# Homework 8 - Bezier Curve

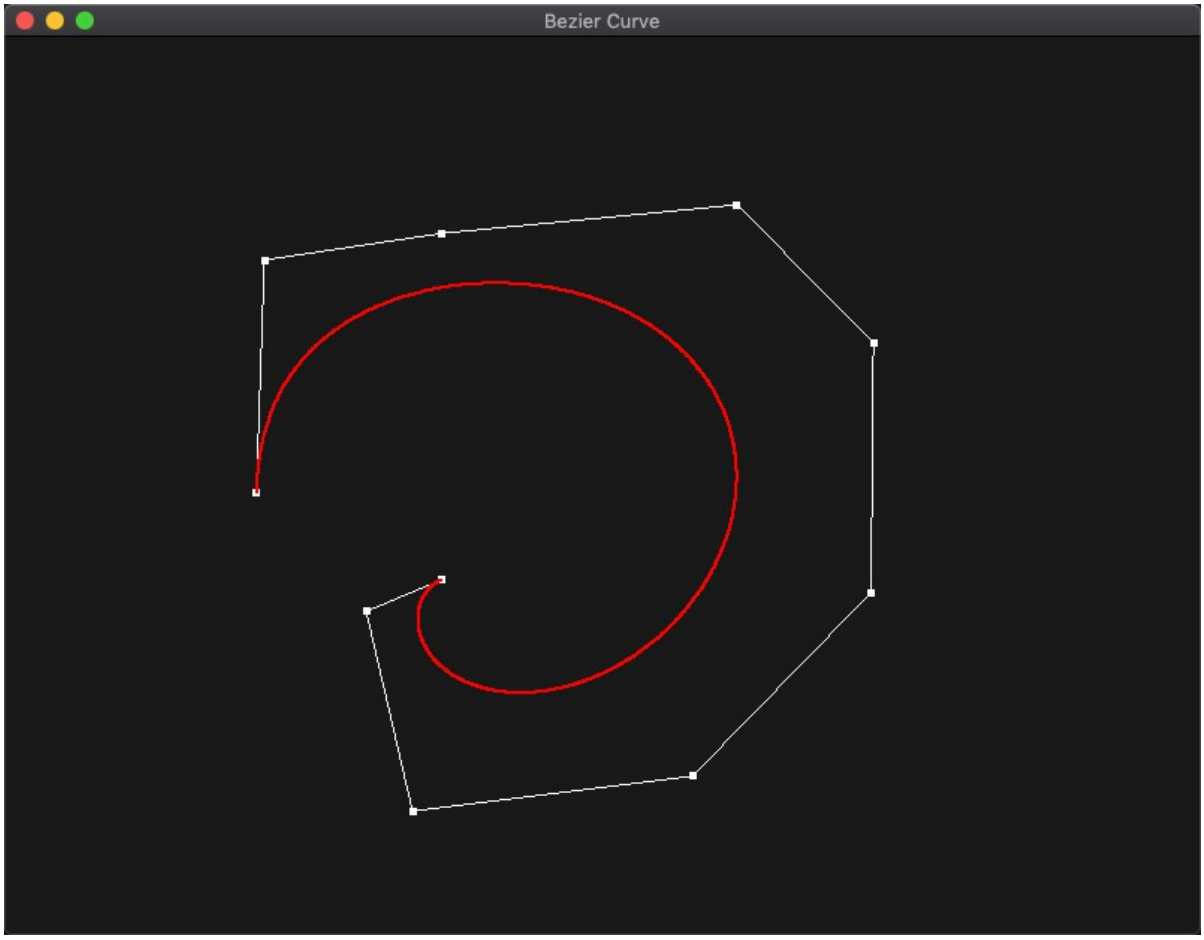
**效果视频：**见 doc/demo.mov 或 https://pan.baidu.com/s/1vNpMIsC47bXJgTzDO2YGfA

