Assignment 2 : Geometric Modeling

NAME: ZHAO DONGHAO
STUDENT NUMBER: 20096438
EMAIL: ZHAODH@SHANGHAITECH.EDU.CN

1 INTRODUCTION

This assignment is about how geometries are constructed with Bzier curves and surfaces, and how meshes are constructed by triangulation.
I construct triangle meshes based on Bzier surfaces, where several control points should be first specified to form a control mesh inculding 5x5 control mesh.
I render the constructed surface with lighting (Gourand or Phong shading), and texture the surface with a provided or your own generated image.
I use keyboard or mouse to view the surfaces at different positions.

2 IMPLEMENTATION DETAILS

For computing the position on the surface of Bzier surface for a pair of (u, v) values. First we compute the fout control points of an auxillary curve along the v direction using the u parameters. We then find P along this curve using the v parameter. For example, computing 4 control points along u direction.
Here is the code.

```
glm::vec3 mlBezier::mlEvalBezierPatch
(const glm::vec3 * controlPoints,
    const float & u, const float & v)
{
    glm::vec3 uCurve[5];
    for (int i = 0; i < 5; i++) {
        glm::vec3 curveP[5];
        curveP[0] = controlPoints[i * 5];
        curveP[1] = controlPoints[i * 5 + 1];
        curveP[2] = controlPoints[i * 5 + 2];
        curveP[3] = controlPoints[i * 5 + 3];
        curveP[4] = controlPoints[i * 5 + 4];
        uCurve[i] = mlEvalBezierCurve(curveP, u);
    }
    return mlEvalBezierCurve(uCurve, v);
}
```

Finally, before the render starts, we need to convert each one of these Bzier patches into a polygon grid. For each converted patch, we set the sixteen control points, then construct a pair of (u, v) values for each vertex of the grid which is used to compute a position on the surface. The function returning this position, computes the four auxillary control points necesary to create a curve along the v direction, using the u parameter. We can then use the v parameter to compute the position on the surface along this curve. The rest of the code sets an array to connect the vertices together, and pushes the resulting polygon mesh to the list of objects.
Here is the code.

```
glm::vec3 mlBezier::mlEvalBezierCurve(const glm::vec3 * P, const float & t)
{
    float b0 = (1 - t) * (1 - t) * (1 - t) * (1 - t);
    float b1 = 4 * t * (1 - t) * (1 - t) * (1 - t);
    float b2 = 6 * t * t * (1 - t) * (1 - t);
    float b3 = 4 * t * t * t * (1 - t);
    float b4 = t * t * t * t;
    return P[0] * b0 + P[1] * b1 + P[2] * b2 + P[3] * b3 + P[4] * b4;
}
```

```
glm::vec3 mlBezier::mlEvalBezierPatch
(const glm::vec3 * controlPoints,
    const float & u, const float & v)
{
    glm::vec3 uCurve[5];
    for (int i = 0; i < 5; i++) {
        glm::vec3 curveP[5];
        curveP[0] = controlPoints[i * 5];
        curveP[1] = controlPoints[i * 5 + 1];
        curveP[2] = controlPoints[i * 5 + 2];
        curveP[3] = controlPoints[i * 5 + 3];
        curveP[4] = controlPoints[i * 5 + 4];
        uCurve[i] = mlEvalBezierCurve(curveP, u);
    }
    return mlEvalBezierCurve(uCurve, v);
}
```

The implementation of lighting and camera manipulation are much similar to the assignment one.

© 2018
Computer Graphics I Course Assignment

1:2 • Name: Zhao Donghao
student number: 20096438
email: zhaodh@shanghaitech.edu.cn

3 RESULTS