Assignment 2 : Geometric Modeling

NAME: NING BI
STUDENT NUMBER: 8574 5238
EMAIL: BINING@SHANGHAITECH.EDU.CN

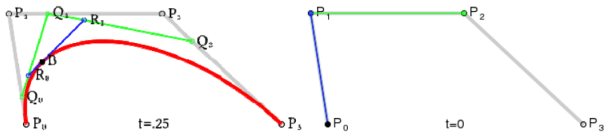

Fig. 1. Bezier Curve with 4 control points

1 INTRODUCTION

Basically, in this assignment we are required to implement Bezier Surface construction method with a particular triangulation algorithm to draw complex shapes. And in the meanwhile, texturing the mesh.
The task I have completed are as follow:

(1) This work can create **any size of control points** with inputs uc and vc.
(2) By using dynamic arrays, it can also create **any size of Bezier Surface** with inputs u, v and triangulated mesh.
(3) Texturing Bezier mesh with .jpg or .png images.
(4) Rendering the mesh with **Phong shading** and I also implement **depth test** and **anti-aliasing**.
(5) Using Shader for texturing, mashing, view changing and lighting.

2 IMPLEMENTATION DETAILS

Since the assignment itself has already gave us a detail descriptions of whole process, I just briefly describe my method and give some image result.

2.1 Bzier curve construction

In Fig.1 is a Bezier curve with 4 control points. However, with more control points, instead of implementing in a repeated way, we compute Bezier curve points in a recursive way as shown in Fig.2. Using this function we can compute the point p with step t under $numofcp$ control points.Moreover, we can obtain the $tangent$ of this p in the last loop as well.

```
// decasteljau function
glm::vec3 decasteljau(glm::vec3* ctrP, float t, int numofcp) {
    glm::vec3 *cp = new glm::vec3[numofcp];
    for (size_t i = 0; i < numofcp; i++)
    {
        cp[i] = ctrP[i];
    }
    for (size_t j = 0; j < numofcp - 1; j++)
    {
        for (size_t i = 0; i < numofcp - j - 1; i++)
        {
            cp[i] = (1 - t)*cp[i] + t * cp[i + 1];
        }
    }
    return(cp[0]);
}
```

Fig. 2. Decasteljau function

2.2 Bezier surface construction

To construct a Bezier surface, we firstly compute Bezier curves (each group of u points) along u direction. Then we compute another v groups of points (each group forming a Bezier curve) along the v direction. After these 2 steps we can obtain a $u * v$ array containing the Bezier surface points for triangulation.The codes are shown in Fig.3.

```
// bezier surface
glm::vec3* evaluateBezierSurface(glm::vec3* ctrlPoints, int u, int v, int uc, int vc) {
    // u cols of points
    // v rows of points
    // uc cols of control points
    // vc rows of control points
    glm::vec3* out = new glm::vec3[u*v];
    glm::vec3** pv = new glm::vec3*[u];
    glm::vec3** pu = new glm::vec3*[vc];
    // compute u control points along u direction
    for (int i = 0; i < vc; i++)
    {
        glm::vec3 *ctrp = new glm::vec3[uc];
        for (int j = 0; j < uc; j++)
        {
            ctrp[j] = ctrlPoints[i * uc + j];
        };
        pu[i] = evalBezierCurve(ctrp, u, uc);
    };
    for (size_t k = 0; k < u; k++)
    {
        glm::vec3 *cpv = new glm::vec3[vc];
        for (size_t j = 0; j < vc; j++)
        {
            cpv[j] = pu[j][k];
        };
        pv[k] = evalBezierCurve(cpv, v, vc);
    }
    for (size_t row = 0; row < v; row++)
    {
        for (size_t col = 0; col < u; col++)
        {
            out[u*row + col] = pv[col][row];
        }
    }
    return out;
}
```

Fig. 3. the function used to define Bezier surface

2.3 Normal Calculation

To give each points a normal, we firstly calculate **tangent** of each point in **u and v 2 directions**. Then we compute **cross product** of 2 tangent to obtain normals. Also, remember **normalizing** the normals. Fig.4 well illustrated this computation.

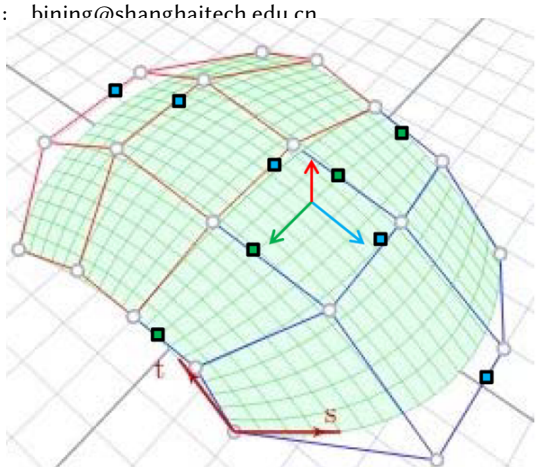

Fig. 4. Normal of each vertex

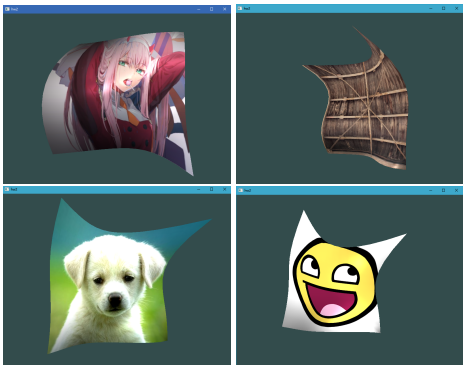

Fig. 5. Some testing results

2.4 Texturing

I implement texturing part in shader. It was quite a easy one, just bound the texture with corresponding coordinates on the mesh. The coordinates of textures should also be **normalized**. See **camera.vs and camera.fs** for more details.

2.5 Lighting

In this part, I applied **Phong shading** on the mesh. The implementation was just as same as last time, so I do not explain more here. See **camera.vs and camera.fs** for more details.

3 SOME RESULTS

Some testing results are shown as follows with more than 5 control points, texturing and Phong lighting.