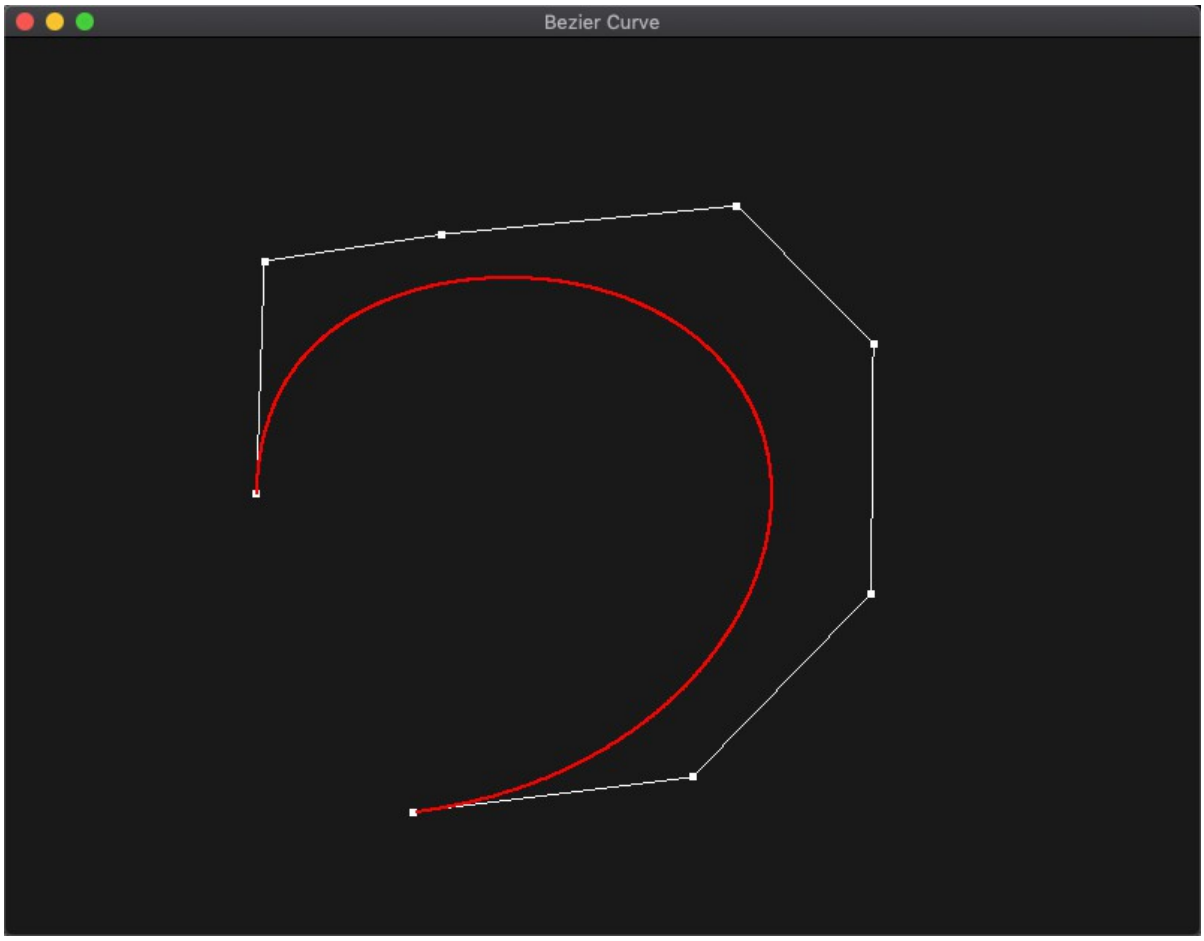
## Basic:

