Homework 9 - Bezier Curve

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。

代码思路

> 贝塞尔曲线于1962年，由法国工程师皮埃尔·贝塞尔（Pierre Bézier）所广泛发表，他运用贝塞尔曲线来为汽车的主体进行设计。贝塞尔曲线最初由Paul de Casteljau 于1959年运用de Casteljau 算法开发，以稳定数值的方法求出贝塞尔曲线。

鼠标移动和点击

调用 `glfwSetCursorPos()` 函数来获取鼠标移动时的位置，用全局变量lastX和lastY来确定最后鼠标点击时的位置

```
void glfwSetCursorPos ( GLFWwindow *  window,
                        double        xpos,
                        double        ypos
                      )
```

This function sets the position, in screen coordinates, of the cursor relative to the upper-left corner of the content area of the specified window. The window must have input focus. If the window does not have input focus when this function is called, it fails silently.

Do not use this function to implement things like camera controls. GLFW already provides the `GLFW_CURSOR_DISABLED` cursor mode that hides the cursor, transparently re-centers it and provides unconstrained cursor motion. See **glfwSetInputMode** for more information.

If the cursor mode is `GLFW_CURSOR_DISABLED` then the cursor position is unconstrained and limited only by the minimum and maximum values of a `double`.

Parameters

[in]	**window**	The desired window.
[in]	**xpos**	The desired x-coordinate, relative to the left edge of the content area.
[in]	**ypos**	The desired y-coordinate, relative to the top edge of the content area.

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    lastX = xpos;
    lastY = ypos;
}

glfwSetCursorPosCallback(window, mouse_callback);
```

用 `glfwSetMouseButtonCallback()` 函数来判定鼠标的点击，并在左键和右键点击时进行添加和删除的操作。

```
GLFWmousebuttonfun glfwSetMouseButtonCallback ( GLFWwindow *        window,
                                                GLFWmousebuttonfun  cbfun
                                              )
```

This function sets the mouse button callback of the specified window, which is called when a mouse button is pressed or released.

When a window loses input focus, it will generate synthetic mouse button release events for all pressed mouse buttons. You can tell these events from user-generated events by the fact that the synthetic ones are generated after the focus loss event has been processed, i.e. after the **window focus callback** has been called.

Parameters

[in]	**window**	The window whose callback to set.
[in]	**cbfun**	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been **initialized**.

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (action == GLFW_PRESS) {
        if (button == GLFW_MOUSE_BUTTON_LEFT) {
            Point new_point(-1.0 + 2 * (lastX / SCR_WIDTH), 1.0 - 2 * (lastY /
SCR_HEIGHT));
            bezier.addPoint(new_point);
        }
        else if (button == GLFW_MOUSE_BUTTON_RIGHT) {
            bezier.deletePoint();
        }
    }
    return;
}

glfwSetMouseButtonCallback(window, mouse_button_callback);
```

Bezier曲线

Bezier曲线的一般化公式

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

其中，Bernstein基函数$b_{i,n}(t)$为

$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, i = 0, 1, \ldots, n$

因此，可以写出辅助函数`BernsteinBasicFunc`

```
float Bezier::BernsteinBasicFunc(float t, int n, int i)
{
    return (fac(n) / (fac(i) * fac(n - i))) * pow(t, i) * pow((1 - t), (n - i));
}
```

其中，求阶乘的函数为

```
int Bezier::fac(int n) {
    if (n == 0) {
        return 1;
    }
    for (int i = n - 1; i >= 1; --i) {
        n *= i;
    }
    return n;
}
```

