Assignment 2 : Geometric Modeling

NAME: HIDDEN
STUDENT NUMBER: HIDDEN
EMAIL: HIDDEN@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, the following things are done:

(1) Camera control and Phong shading (same as the last assignment)
(2) Bezier curve and bezier surface
    - Evalauate the position of any point on a Bezier curve
    - Construct a bezier surface from a set of control points
    - Read in `.bzs` files which contains a bunch of control points and render the corresponding multiple bezier surfaces
(3) B-spline curve and B-spline surface
    - Construct a B-spline curve
    - Construct a B-spline surface from a set of control points, and render it.
(4) Adaptive mesh generation
    - In parameter space, adaptively divide the line into serveral segments, and then construct a Bezier curve from the control points.
    - Read in `.bzs` files and render the bezier surfaces with adaptive mesh.

2 IMPLEMENTATION DETAILS

2.1 Camera Control and Phong lighting model

The camera control is the same as the last assignment, except that polygon mode control is added. Basically:

- Use `WASD` to move the camera upward, left, downward and right.
- Use `RF` to move the camera forward and backward.
- Use mouse to move camera location.
- Use `O` to apply `glPolygonMode GL_FILL` to render the filled polygon.
- Use `P` to apply `glPolygonMode GL_LINE` to render the wireframe, which can help us take a look at the mesh structure precisely.

The Phong lighting model implementation and light settings are same as the last assignment, so we will not discuss it here.

2.2 Bezier Curve

The Bezier curve is defined by the following equation:

$$P(t) = \sum_{i=0}^{n} P_i B_{n,i}(t) \tag{1}$$

where $B_{n,i}(t)$ is the Bernstein polynomial of degree n and index i

To achieve higher precision and efficiency, we will use de Casteljau's algorithm to calculate this. The algorithm is as follows:

Suppose there are $n + 1$ control points with a real number t (t is between 0 and 1), then

Algorithm 1 De Casteljau's algorithm

```
1: procedure DeCasteljau($P_0, P_1, \cdots, P_n, t$)
2:     $Q_0 = P_0$
3:     for $i = 1$ to $n$ do
4:         $Q_i = (1 - t)Q_{i-1} + tP_i$
5:     end for
6:     return $Q_n$
7: end procedure
```

The tangent vector of each point is also calculated in the above process.

Therefore given a set of control points, we increase t gradually from 0 to 1 with step `float DIM_RECIPROCAL = 0.02` and store each result point in `std::vector<vec3>`, and we can get a approximated continous curve.

2.3 Bezier Surface

Now that we know how to evaluate a single-dimensional Bezier curve. To evaluate a 2-dimensional Bezier surface, we simply calculate the tensor product of the two single-dimensional curves.

In detail, the evaluation can be divided into the following steps:

(1) Evaluate the position of each point on each set of control points along `u` direction.
(2) Utilize the evaluated points as a new set of control points, and generate the final point along `v` direction. Note that the tangent along `v` direction is simutaneously calculated.
(3) Swap `u` and `v` and repeat the above two steps again. This time we get the same position and the tangent along `u` direction. The normal vector is the cross product of the two orthogonal tangent vectors.

After the above process, we have completed the generation of a mesh grid. We know each points' position and normal. In order to render it, we simply divide each quadrilateral into two triangles. Then we bind then to `VAO`, `VBO` and `EBO` and apply Phong lighting model similar to what we did in assignment 1.

2.4 Multiple bezier surface and stitching

In order to render multiple bezier surfaces, we need to be sure it satisfies continuity.

Since the provided `.bzs` files are all already continous, we can simply process them one by one and load them together.

student number: hidden
email: hidden@shanghaitech.edu.cn

2.5 B-spline curve

Given $n+1$ control points $\boldsymbol{P}_0, \boldsymbol{P}_1, \cdots, \boldsymbol{P}_n$ and a knot vector $\boldsymbol{K} = (k_0, k_1, \cdots, k_{n+m+1})$, the B-spline curve of degree p at point u is defined by the following equation:

$$\boldsymbol{P}(u) = \sum_{i=0}^{n} \boldsymbol{P}_i N_{i,p}(u) \tag{2}$$

where n, m, p must satisfy $m = n + p + 1$.

