Assignment 2:

NAME: XIAOHAN WU
STUDENT NUMBER: 2019533093
EMAIL: WUXH@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In HW2, we are required to generate Bézier surfaces, and other surfaces such as B-Spline/NURBS surfaces using openGL programming. The program contains 4 basic parts and 2 optional parts. The first four parts are: 1.Implementing the basic iterative de Casteljau Bézier vertex evaluation algorithm. 2.Constructing Bézier surfaces with the normal evaluation at each mesh vertex. 3.Rendering the Bézier surfaces based on the vertex array. 4.Creating more complex meshes by stitching multiple Bézier surface patches together. For the last two parts,we are required to construct the B-Spline surfaces and support the interactive editing (by selection) of control points.

2 IMPLEMENTATION DETAILS

In this section, I will introduce how I implement my program for each part respectively.

1.The realization of de Casteljau Bézier vertex evaluation algorithm is relatively easy after giving the notes, shown as below.The return value of such function is the position of the point that we are going to calculate.

```
Input: array P[0:n] of n+1 points and real number u in [0,1]
Output: point on curve, C(u)
Working: point array Q[0:n]

for i := 0 to n do
      Q[i] := P[i]; // save input
for k := 1 to n do
      for i := 0 to n - k do
            Q[i] := (1 - u)Q[i] + u Q[i + 1];
return Q[0];
```

Fig. 1. De Casteljau Bézier vertex evaluation algorithm

2.After setting the position of our control points to the surface,we start to evaluate the position and normal of the points sampled on the Bézier surface.Given any (u,v) ranging from [0,1]*[0,1], we can use the algorithm demonstrated in Fig.2 to get the position of its corresponding point on the surface.

To calculate the normal of the point,we have to calculate the tangent vector of the point along u and v direction first, and then calculate the cross product of these two vectors .As it is shown in Fig.3, in order to calculate the tagent, we have to note down the last two points which form the point we want, and minus their position.

```
Input: a m+1 rows and n+1 columns of control points and (u,v).
Output: point on surface p(u,v)
Algorithm:

    for i := 0 to m do
        begin
            Apply de Casteljau's algorithm to the i-th row of control points with v;
            Let the point obtained be q_i(v);
        end
    Apply de Casteljau's algorithm to q_0(v), q_1(v), ..., q_m(v) with u;
    The point obtained is p(u,v);
```

Fig. 2. Position calculation of points on the Bézier surface

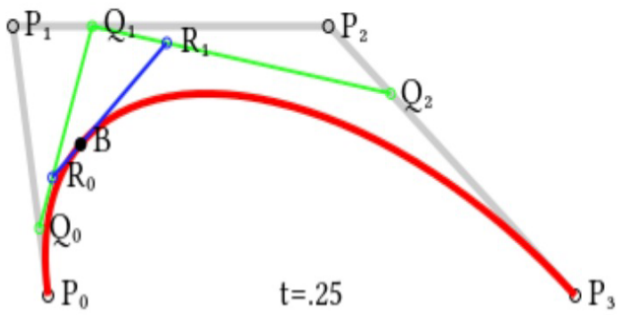


Fig. 3. Tangent vector calculation

3.After finishing the procedure of how to calculate all the needed information for any (u,v) point, we evenly divide [0,1]*[0,1] into 64*64 grids,and each grid vertex is expressed as (u,v). Then we calculate all of those grid vertices and store their content into a vertex array, and set up the indices array which stores the sequence of every three points to construct the meshes.

4.In the stitching part, I stitch two Bézier surface patches together, maintaining 1st order continuity(i.e.not only the points are connected, but also their tangents are continuous at the joint.) To ensure this, we need to make sure that the control polygon are colinear at the last control point of the first surface and the first control point of the second surface. This can be achieved by setting the positions of the control points manually.

5.To construct a B-spline surface, we first have to evaluate the position of a point on the B-spline curve given the control points,named as De Boor's algorithm. The algorithm is shown in Fig.4

Then it's relatively easy to find the position of a point on the B-spline surface based on De Boor's algorithm.The whole procedure is shown in Fig.5,and to tell the truth, the procedure is nearly as same