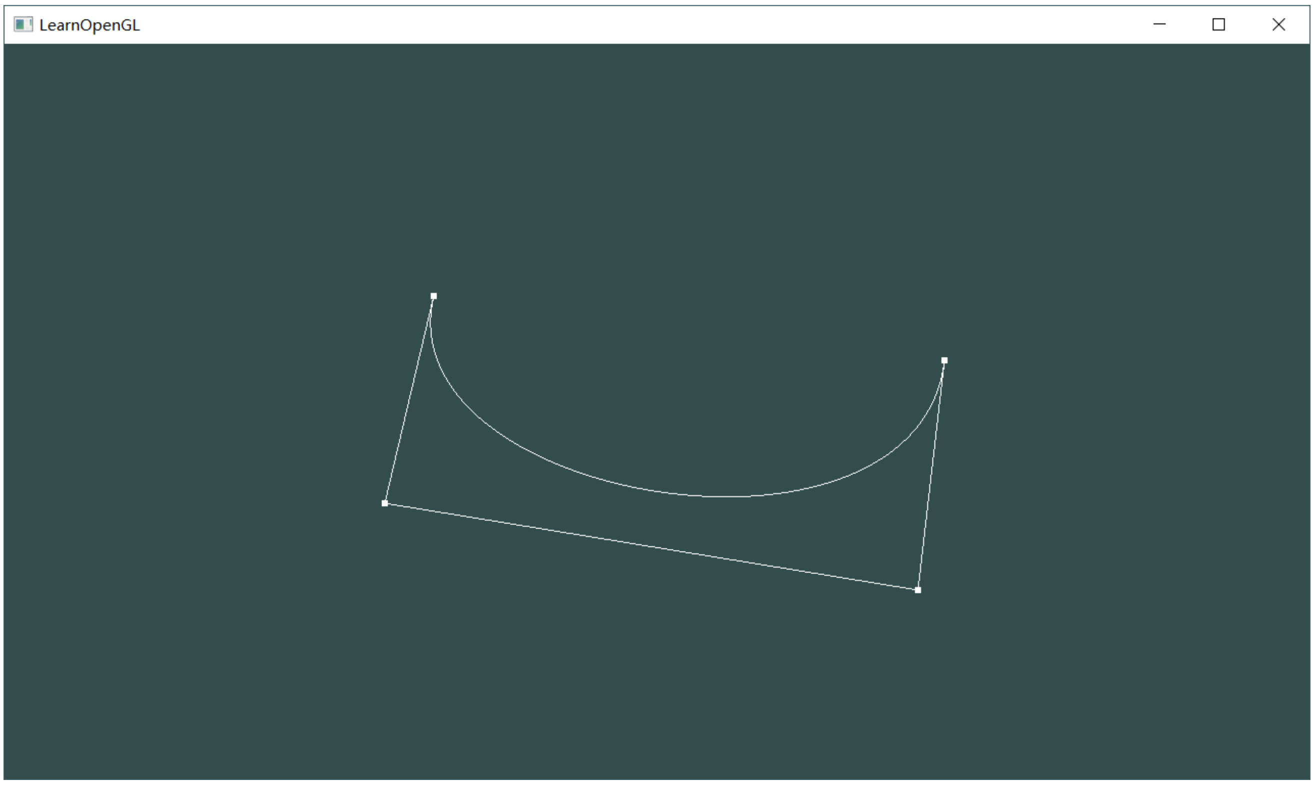
我们定义Point结构体，作为存储点的单元结构，并用vector容器进行存储，这里定义`vector<Point> points_list;`

```
struct Point {
    float x, y, z;
    Point() {};
    Point(float xPos, float yPos) {
        x = xPos;
        y = yPos;
        z = 0.0f;
    }
};
```