> 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。

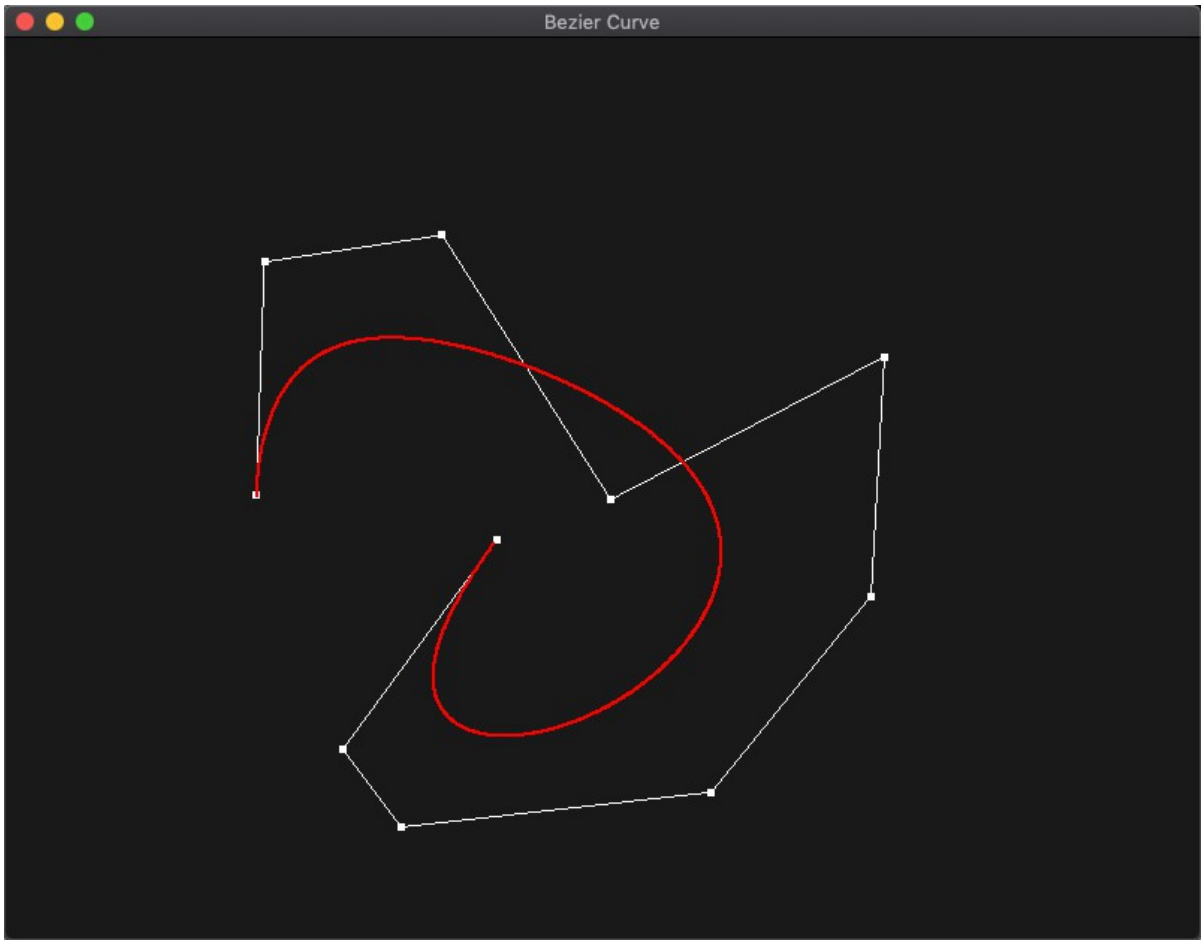
## 效果展示：

1. 用户点击鼠标左键添加控制点，右键对当前添加的最后一个控制点进行消除，Bezier 曲线实时更新。






2. 当鼠标光标移动到控制点的附近时，可以进行拖拽以改变控制点的位置。



## 实现步骤：

1. 添加全局变量 `vector<glm::vec3> controlPoints;` 存储所有控制点的坐标。控制点的添加和删除可以通过在GLFW 中监听鼠标输入的回调函数中完成。

