计算机图形学实验报告

学号：16340054

姓名：戴馨乐

学院：数据科学与计算机学院

作业：hm8

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**

   经过查阅文档，得知:

   监听鼠标位置的函数为:

   $$glfwSetCursorPosCallback(GLFWwindow * window, cursor\_positon\_callback)$$

   监听鼠标点击的函数为:

   $$glfwSetMouseButtonCallback(GLFWwindow * window, mouse\_button\_callback)$$

```
// 处理鼠标
glfwSetCursorPosCallback(window, cursor_position_callback);
glfwSetMouseButtonCallback(window, mouse_button_callback);
```

   那么，要实现上面的功能就需要在两个回调函数里面进行编写了。

   首先，为了方便之后 bezier 曲线生成算法的实现，这里使用一个 list 来保存添加的控制点

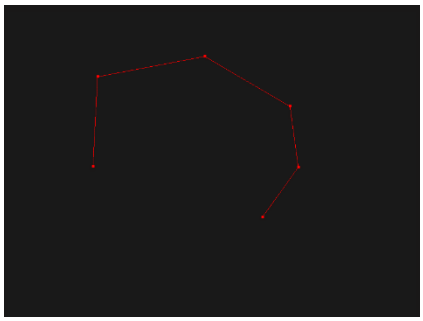
```
// 控制点
list<MyPoint> control_points;
float* control_points_arr = NULL;
```

   然后，由于监听鼠标的函数并不会将鼠标位置传给回调函数，所以这里点击鼠标左键，我们只能更新设置一个状态，这个状态决定了是否添加入新的控制点，然后监听鼠标位置的函数会在状态设置为 true 的时候，将新点添加进入控制点的 lis 中。然后点击鼠标右键，只需要将最尾部的点 pop 出来，即可实现消除最后一个控制点的效果。

```
// 处理鼠标位置事件
void cursor_position_callback(GLFWwindow* window, double x, double y) {
    if (canAddNewControlPoint) {
        selectedPoint.x = x;
        selectedPoint.y = y;
        selectedPoint.z = 0.0f;
        normalize_point(selectedPoint);
        control_points.push_back(selectedPoint);
        canAddNewControlPoint = false;
    }
}
```

```
// 处理鼠标点击事件
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods) {
    if (action == GLFW_PRESS)
        switch (button)
        {
        case GLFW_MOUSE_BUTTON_LEFT:
            canAddNewControlPoint = true;
            break;
        case GLFW_MOUSE_BUTTON_RIGHT:
            if (!control_points.empty()) {
                control_points.pop_back();
            }
            break;
        }
}
```

