CAP 5705 – Homework 3

Curve Modeling

I. Introduction

The project designs a user interface so that the user can input several control points, and renders a cubic C1 Bezier curve from the control points the user have inputted. Each step of the project can be realized separately, by choosing from the main menu. The main menu is as below(Figure 1-1).

The environment of the homework is Windows7+Visual-Studio2010+Glut, the code and the makefile are included in the zip file. To make sure they can work well, I have tried to run it on computers in the lab.

```
[1] Blending Function
[2] De Casteljau
[3] OpenGL function
[4] Bazier surface
Please input your option:
1
Please select number of control points: 6/8? (6:2 segments 8:3 segments)
6
Do you want to input the control points or use the set ones:(y/n)
n
```

```
[1] Blending Function
[2] De Casteljau
[3] OpenGL function
[4] Bazier surface
Please input your option:
2
Please select number of control points: 6/8? (6:2 segments 8:3 segments)
6
Do you want to input the control points or use the set ones:(y/n)
y
input control points:
-0.8 0.5
-0.6 -0.6
-0.3 -0.6
0.2 0.3
0.4 0.3
0.6 -0.5
```

Figure 1-1

II. The Bezier Curve Drawing

Bezier curves are one of the most common representations for free-form curves in computer graphics. is a polynomial curve that approximates its control points. The curves can be a polynomial of any degree. Often, complex shapes are made by connecting a number of Bezier curves of low degree, cubic(d = 3) Bezier curves are commonly used for this purpose.

1. Blending Function

The basic idea of Bezier curves is as the equation (2-1), and the basis(blending) function is shown as equation (2-3).

$$(t+(1-t))^n = 1 \tag{2-1}$$

$$b_{k,n}(t) = \binom{n}{k}(1-t)^{n-k}t^k \tag{2-2}$$

In this way, the Blending function of the Cubic Bézier curves (four points) is as the following (equation 2-3) and (figure 2-1).

$$B(t) = (1-t)^3\boldsymbol{p}_0 + 3t(1-t)^2\boldsymbol{p}_1 + 3t^2(1-t)\boldsymbol{p}_2 + t^3\boldsymbol{p}_3 \tag{2-3}$$

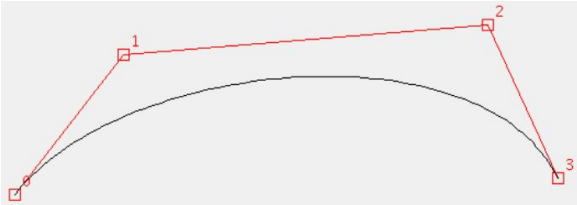


Figure 2-1

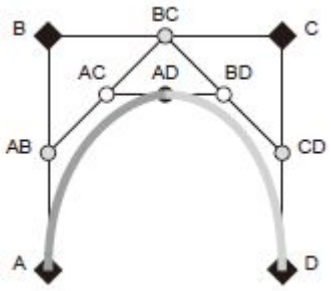


Figure 2-2

With the method above, the implementation of the blending functions is shown in figure 2-3.

2. De Casteljau Algorithm

The de Casteljau algorithm uses a sequence of linear interpolations to compute the positions along the Bezier curve of arbitrary order. It begins by connecting every adjacent set of points with lines, and finding the point on these lines that is the u interpolation, giving a set of n−1 points. These points are then connected with straight lines, those lines are interpolated (again by u), giving a set of n−2 points. This process is repeated until there is one point. An illustration of this process is shown in Figure 2-2. With the method above, the implementation of the blending functions is shown in figure 2-3.

3. The OpenGL API

OpenGL supports Bezier curves through evaluators that compute Bernstein polynomials for different values of the curve parameter. Here are the functions used by the project in the library and the meaning of the parameters.