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    double xpos, ypos;
    glfwGetCursorPos(window, &xpos, &ypos);

    if (button == GLFW_MOUSE_BUTTON_LEFT) {
        // add one point on the canvas  && move the selected points
        if (action == GLFW_PRESS) {
            isLeftButtonPressed = true;
            if (findNearestControlPoint(xpos, ypos, 100) == controlPoints.end()) {
                // add the selected point
                controlPoints.push_back(glm::vec3(xpos, ypos, 0.0f));
            }
        }

        if (action == GLFW_RELEASE) {
            currPointIter = controlPoints.end();
            isLeftButtonPressed = false;
        }
    }

    if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
        // delete the last control point on the canvas
        if(!controlPoints.empty()){
           controlPoints.pop_back();
        }
    }
}
```

在光标附近无已有控制点则添加新的控制点

删除最后一个控制点

其中 `findNearestControlPoint` 会遍历所有控制点的位置，以粗略查找点击范围中是否有可以控制的点，有的话则返回最近的控制点的 iterator。

2. 在渲染循环中，完成控制点坐标的实时改变。

```
// Render Control Points
if (isLeftButtonPressed) {
    // record the selected point index
    double xpos, ypos;
    glfwGetCursorPos(window, &xpos, &ypos);
    // check if can selected the nearest control point
    currPointIter = findNearestControlPoint(xpos, ypos, 100);
    if (currPointIter != controlPoints.end()){
        // update position
        *currPointIter = glm::vec3(xpos, ypos, 0.0f);
    }
}
```

在拖拽模式下，如果找到最近的控制点，则更新控制点的位置为当前光标位置

3. 在渲染之前，将 controlPoints 中的屏幕坐标转换到 NDC 坐标系下的坐标，并存储为包含顶点数据的 float 数组。

```
vector<GLfloat> controlPoints2floatVector(vector<glm::vec3> cp){
    vector<GLfloat> res;
    res.clear();
    for (auto iter = cp.begin(); iter != cp.end(); iter++){
        res.push_back(iter->x);
        res.push_back(iter->y);
        res.push_back(iter->z);
    }
    // normalize to [-1, 1]
    for (unsigned int i = 0; i < res.size(); i = i + 3) {
        auto norx = (2 * res[i]) / SCR_WIDTH - 1;
        auto nory = 1 - (2 * res[i + 1]) / SCR_HEIGHT;
        res[i] = norx;
        res[i + 1] = nory;
    }
    return res;
}
```

4. 然后就可以将控制点渲染出来了，每次需要同时绘制点和之间的连线。

```
void renderPointsLines(Shader &shader,const vector<GLfloat> &nodeData, float pointSize)
    {                                          点的 NDC 坐标数组          点的大小
    shader.use();
    // bind the Vertex Array Object first, then bind and set vertex buffer(s), and then
        configure vertex attributes(s).
    glBindVertexArray(pointVAO);
    glBindBuffer(GL_ARRAY_BUFFER,pointVBO);

    glBufferData(GL_ARRAY_BUFFER, nodeData.size() * sizeof(GLfloat), nodeData.data(),
        GL_STATIC_DRAW);
    // position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glPointSize(pointSize);
    glDrawArrays(GL_POINTS, 0, int(nodeData.size() / 3));