运行效果：



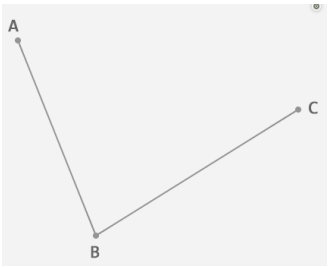
2. **工具根据鼠标绘制的控制点实时更新 Bezier 曲线**

关于 bezier 曲线的学习参考了这篇博客：beizier 曲线

- **原理分析**

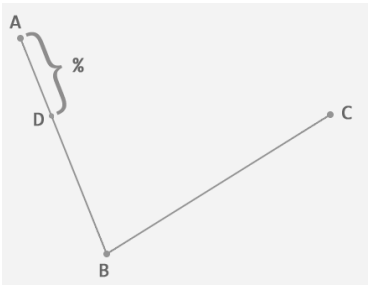
以 3 个控制点为例：

有 A，B，C 3 个控制点：



然后定义一个比例为 t，范围在$0-1$之间。

在向量AB上，得到一个中间点D，其中$|AC|:|AB|=t$

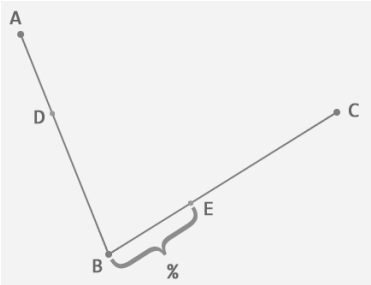


可以通过推算得到 D 点的坐标：

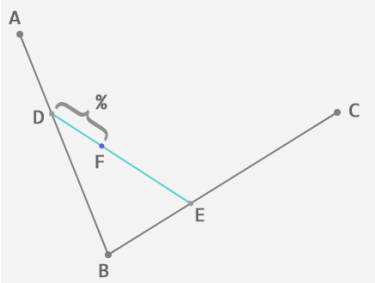
$x_D = tx_B + (1-t)x_A$

$y_D = ty_B + (1-t)y_A$

同样，在向量BC上也可以通过同样的方法得到中间点E



然后以点D，E为新的控制点，得到了中间点F



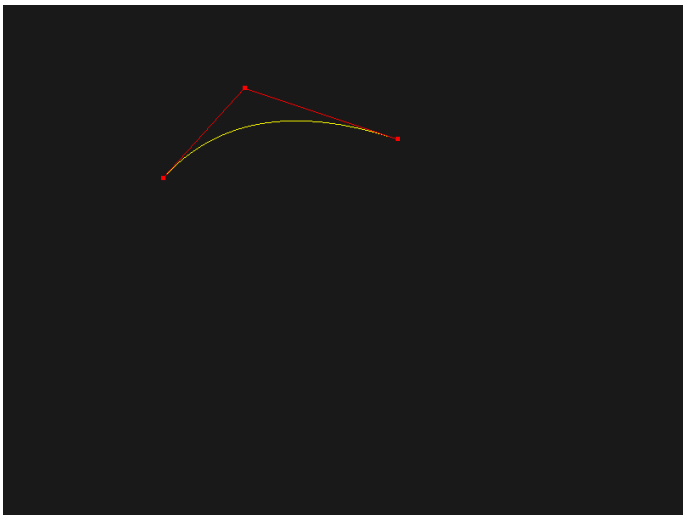
由于只有 F 一个点了，所以 F 点为曲线上的点。

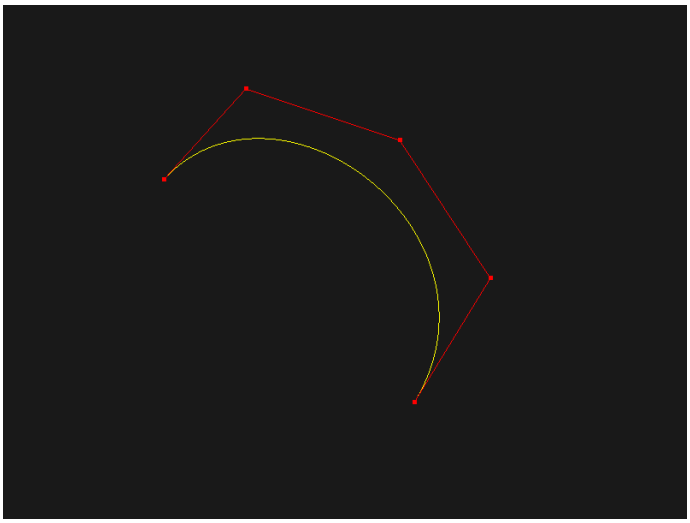
那么，要得到曲线，只需要将比例从 0 逐渐增加到 1，得到的曲线点就可以组成需要的直线了。