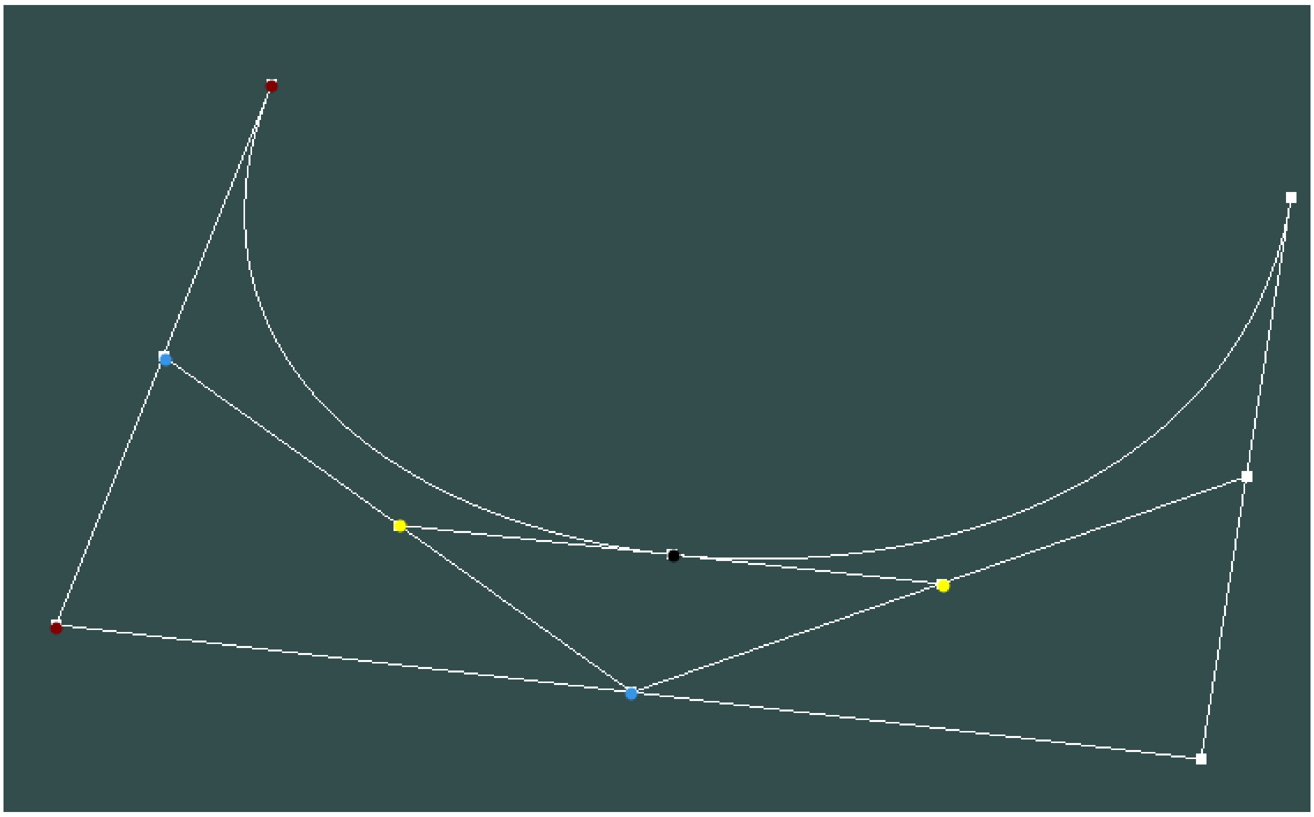
之后，由Bezier曲线公式，有

```
float xPos, yPos;
int count = 0;
int freq = 200;
float t;
for (t = 0; t <= 1; t += 1.0 / freq) {
    xPos = yPos = 0;
    for (int i = 0; i < num; ++i) {
        xPos += BernsteinBasicFunc(t, num-1, i) * points_list[i].x;
        yPos += BernsteinBasicFunc(t, num-1, i) * points_list[i].y;
    }
    vertices[3 * count] = xPos;
    vertices[3 * count + 1] = yPos;
    vertices[3 * count + 2] = 0.0f;
    count++;
}
```

结果如图：



动态展示



动态展示的思路是，如图，蓝色点在红色两点间“滑动”，黄色点在蓝色两点间“滑动”，依此类推，我们可以用一个循环来表示该过程，这里的“滑动”，可以由一阶Bezier曲线来表示，即

$B(t) = P_0 + (P_1-P_0)t = (1-t)P_0+tP_1$

```
for (int count = num - 1; count >= 0; count--) {
    for (int i = 0; i < count; i++) {
        vertices[3 * i] = (1 - t) * vertices[3 * i] + t * vertices[3 * i + 3];
        vertices[3 * i + 1] = (1 - t) * vertices[3 * i + 1] + t * vertices[3 * i + 4];
        vertices[3 * i + 2] = 0.0f;
    }
}
```