Assignment 2
Geometric Modeling

NAME: LIU YIFEI
STUDENT NUMBER: 2020533131
EMAIL: LIUYF7@SHANGHAITECH.EDU.CN

Contents

1 INTRODUCTION

In this assignment, the following features have been achieved.

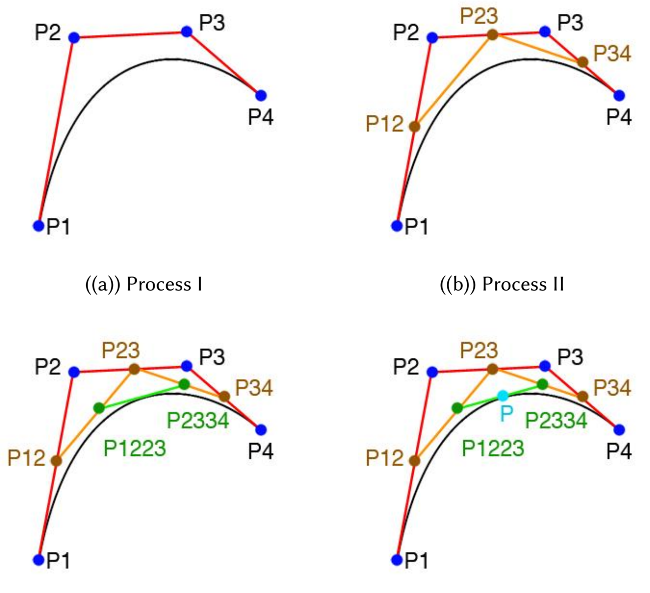
- Implementation of the basic iterative de Casteljau Bézier vertex evaluation algorithm.
- Construction of Bézier Surface using Bézier curve.
- Cut the Bézier surface with a z-plane.

2 IMPLEMENTATION DETAILS

2.1 Bézier curve

As we all know, the de Casteljau Bézier vertex evaluation algorithm is a computer-friendly method to evaluate a Bézier curve. It does interpolations between two points in sequence and does exactly the same thing to the new-generated points, until there is only one point. By the way, the tangent vector is the same as the direction of connecting the last two points.
The process is shown as in Fig.1. The algorithm itself is straightforward, thus the code needs no more extra explaining.

```
Vertex BezierCurve::evaluate(std::vector<vec3>& control_points, float t) {
  std::vector<vec3> nowControlPoints = control_points;
  Vertex result;
  while (nowControlPoints.size() > 1) {
    std::vector<vec3> nextControlPoints;
    for (int i = 0; i < nowControlPoints.size() - 1; i++) {
      vec3 nextPoint = (1 - t) * nowControlPoints[i] + t * nowControlPoints[i + 1];
      nextControlPoints.push_back(nextPoint);
    }
    if (nextControlPoints.size() == 2)
    {
      result.normal = normalize(nextControlPoints[1] - nextControlPoints[0]);
    }
    nowControlPoints = nextControlPoints;
  }
  result.position = nowControlPoints[0];
  return result;
}
```

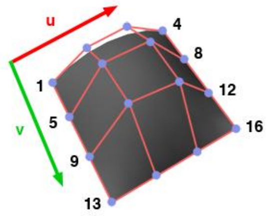


((a)) Process I ((b)) Process II
((c)) Process III ((d)) Process IV

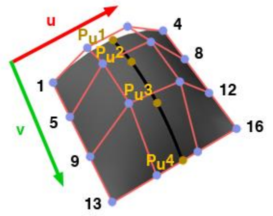
Fig. 1. Evaluation Process of Bézier curve

2.2 Bézier surface

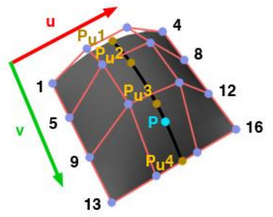
As long as the Bézier curve can be constructed properly, the Bézier surface is piece of cake. We can use the Bézier curve to rapidly build a Bézier surface by firstly evaluate in one direction and then in the other direction.
The only thing that needs attention is that in order to obtain the normal vector, we have to evaluate in direction order u->v and v->u separately, resulting in the tangent vector in the direction of v and u. And next cross these two tangent vectors can we get the normal vector (normalize is optional).