    // Render lines between control points
    glLineWidth(1.0f);
    glDrawArrays(GL_LINE_STRIP, 0, int(nodeData.size() / 3));
    glBindVertexArray(0);
}
```

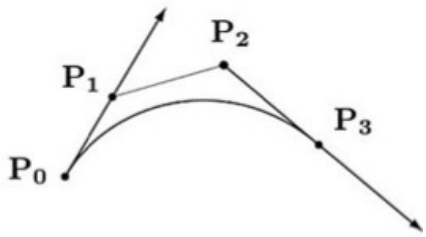
5. 下面画 Bezier 曲线，如下所示的参数方程，曲线上的点的坐标可以由控制点插值生成，每个0-1之间的 t 值可以生成曲线上的一个点。

- 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：

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

- 上式为$n$次多项式，具有$n+1$项。其中，$P_i(i=0,1\ldots n)$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**，其多项式表示为：

$$B_{i,n}(t)=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i}\,,\quad i=0,\ 1\ldots n$$




建立 curveShader，首先接受控制点的个数，所有控制点的 NDC 坐标和当前 t 值，然后在片段着色器中计算伯恩斯坦基函数，最后对控制点的坐标进行插值，得到对应的曲线上的点的坐标。

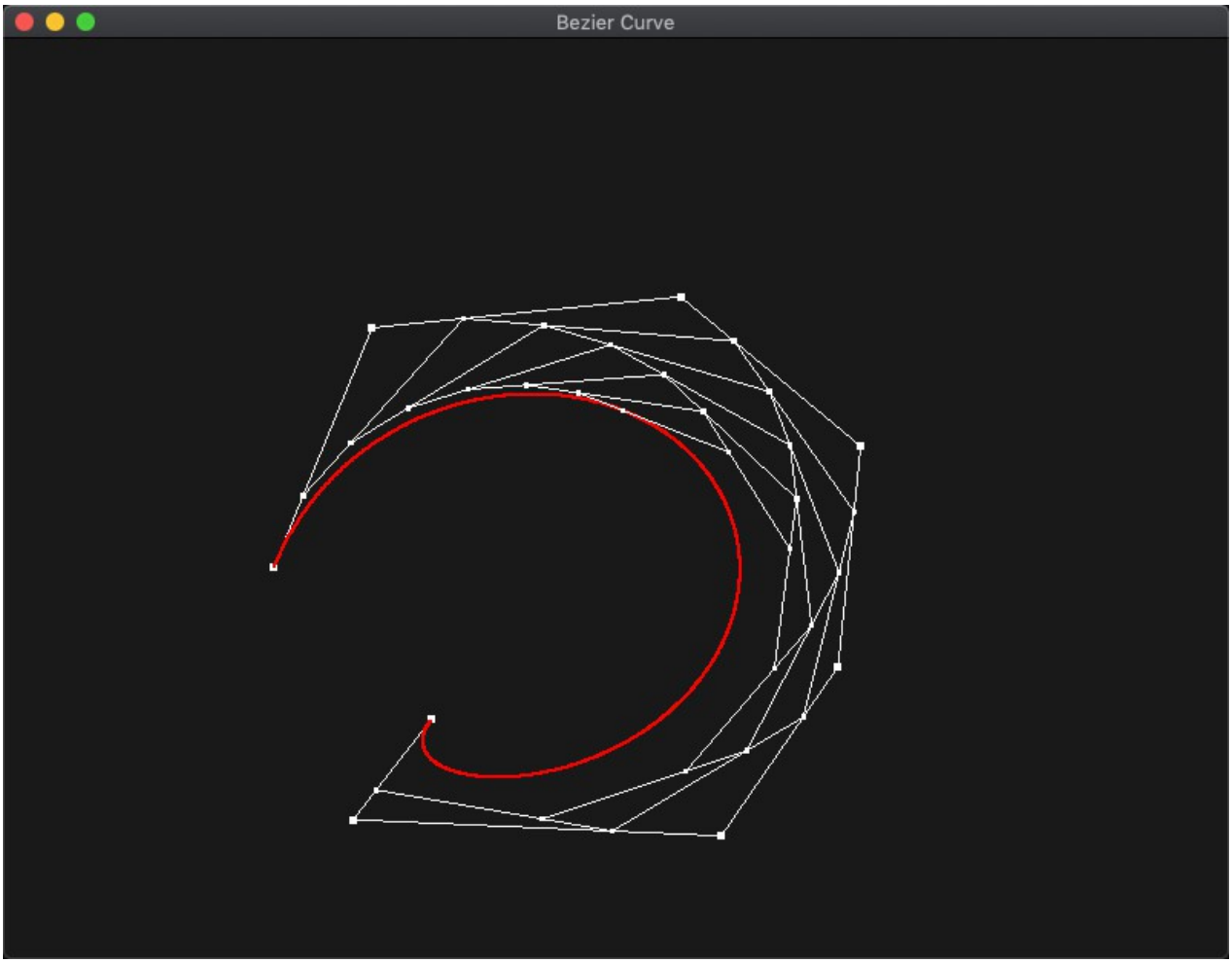
```
#version 330 core
#define MAX_POINTS 128
layout(location = 0) in float t;