The basis function $N_{i,p}(u)$ is defined by the Cox-de Boor recursion formula:

$$N_{i,0}(u) = \begin{cases} 1, & k_i \le u < k_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$N_{i,p}(u) = \frac{u - k_i}{k_{i+p} - k_i} N_{i,p-1}(u) + \frac{k_{i+p+1} - u}{k_{i+p+1} - k_{i+1}} N_{i+1,p-1}(u) \tag{4}$$

Since $N_{i,p}$ is non-zero only when $k_i \le u < k_{i+p+1}$, at most $p+1$ degree-p-based functions are nonzero. Therefore, the B-spline curve can be approximated by the following algorithm:

$$Q_i = (1 - a_i)P_{i-1} + a_i P_i$$

where a_i is

$$a_i = \frac{u - k_i}{k_{i+p+1} - k_i}$$

for $k - p + 1 \le i \le k$

The Q_i is the new control point calculated by P_{i-1} and P_i.

If a knot u is inserted repeatedly until its multiplicity is p, the last generated new control point is exactly the result $P(u)$. Therefore if u lies in the interval $[k_i, k_{i+1})$, the new control point Q_i is the result $P(u)$.

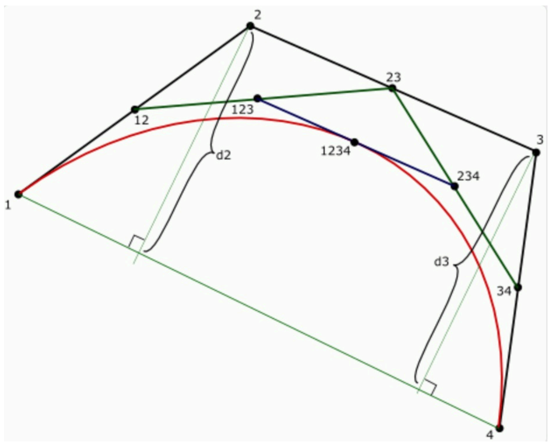
2.6 B-spline surface

The construction of B-spline surface is almost the same as the Bezier surface. The only difference is that we need to calculate the B-spline basis function instead of the Bernstein polynomial. Therefore we won't go into details here. Refer to subsection 2.2 (Bezier surface) for detailed explanations.

2.7 Adaptive mesh generation

In the previous sections, we have already known how to generate a mesh grid. However, the sample points along u and v directions have fixed density, which is defined by `float DIM_RECIPROCAL = 0.02` in the implementation. There might be too many unnecessary points on an almost straight curve, or too few points on a winding curve. We can solve this problem by generating an adaptive mesh, letting the number of sample points varying according to the curvature of the curve.

First let's focus on the Bezier curve. We use divide and conquer to divide the curve recursively until the curvature of the curve is small enough. To demonstrate the algorithm, we take a look at the picture below (the picture is from sourceforge):



Suppose there are 4 control points, namely p1, p2, p3 and p4. We first calculate the middle point p12, p23 and p34. Similar for p123 and p234, and finally p1234.

We will check if the distance between p1234 and the line segment between p1 and p4 is small enough. If it is, we will simply add p1234 to the result vector, because the curve is straight enough so that we can use a straight line to simulate it. Otherwise, we will divide the curve into two parts and recursively call the function, for densier division.

The critical part is that, how do we check the 'distance' is small enough or the segment is flat enough. To do that, we first calculate distance from p2 to the line p1p4, namely d2 as shown in the picture. Similar for the distance between d3 and p1p4 called d3. Then we check whether

$$d_2 + d_3 \le \epsilon \cdot d_{1,4}$$

to determine the flatness. The ϵ is a tolerance paramter, that is, higher value means higher tolerance and fewer points will be generated, and vice versa.