((a)) Process I



((b)) Process II



((c)) Process III

Fig. 2. Evaluation Process of Bézier surface

```
Vertex BezierSurface::evaluate(float u, float
    v) {
  std::vector<vec3> controlPoints;

  //First in direction u
  for (size_t i = 0; i < 4; i++)
  {
    BezierCurve bc(control_points_m_[i]);
    controlPoints.push_back(bc.evaluate(u).
        position);
  }
  //next in direction v using these 4 points
      in u direciton
  BezierCurve bc(controlPoints);
  Vertex dv = bc.evaluate(v);

  //do again in the reverse order
  controlPoints.clear();
  for (size_t i = 0; i < 4; i++)
  {
    BezierCurve bc(control_points_n_[i]);
    controlPoints.push_back(bc.evaluate(v).
        position);
  }
  BezierCurve bc2(controlPoints);
  Vertex du = bc2.evaluate(u);
  Vertex result;
  result.position = du.position;
  result.normal = cross(du.normal, dv.normal)
      ;
  return result;

}
```

2.3 Interactive editing of control points

Interactive editing is a challenge work since it needs collaboration among frontend UI, backend computation and shader program on GPU.

2.3.1 UI.

The first step is to plot the control points. I define an Object aiming to this propose.

```
Object controlPoints;
controlPoints.draw_mode.primitive_mode =
    GL_POINTS;
for each (BezierSurface bs in vertexs)
    for each (auto points in bs.
        control_points_m_)
        for each (auto point in points)
            controlPoints.vertices.push_back(
                Vertex{ point,point });
testobj.init();
```

Change the default point size which was one pixel to ten pixels for better visualization by calling glPointSize(10.f); Beside, I design two styles representing if one is being selected by the user. The final result is shown as in Fig.3

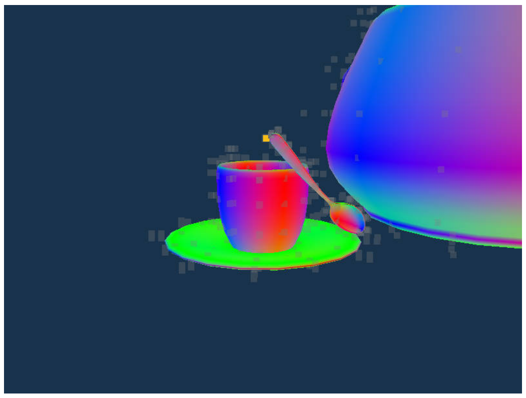


Fig. 3. Point Selection

The main program maintain one variable as the vertex selected, and transfer it to the fragment shader. In the FS, color the vertex depends whether the vertex is selected.

```
#version 330 core
out vec4 color;
in vec3 normalV;

uniform vec3 selection;

void main(){
  if(normalV==selection)
  {
  color = vec4(.968f,.745f,0.094f,1.f);

  }
  else
  {
  color = vec4(.5f,.5f,.5f,0.3f);
  }
}
```

2.3.2 *Reconstruct the Bezier surface.*

Whenever the control points are changed, the Bezier surface requires immediate reconstruction. Synchronization with selection is also need attention. The principle is straightforward but the code is a little tricky because it interrupts the original program pipeline. Details are too complicated to be discussed here.

2.4 Cut Bézier surface

Generating a fine-resolution cut Bézier surface is not a easy work, however there are a few methods to approximate its effect.

2.4.1 *Sample on Plane.* .

After triangulation in u,v parameter space, check the reletionship between triangles and the plane. I define three types of triangle, as shown in Fig.5.

First define the rainbow direction of the plane is "inside". If all three vertex of the triangle are inside, like the blue triangle, just do nothing about it. If all three vertex of the triangle are outside, like the red one, just drop the triangle. The other situation is just like the yellow ones, whose vertex are inside and outside.