uniform int size;
uniform vec3 points[MAX_POINTS];

int factorial(int n){
    int r = 1;
    for(int i = n; i>0; i--){
        r *= i;
    }
    return r;
}

float bernstein(int i, int n, float t){
    float r =  float(factorial(n)) / float((factorial(i) * factorial(n - i)));
    r *= pow(t,i);
    r *= pow(1-t,n-i);
    return r;
}

void main(){
    vec3 res;
    int n = size-1;
    for (int i = 0; i < size; i++){
        float b = bernstein(i, n, t);
        res += points[i] * b;
    }
    gl_Position = vec4(res, 1.0f);
}
```

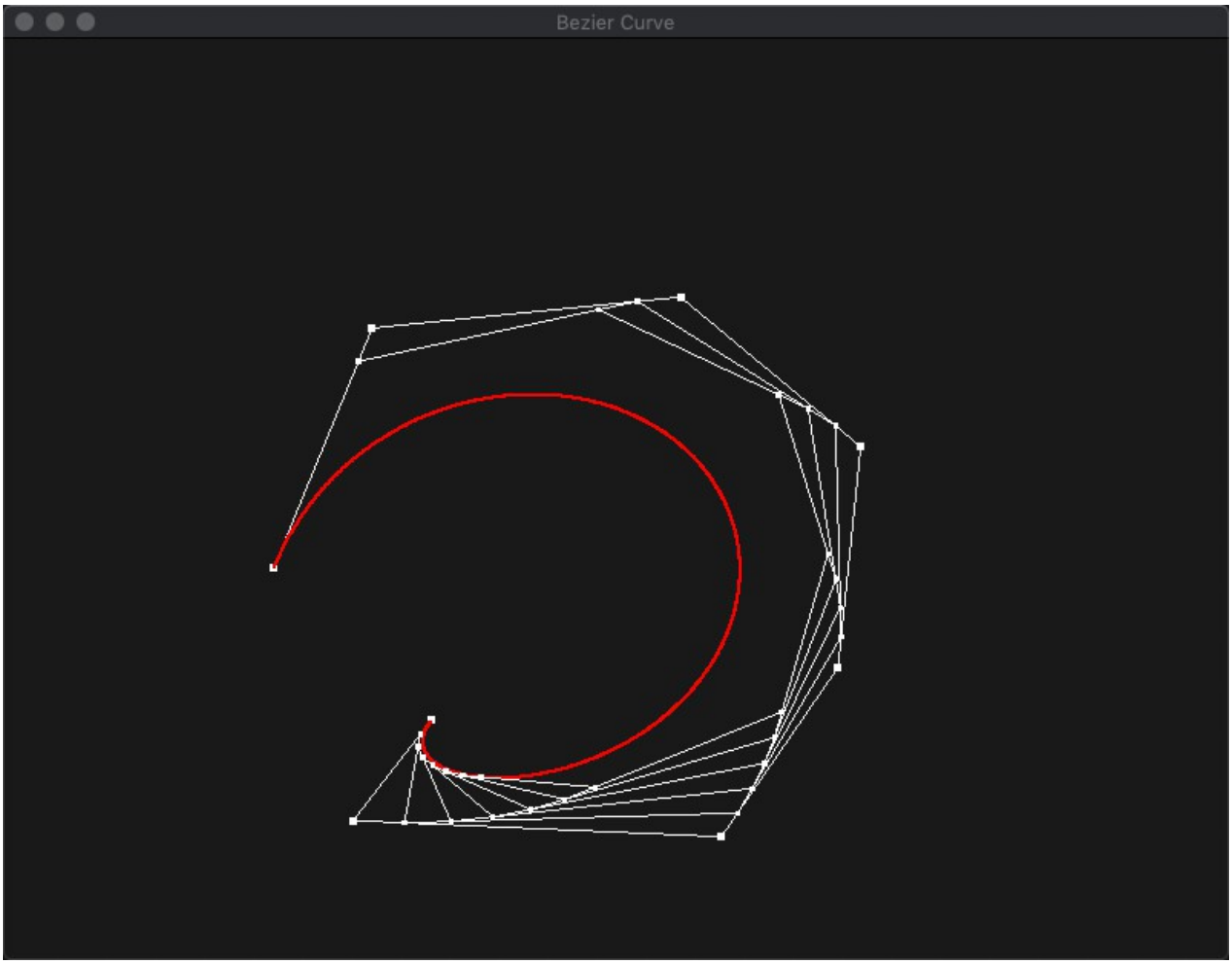
6. 在渲染循环中传入 uniform 变量给 curveShader 即可以完成实时更新。

# Bonus

> 可以动态地呈现Bezier曲线的生成过程。

## 效果展示






## 实现步骤

1. 添加对键盘 D 键输入的处理，可以将模式切换到动态模式。