Then, given 4 control points, we use the following code to adaptively divide the line:

Algorithm 2 Recursive subdivision

```
1:  procedure DIVISION($P_1, P_2, P_3, P_4$) return sample points
2:      if recursion level is too deep then
3:          return Points
4:      end if
5:      calculate mid points $p12, p23, p34, p123, p234, p1234$
6:      calculate distance $d2$ and $d3$
7:      if $d2 + d3 \le tolerance * d14$ then
8:          Add the midpoint $p1234$ to Points
9:          return Points
10:     end if
11:     Division($P_1, P_{12}, P_{123}, P_{1234}$)
12:     Division($P_{1234}, P_{234}, P_{34}, P_4$)
13: end procedure
```

After that we can use the procedure to help us generate points for a Bezier surface. Suppose we have a 4×4 control point matrix, then the adaptive mesh generation is done with the following steps:

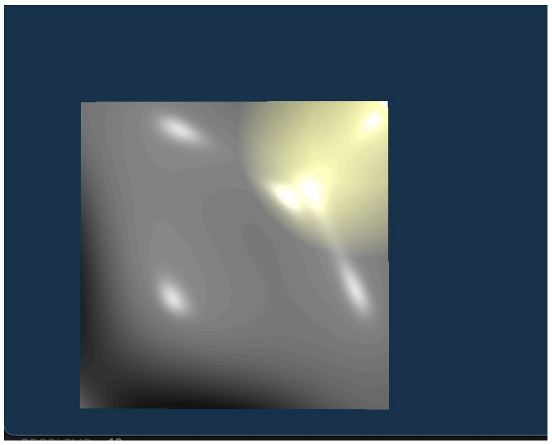
(1) Apply the above division process to each set of control points along `u` direction and store the generated sample points.
(2) Divide [0,1] into ten segments. For each segment, taking [0.2, 0.3] as an example, apply De Casteljau's algorithm to get the position, and check how many points lies in the corresponding interval.
(3) Repeat the above step for each row of adaptive sample points and take the max number of points `n`. Then divide the interval [0.2, 0.3] into `n` small segments.
(4) For `v` direction, apply similar process and obtain the points on `v` direction.
(5) Similar as the Bezier surface generation, generate the triangles and bind `VAO, VBO, EBO`.

In this way, we can avoid the complex trianglization, since what we have generated is also a quadrangle-shaped mesh, and the object generation is almost the same.

3 RESULTS

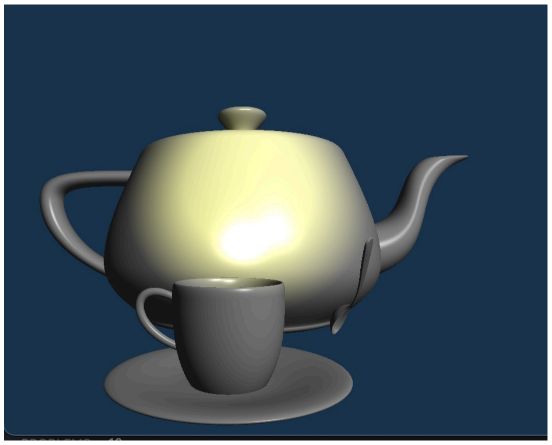
3.1 Bezier surface (uniform single bezier surface)

Below is a single bezier surface with specified 4*4 control points matrix.