Input: a value u
Output: the point on the curve, $\mathbf{C}(u)$

If u lies in $[u_k, u_{k+1})$ and u != u_k let $h = p$ (i.e., inserting u p times) and $s = 0$;
If $u = u_k$ and u_k is a knot of multiplicity s, let $h = p - s$ (i.e., inserting u $p - s$ times);
Copy the affected control points $\mathbf{P}_{k-s}$, $\mathbf{P}_{k-s-1}$, $\mathbf{P}_{k-s-2}$, ..., $\mathbf{P}_{k-p+1}$ and $\mathbf{P}_{k-p}$ to a new array and rename them as $\mathbf{P}_{k-s,0}$, $\mathbf{P}_{k-s-1,0}$, $\mathbf{P}_{k-s-2,0}$, ..., $\mathbf{P}_{k-p+1,0}$

for r:= 1 **to** h **do**
  for i:= $k-p+r$ **to** $k-s$ **do**
    begin
      Let $a_{i,r} = (u - u_i) / (u_{i+p-r+1} - u_i)$
      Let $\mathbf{P}_{i,r} = (1 - a_{i,r})\,\mathbf{P}_{i-1,r-1} + a_{i,r}\,\mathbf{P}_{i,r-1}$
    end
$\mathbf{P}_{k-s,p-s}$ is the point $\mathbf{C}(u)$.

Fig. 4. De Boor's algorithm

as what we have done in constructing the Bézier surface.Therefore, the detailed information such as how to sample the points and construct the mesh is ommited in this step.

Input: a value u
Output: the point on the curve, $\mathbf{C}(u)$

If u lies in $[u_k, u_{k+1})$ and u != u_k let $h = p$ (i.e., inserting u p times) and $s = 0$;
If $u = u_k$ and u_k is a knot of multiplicity s, let $h = p - s$ (i.e., inserting u $p - s$ times);
Copy the affected control points $\mathbf{P}_{k-s}$, $\mathbf{P}_{k-s-1}$, $\mathbf{P}_{k-s-2}$, ..., $\mathbf{P}_{k-p+1}$ and $\mathbf{P}_{k-p}$ to a new array and rename them as $\mathbf{P}_{k-s,0}$, $\mathbf{P}_{k-s-1,0}$, $\mathbf{P}_{k-s-2,0}$, ..., $\mathbf{P}_{k-p+1,0}$

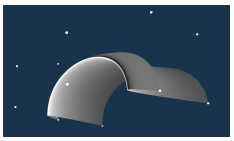
for r:= 1 **to** h **do**
  for i:= $k-p+r$ **to** $k-s$ **do**
    begin
      Let $a_{i,r} = (u - u_i) / (u_{i+p-r+1} - u_i)$
      Let $\mathbf{P}_{i,r} = (1 - a_{i,r})\,\mathbf{P}_{i-1,r-1} + a_{i,r}\,\mathbf{P}_{i,r-1}$
    end
$\mathbf{P}_{k-s,p-s}$ is the point $\mathbf{C}(u)$.

Fig. 5. Position calculation of points on the B-Spline surface

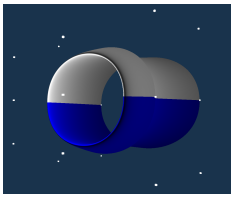
6.To support the interactive editing (by selection) of control points,my method is that we should first store the position of current cursor,also the projection,view matrix, and then use the function glm::unproject() to backproject the screen position to 3d dimension.After that,compare this position to all control points' position so that we can find the nearest control point. Therefore, we can update the position of that control point to wherever we drag it in the next continuous frames. Also, after updation, we have to update the VAO,VBO of the surface in order to generate the updated surface.

3 RESULTS

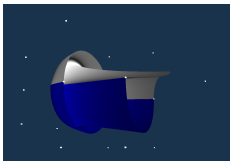
Below are the results of my programming work.Figure(a) to Figure(d) represent 'Single Bézier surface', 'Stitching Bézier surfaces', 'Bézier surface with moved controlpoints' and 'Single B-Spline surface' respectively.What's worth mentioning is that for Figure(d),my principle of selecting control points is to select 5*5 evenly distributed points on the planar surface, and move away the middle one of those points. The reason why I choose my control points in this way is that it can easily demonstrate the correctness of my B-spline surface realization,(also for the demonstration that my algorithm does not stuck into the poor condition that the surface converages into the zero point.)



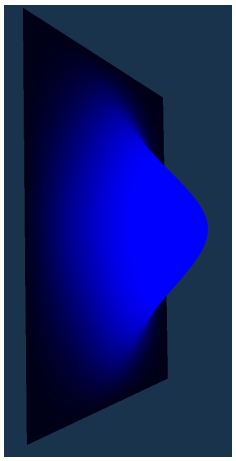
(a) Single Bézier surface



(b) Stitching Bézier surfaces



(c) Bézier surface with moved controlpoints



(d) Single B-Spline surface