The command glMap1() defines a one-dimensional Bezier evaluator that uses the Bernstein Polynomial equations:

1) void glMap1{fd}(GLenum target, TYPEu1, TYPEu2, GLint stride, GLint order, const TYPE*points);

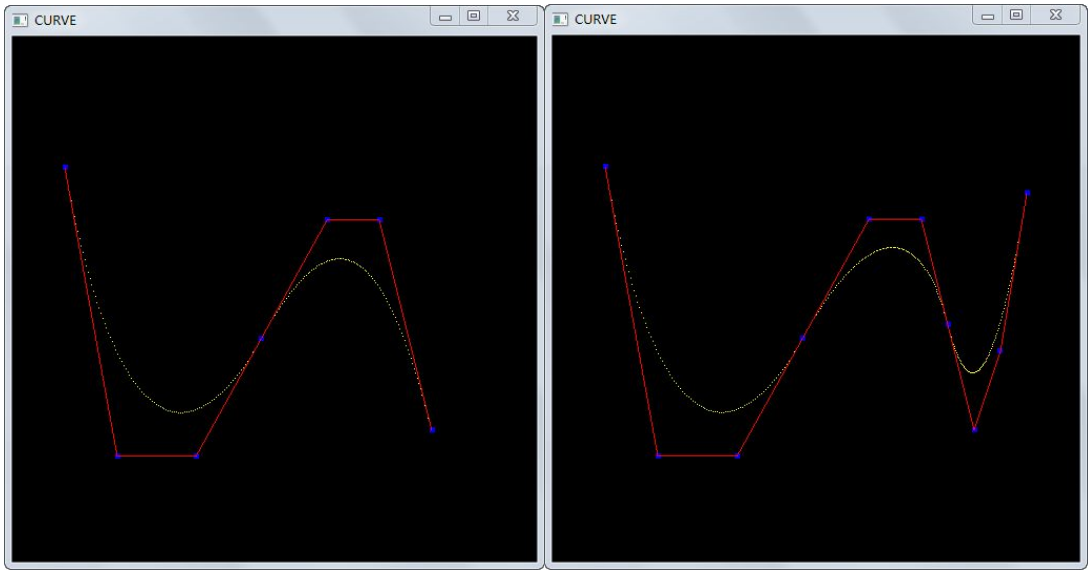
- The **target** parameter specifies what the control points represent, and therefore how many values need to be supplied in points.

- The second two parameters, **u1** and **u2**, indicate the range for the variable u.
- The variable **stride** is the number of single- or double-precision values (as appropriate) in each block of storage. Thus, it's an offset value between the beginning of one control point and the beginning of the next.
- The **order** is the degree plus one, and it should agree with the number of control points.
- The **points** parameter points to the first coordinate of the first control point. Using the example data structure for glMap1*() .

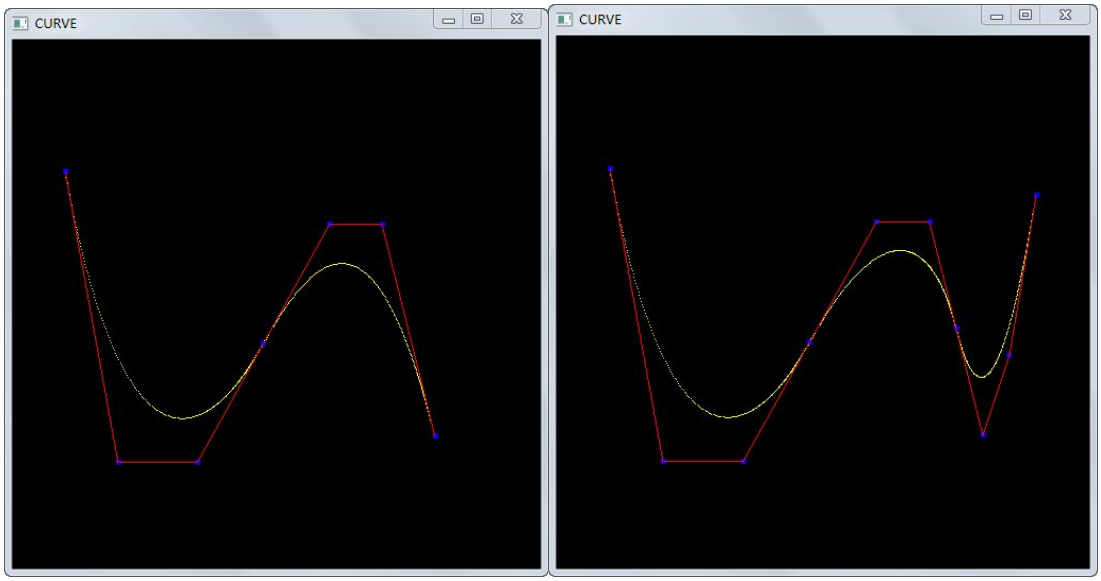
2) glEvalCoord1*() to evaluate a defined and enabled one-dimensional map:
void glEvalCoord1{fd}{v}(TYPE u);

- **u** is the value (or a pointer to the value, in the vector version of the command) that is the domain coordinate.