实现上采用链表的后入先出的特点来进行实现如下:

```
// 生成贝塞尔曲线
void generate_bezier_curve() {
    curve_points.clear();
    middle_points.clear();
    MyPoint endPoint(-1, -1, -1, true);
    if (control_points.size() <= 1) {
        return;
    }
    for (float i = 0.0; i <= 1; i += step) {
        auto tmp_ctr_points = control_points;
        tmp_ctr_points.push_back(endPoint);
        vector<MyPoint> temp;
        while (true) {
            if (tmp_ctr_points.size() == 2) break;
            MyPoint point1 = tmp_ctr_points.front();
            tmp_ctr_points.pop_front();
            MyPoint point2 = tmp_ctr_points.front();
            if (!point2.isEndPoint()) {
                MyPoint middlePoint = get_curve_point(point1, point2, i);
                tmp_ctr_points.push_back(middlePoint);
                if (i == currentRatio) {
                    temp.push_back(middlePoint);
                }
            }
            else {
                tmp_ctr_points.pop_front();
                tmp_ctr_points.push_back(endPoint);
                if (i == currentRatio) {
                    middle_points.push_back(temp);
                    temp.clear();
                }
            }
        }
        curve_points.push_back(tmp_ctr_points.front());
    }
}
```

运行效果：





Bonus：

1. **可以动态地呈现 Bezier 曲线的生成过程**

   要实时动态呈现生成 Bezier 曲线的过程，首先要获取各个中间点，然后对应连线。

   以 3 个控制点为例，可以将其分为 3 组：

   1. A，B，C
   2. E，F
   3. G

   那么我们要的中间点其实就是 E，F，G

   然后将各个组中的点各自连线，就得到了该比例下的生成状态。

   要动态呈现所有状态，可以在每次渲染时候，按照固定的步长改变比例，从 0 变化到 1 再循环这个过程，从而可以展示所有的动态呈现从 0 到 1 的生成状态。

   代码实现：

   在生成贝塞尔曲线过程中，获取当前比例的中间点：

```
// 生成贝塞尔曲线
void generate_bezier_curve() {
    curve_points.clear();
    middle_points.clear();
    MyPoint endPoint(-1, -1, -1, true);
    if (control_points.size() <= 1) {
        return;
    }
    for (float i = 0.0; i <= 1; i += step) {
        auto tmp_ctr_points = control_points;
        tmp_ctr_points.push_back(endPoint);
        vector<MyPoint> temp;
        while (true) {
            if (tmp_ctr_points.size() == 2) break;
            MyPoint point1 = tmp_ctr_points.front();
            tmp_ctr_points.pop_front();
            MyPoint point2 = tmp_ctr_points.front();
            if (!point2.isEndPoint()) {
                MyPoint middlePoint = get_curve_point(point1, point2, i);
                tmp_ctr_points.push_back(middlePoint);
                if (i == currentRatio) {
                    temp.push_back(middlePoint);
                }
            }
            else {
                tmp_ctr_points.pop_front();
                tmp_ctr_points.push_back(endPoint);
                if (i == currentRatio) {
                    middle_points.push_back(temp);
                    temp.clear();
                }
            }
        }
        curve_points.push_back(tmp_ctr_points.front());
    }
}
```

将各组中间点连线，渲染出来：

```
// 渲染生成曲线的过程
void render_middle_process() {
    if (control_points.size() <= 1) return;
    if (middleVAO == 0) {
        glGenVertexArrays(1, &middleVAO);
        glGenBuffers(1, &middleVBO);
    }
    for (int i = 0; i < middle_points.size(); i++) {
        // 转成数组
        convert_pointvector_to_arr(middle_points[i], middle_points_arr);
        // 绑定
        glBindVertexArray(middleVAO);
        glBindBuffer(GL_ARRAY_BUFFER, middleVBO);
        glBufferData(GL_ARRAY_BUFFER, middle_points[i].size() * 2 * sizeof(float), middle_points_arr, GL_STATIC_DRAW);
        glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
        glEnableVertexAttribArray(0);
        glBindVertexArray(middleVAO);
        glPointSize(5.0f);
        glDrawArrays(GL_POINTS, 0, middle_points[i].size());
        glDrawArrays(GL_LINE_STRIP , 0, middle_points[i].size());
        glBindVertexArray(0);

    }
    currentRatio += step;
    if (currentRatio >= 1.0) {
        currentRatio = step;
    }
}
```

运行效果：