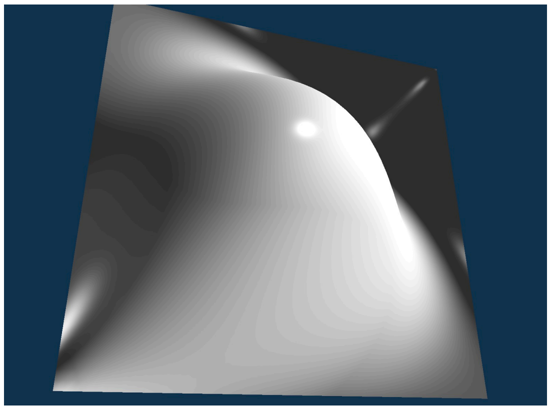
3.2 Multiple bezier surface (tea set)

Below is the rendered result of `assets/tea.bzs` file. The file contains multiple continous bezier surfaces' control points, which forms the tea set.



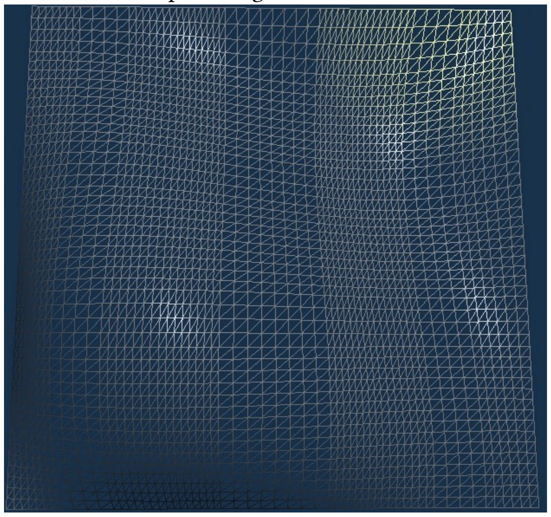
student number: hidden
email: hidden@shanghaitech.edu.cn

3.3 B-spline surface (uniform single b-spline surface)

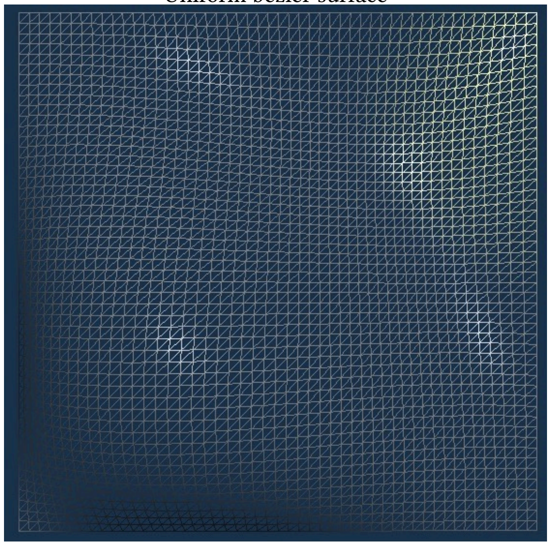


3.4 Adaptive Mesh (adaptive single bezier surface)

Adaptive single bezier surface



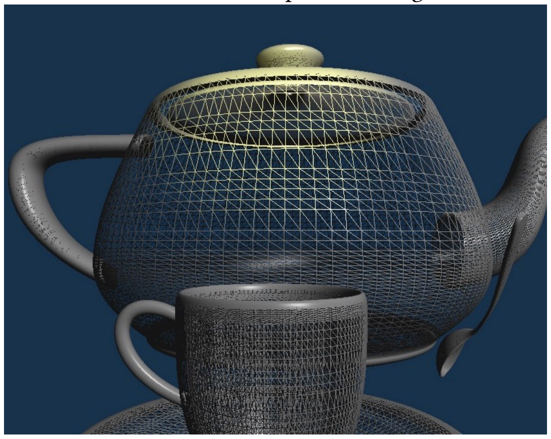
Uniform bezier surface



3.5 Adaptive Mesh (adaptive multiple bezier surface)

From the picture shown below, with adaptive mesh generation, the mesh is denser on the winding part (e.g. the front tea cup) and sparser on the straight part (the rear larger tea set), while without adaptive meshing, the mesh has the same density everywhere.

Tea set with adaptive meshing



Tea set without adaptive meshing (uniform meshing)