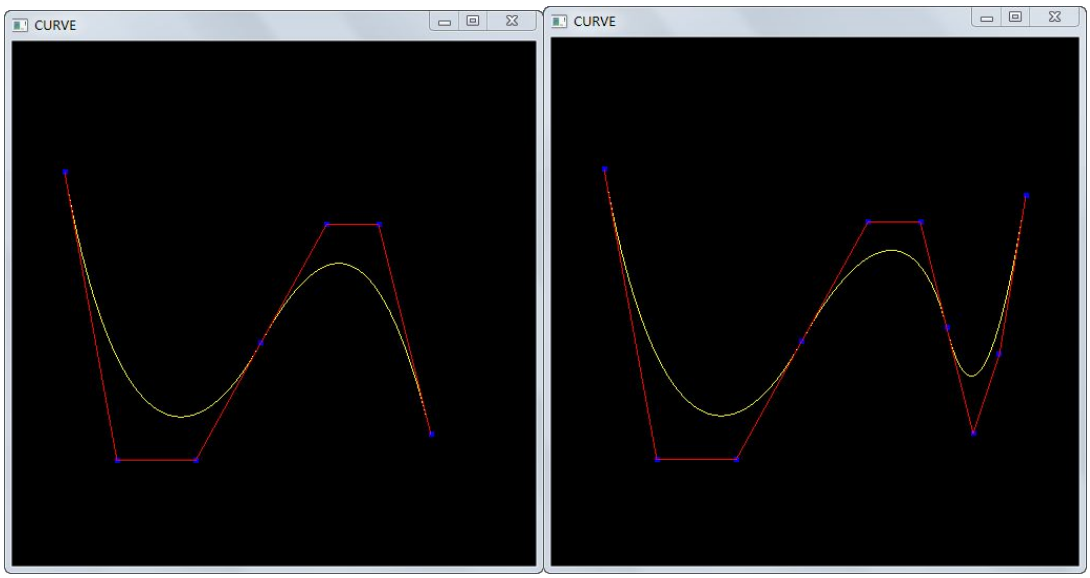
With the functions above, the result will be shown in figure 2-3.



Blending function with 6 (left) and 8 (right) control points



The de casteljau algorithm with 6 (left) and 8 (right) control points



The OpenGL API with 6 (left) and 8 (right) control points

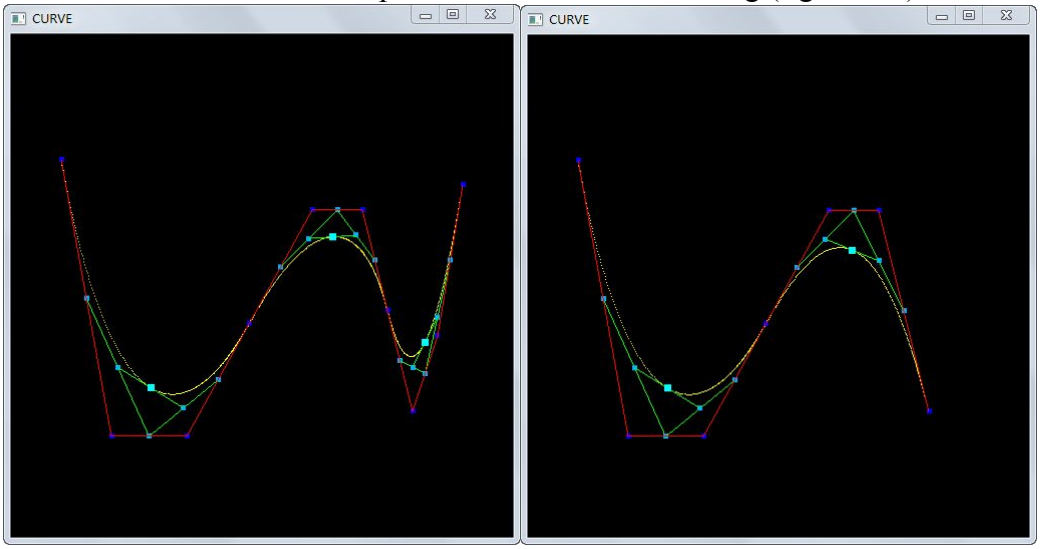
Figure 2-3

III. Subdivison

The meaning of subdividing a curve is to cut a given Bézier curve at C(u) for some u into two curve segments, each of which is still a Bézier curve.Moreover, since the original Bézier curve of degree n is cut into two pieces, each of which is a subset of the original degree n Bézier curve, the resulting Bézier curves must be of degree n. So the problem is:

- Given a set of n + 1 control points P0, P1, P2, ..., Pn and a parameter value u in the range of 0 and 1, we want to find two sets of n+1 control points Q0, Q1, Q2, ..., Qn and R0, R1, R2, ..., Rn such that the Bézier curve defined by Qi's (resp., Ri's) is the piece of the original Bézier curve on [0,u] (resp., [u,1]).

The results of the subdivison steps are shown as the following (figure 3-1)



The subdivision of 8(left) and 6 (right) control points

Figure 3-1

IV. Surface

Use glMap2*() and glEvalCoord2*() to define and then invoke a two-dimensional evaluator.

1) void glMap2{fd}(GLenum target, TYPEu1, TYPEu2, GLint ustride, GLint uorder, TYPEv1, TYPEv2, GLint vstride, GLint vorder, TYPE points);

- The **target** parameter is similar to the one in MAP1, except that the string MAP1 is replaced with MAP2.
- The values are also used with glEnable() to enable the corresponding evaluator.
- Minimum and maximum values for both u and v are provided as **u1, u2, v1,** and **v2.**
- The parameters **ustride** and **vstride** indicate the number of single- or double-precision values (as appropriate) between independent settings for these values, allowing users to select a subrectangle of control points out of a much larger array.

2) void glEvalCoord2{fd}(TYPE u, TYPE v);

- Causes evaluation of the enabled two-dimensional maps. The arguments **u** and **v** are the values (or a pointer to the values u and v, in the vector version of the command) for the domain coordinates. If either of the vertex evaluators is enabled (GL_MAP2_VERTEX_3 or GL_MAP2_VERTEX_4), then the normal to the surface is computed analytically.

The result of the surface is as the following(figure 4-1).

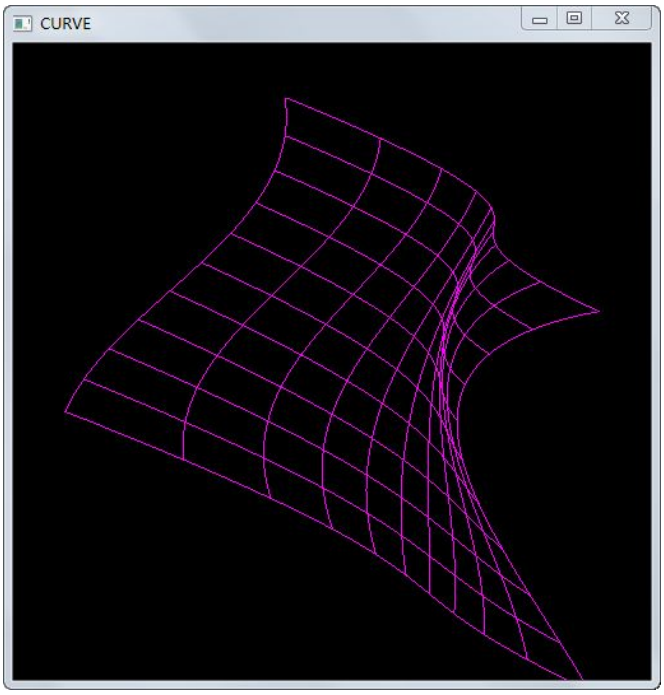


Figure 4-1

V. Result Analysis

1) Advantages and Disadvantages

- The advantage of using polynomial blending functions is that the blending curve is polynomial if the curves segments k1 and k2 are so. A disadvantage of the uniquely defined blending functions is the lack of design parameters.
- The de Casteljau's algorithm is slower than directly using the parametric formula, but it is numerically more stable to implement it for rendering (cubic) Bezier curves. But it looks smoother than the blending function in the homework.
- Taking advantage of OpenGL's existing mechanisms for texturing, programmability, and per-fragment operations can make it easier to enumerate points on surface. So it’s much faster and smoother by using the APIs in OpenGL, and it looks smoother than the de casteljau’s algorithm.

2) What happens when a control point is repeated?

- Adding multiple control points at a single position in space will add more weight to that point "pulling" the Bézier curve towards it.