```
void processInput(GLFWwindow *window)
{
    if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS){
        glfwSetWindowShouldClose(window, true);
    }

    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS){
        isDrawing = !isDrawing;
    }
}
```

2. 添加 `void draw_process(vector<glm::vec3> points, float t)` 函数，对于给定的 t，先在输入的 n + 1 个点形成的 n 条线上各取比例为 t 的点，即 $(1-t)P_{i-1}+tP_i$ ，得到 n 个点，然后再去渲染这 n 个点和它们之间的连线。最后再用这 n 个点递归调用 draw_process，t 不变，执行相同的过程，最后得到的点就在 Bezier 曲线上。

```
void draw_process(vector<glm::vec3> points, float t){
    if (points.size() < 2){
        return;
    }
    vector<glm::vec3> next_nodes;
    for (int i = 0; i < points.size() - 1; ++i)
    {
        float x = (1 - t) * points[i].x + t * points[i + 1].x;
        float y = (1 - t) * points[i].y + t * points[i + 1].y;
        next_nodes.push_back(glm::vec3(x, y, 0.0f));
    }
    auto nodeData = controlPoints2floatVector(next_nodes);
    renderPointsLines(pointShader, nodeData, 3.5f);
    draw_process(next_nodes, t);
}
```

3. 在渲染循环中，当进入动态模式时，首先停止捕捉鼠标状态，并让 t 从 0 开始，每个渲染循环增加 0.0005，最后传入当前控制点和 t 给 `draw_process` 进行递归画线。退出动态模式后，重新可以捕捉鼠标状态，并重置 t 为0，保证更新控制点后可以重新显示动画。

```
// dynamic draw
if(isDrawing){
    // deactivate the mouse movement
    glfwSetMouseButtonCallback(window, NULL);
    if (current_t <= 1.0) {
        current_t += 0.0005;
        draw_process(controlPoints, current_t);
    }
} else {
    current_t = 0.0;
    // reactivate the mouse movement
    glfwSetMouseButtonCallback(window, mouse_button_callback);
}
```