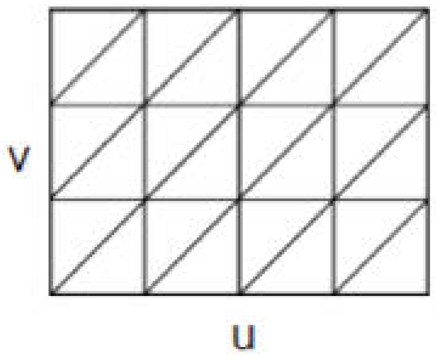


Fig. 4. triangulation

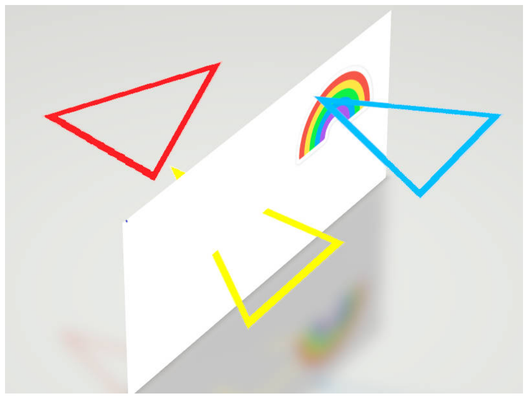


Fig. 5. triangle type

To "fix" those triangles, do interpolation on the plane, generate two new points on the plane and on the edges crossing the plane, as the green points in Fig.6. With these four vertex, the original triangle is converted to four new triangles inside the plane. Process all triangles on the mesh with this method, and no other modification is needed.

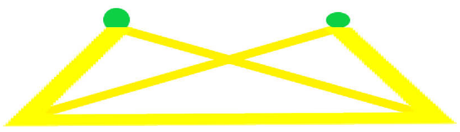


Fig. 6. interpolation on the plane

3 RESULTS

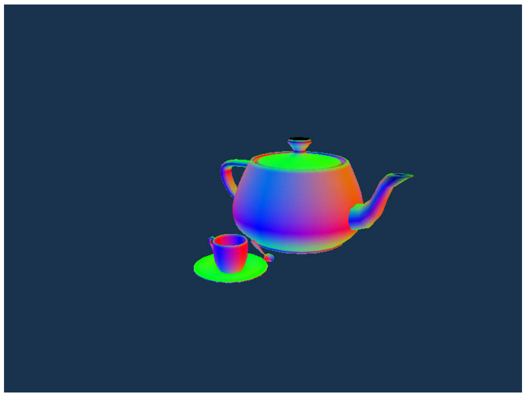


Fig. 7. Shader a model using Bézier.

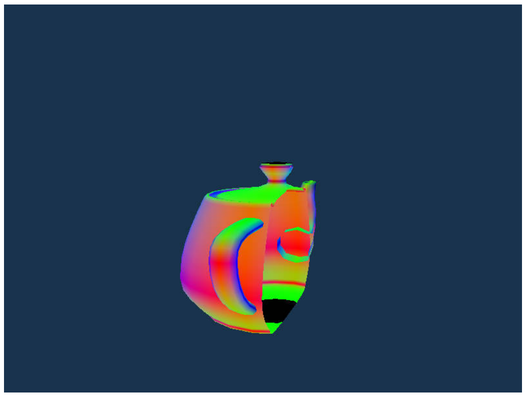


Fig. 8. Cut Bézier surface with a plane.

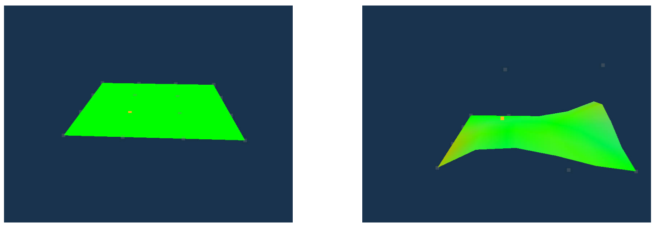

((a)) Before Editing ((b)) After Editing

Fig. 9. Interactive editing (by selection) of control points