计算机图形学 | hw8

一、理论基础

Bézier curve本质上是由调和函数（Harmonic functions）根据控制点（Control points）插值生成。其参数方程如下：

$Q(t) = \sum_{i=0}^{n} P_i B_{i,n}(t), t \in [0,1]$

上式为n次多项式，具有 n + 1项。其中，$Pi(i = 0, 1 \dots n)$表示特征多边形的n + 1个顶点向量； $Bi,n(t)$为伯恩斯坦（Bernstein）基函数，其多项式表示为：

$B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}, i = 0, 1, 2 \cdots \cdot n$

二、代码实现

曲线绘制

```
curveCount = 0;
if (line.size() >= 2) {
    for (float t = 0; t < 1; t += 0.001) {
        float cx = 0, cy = 0;
        for (int i = 0, n = line.size()-1; i <= n; i++) {
            float bernstein = factorial(n) / (factorial(i) * factorial(n - i)) * pow(t, i) *
pow(1 - t, n - i);
            cx += line[i].x * bernstein;
            cy += line[i].y * bernstein;
        }
        curve[curveCount * 2] = cx;
        curve[curveCount * 2 + 1] = cy;
        curveCount++;
    }
}

// 计算阶乘
long int factorial(int x) {
    if (x == 0) return 1;
    int result = 1;
    for (int i = 1; i <= x; i++) {
        result *= i;
    }
    return result;
}
```

动态生成过程

动态生成的过程使用了递归，对于n个节点的特征多边形，就需要绘制n-1次，节点个数从n-1递减到1，再每一次递归的过程中，进行一次二次多项式$Q_i(t) = (1-t)P_i - t * P_{i+1}$，依次递归下去就可以成功实现。

```
void transform(vector<glm::vec2> vertex) {
```

```
    int n = vertex.size();
    if (n == 1) return;

    vector<glm::vec2> nextVertexs = vector<glm::vec2>();
    for (int i = 0; i < n - 1; i++) {
        float tx = (1 - animation) * vertex[i].x + animation * vertex[i + 1].x;
        float ty = (1 - animation) * vertex[i].y + animation * vertex[i + 1].y;
        glm::vec2 nextVertex = glm::vec2(tx, ty);
        transformVertex[i * 2] = tx;
        transformVertex[i * 2 + 1] = ty;
        nextVertexs.push_back(nextVertex);
    }

    unsigned int VBO;
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, 2 * sizeof(float) * nextVertexs.size(), transformVertex,
GL_STATIC_DRAW);

    unsigned int VAO;
    glBindBuffer(GL_ARRAY_BUFFER, VBO);

    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);

    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glPointSize(10.0f);
    glDrawArrays(GL_POINTS, 0, nextVertexs.size());

    glPointSize(1.0f);
    glDrawArrays(GL_LINE_STRIP, 0, nextVertexs.size());

    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);

    transform(nextVertexs);
}
```

三、实验效果


LearnOpenGL