second, we consider replacing the middle Bézier site—which, for the true torus, is the zero site—with the mid-point of the surface patch, scaled by some multiplier m. For example, consider the patch in which Y and Z are both positive. The mid-point of that patch corresponds to the parameter values $\alpha = \beta = 90$ and has the coordinates $(X, Y, Z) = (0, R, r)$. We place the middle Bézier site of that patch at the site $(w; x, y, z) = (m; 0, mR, mr)$, for some real number m.

Note that choosing $r > R$ leads to a torus that intersects itself at a cone-like singular point on the Z axis. Note also that choosing any value of m different from 0 leads to a surface that is not a torus, even though the boundary curves of each patch don't move. In fact, the resulting surface doesn't even have tangent continuity.

Viewing the torus

Your torus will be illuminated by two lights placed in the scene. Please see the header file `/usr/class/cs348a/pp/pp1/pp1.h` for the position and characteristics that you should use for the light sources. The torus itself will be made up of a material whose characteristics are also defined in that header file. Please use this header file as part of your code.

You should open and paint two windows: a main window, where the torus itself is displayed, and a secondary smaller window, where the coordinate axes, the torus, and the viewpoint are all displayed. The latter window is there to help you in understanding how to move the viewpoint around the torus.

Interacting with the view

You need to implement two kinds of interactions. First of all, your users should be able to modify any of the free parameters r, R, and m defining the torus. They will do this by moving sliders corresponding to the free parameters. They should also be allowed to change the viewpoint by moving three additional sliders corresponding to the coordinates of the viewpoint in the natural (X, Y, Z) coordinate frame of the torus.

Sample program

In the directory `/usr/class/cs348a/pp/pp1` there is a sample program called `pp1.c` that implements part of the funcionality of the program that you need to write. In this sample program, a sphere is visualized instead of the torus. We suggest that you modify this code, including the information to create and visualize the torus. An interface defined using forms can be found at the file `pp1_ui.fd`. Just type `fdesign pp1_ui` to see what the interface looks like, and use `fdesign` to modify the interface as desired. The output of `fdesign` is a C program with appropriate calls to the forms library. The forms library can be found in the SGI machines at Sweet Hall in the directory `/usr/local/X/lib`.

OpenGL has a special way to tesselate NURBS, and in order to have reasonable performance, change the following properties of the NURB object you create in OpenGL in the following way:

```
gluNurbsProperty(theNurb,GLU_SAMPLING_METHOD, GLU_DOMAIN_DISTANCE);
gluNurbsProperty(theNurb,GLU_U_STEP,15);
gluNurbsProperty(theNurb,GLU_V_STEP,15);
```

These properties control how fine the tesselation will be. A performance penalty will be added if the step values are increased, and the default values tesselate the nurbs in great detail.

What to hand in

Remember that in programming assignment you are allowed to work in teams of up to three students, and that each team need hand in only a single write up. On or before the due date, you must hand in a paper write-up explaining your program and e-mail the TAs a copy of your final code. We will then set up a time in the graphics lab during which you will have to demo your program to the TAs.

Since in this assignment OpenGL subroutines will do the bulk of the hard work, concentrate your work on making your program as simple and as clear as possible.

Programming in OpenGL

OpenGL provides extensive support for 2-D and 3-D color graphics and animation. OpenGL routines are built as a logical layer above the X window system. Apart from being able to draw polygons and lines, you can also easily render more complicated graphical objects including B-Splines and NURBS. For instance, once you know the Bézier control sites of a rational B-spline, rendering the spline is just a matter of making a call to the appropriate OpenGL routine.

Detailed information about how to use the library is provided in the OpenGL Programming Guide. Lots of example OpenGL programs are provided in the directory `/usr/class/cs348a/pp/opengl`. A good way to learn OpenGL is to play around with these example programs. Also, in any SGI machine there are online versions of the OpenGL programming guide. Just type `insight`, which will lead you to the library of online books.