计算机图形学 Homework 8

(一) 作业要求

Basic:

1. 用户能通过左键点击添加Bezier曲线的控制点，右键点击则对当前添加的最后一个控制点进行消除
2. 工具根据鼠标绘制的控制点**实时**更新Bezier曲线。

Hint: 大家可查询捕捉mouse移动和点击的函数方法

Bonus:

1. 可以动态地呈现Bezier曲线的生成过程。

(二) 代码解释

1. 计算 Bernstein 基函数

```
// 计算Bernstein基函数
// ------------------
float computeBernstein(int i, int n, float t) {
    int numerator = 1, denominator = 1;
    for (int k = n; k > n - i; k--) { // 计算阶乘分子
        numerator *= k;
    }
    for (int k = i; k > 0; k--) { // 计算阶乘分母
        denominator *= k;
    }
    float result = (float)numerator / (float)denominator * pow(t, i) * pow(1 - t, n - i);
    return result;
}
```

根据公式 $B_{i,n}(t) = \frac{n!}{i!(n-i)!}t^i(1-t)^{n-i} = \frac{n(n-1)\dots(n-i+1)}{i!}t^i(1-t)^{n-i}, \mathrm{n} = 0,1,\dots,\mathrm{n}$ 计算 Bernstein 基函数。

2. 计算 Bezier 曲线 Q(t)

```
// 计算Bezier曲线Q(t)
// ------------------
glm::vec2 computeBezierPoint(deque<glm::vec2> controlPoints, float t) {
    glm::vec2 q = glm::vec2(0.0f);

    // Bezier曲线公式
    int n = controlPoints.size() - 1;
    for (int i = 0; i <= n; i++) {
        glm::vec2 p = controlPoints[i];
        float b = computeBernstein(i, n, t);
        q += p * b;
    }

    return q;
}
```

Bezier 曲线本质上由调和函数（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，$P_i(i = 0,1,\dots,n)$表示特征多边形的 n+1 个顶点向量；$B_{i,n}(t)$为伯恩斯坦（Bernstein）基函数。

3. 使用计数器获得 Bezier 曲线的参数 t

```
const unsigned int MAX_COUNT = 1000;

// 创建计数器
// ----------
unsigned int count = 0;

// 更新计数器
// ----------
count++;
if (count > MAX_COUNT) {
    count = 0;
}
```

整型变量 count 取值范围为[0, MAX_COUNT]，其值在每个 while 循环得到更新。通过计算 count / MAX_COUNT 得到参数 t∈[0, 1]。

4. 鼠标左键添加控制点，鼠标右键删除最后一个控制点

```
// Bezier曲线控制点
deque<glm::vec2> controlPoints;

// 鼠标按键回调函数
// -----------------
void mouse_button_callback(GLFWwindow *window, int button, int action, int mods) {
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
        double xpos, ypos;
        glfwGetCursorPos(window, &xpos, &ypos);

        // 转换坐标空间
        double x = 2 * xpos / SCR_WIDTH - 1;
        double y = 1 - 2 * ypos / SCR_HEIGHT;

        controlPoints.push_back(glm::vec2(x, y));
    }
    else if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
        if (controlPoints.size()) {
            controlPoints.pop_back();
        }
    }
}
```

使用双端队列 deque 存储控制点，便于高效插入和删除。

如果按下鼠标左键，首先通过 glfwGetCursorPos 函数获得鼠标当前位置，注意此时的位置坐标为屏幕坐标，即窗口左上角为(0, 0)，右下角为(width, height)。然后根据公式

$$x = \left(\frac{x}{width} - 0.5\right) \times 2$$

$$y = \left(-\frac{y}{height} + 0.5\right) \times 2$$

转换坐标空间，使得窗口中心位置为(0, 0)，并调转 y 轴方向。

如果按下鼠标右键，调用 deque 的 pop_back 函数删除最后一个控制点。

5. 绘制 Bezier 曲线

```
if (controlPoints.size()) {
    // 计算Bezier曲线
    // --------------
    vector<float> bezierPoints;
    for (unsigned int k = 0; k <= count; k++) {
        float t = (float)k / (float)MAX_COUNT;
        glm::vec2 q = computeBezierPoint(controlPoints, t);
        bezierPoints.push_back(q.x);
        bezierPoints.push_back(q.y);
    }

    // 设置Bezier曲线VAO和VBO
    // ----------------------
    unsigned int VAO, VBO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);

    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * bezierPoints.size(), &bezierPoints[0], GL_DYNAMIC_DRAW);

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

    // 绘制Bezier曲线
    // --------------
    shader.use();
    glBindVertexArray(VAO);
    glDrawArrays(GL_POINTS, 0, bezierPoints.size() / 2);
```

① 调用 computeBezierPoint 函数绘制 t 时刻以及 t 时刻之前的点，存储到 vector 数组中

② 创建并绑定 VAO 和 VBO，设置顶点属性指针

③ 调用 glDrawArrays 函数绘制点

6. 绘制 Bezier 曲线动态生成过程中的直线

```
// 绘制Bezier曲线生成过程中的直线
// ------------------------------
void drawLines(Shader &shader, vector<float> points, float t) {
    if (points.size()) {
        // 设置直线VAO和VBO
        // ----------------
        unsigned int lineVAO, lineVBO;
        glGenVertexArrays(1, &lineVAO);
        glGenBuffers(1, &lineVBO);

        glBindVertexArray(lineVAO);
        glBindBuffer(GL_ARRAY_BUFFER, lineVBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * points.size(), &points[0], GL_DYNAMIC_DRAW);

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

        // 绘制直线
        // --------
        shader.use();
        glBindVertexArray(lineVAO);
        glDrawArrays(GL_LINE_STRIP, 0, points.size() / 2);
```

传入控制点的 vector 数组，创建并绑定直线的 VAO 和 VBO，设置顶点属性指针，调用 glDrawArrays 函数绘制直线。

```
// 计算每条直线上新的控制点
// ------------------------
vector<float> newPoints;
for (unsigned int k = 0; k < points.size() - 2; k += 2) {
    float x0 = points[k];
    float y0 = points[k + 1];
    float x1 = points[k + 2];
    float y1 = points[k + 3];
    float newX = (1 - t) * x0 + t * x1;
    float newY = (1 - t) * y0 + t * y1;
    newPoints.push_back(newX);
    newPoints.push_back(newY);
}
```

根据公式$\mathrm{P(t)} = (1 - \mathrm{t})P_0 + tP_1$获得直线上新的控制点，将这些控制点添加到一个新的 vector 数组。

```
// 递归绘制直线
// ------------
drawLines(shader, newPoints, t);
```

递归调用 drawLines 函数绘制直线，直到绘制完最后一个控制点，该点即为 Bezier 曲线 t 时刻的点。

(三) 运行结果