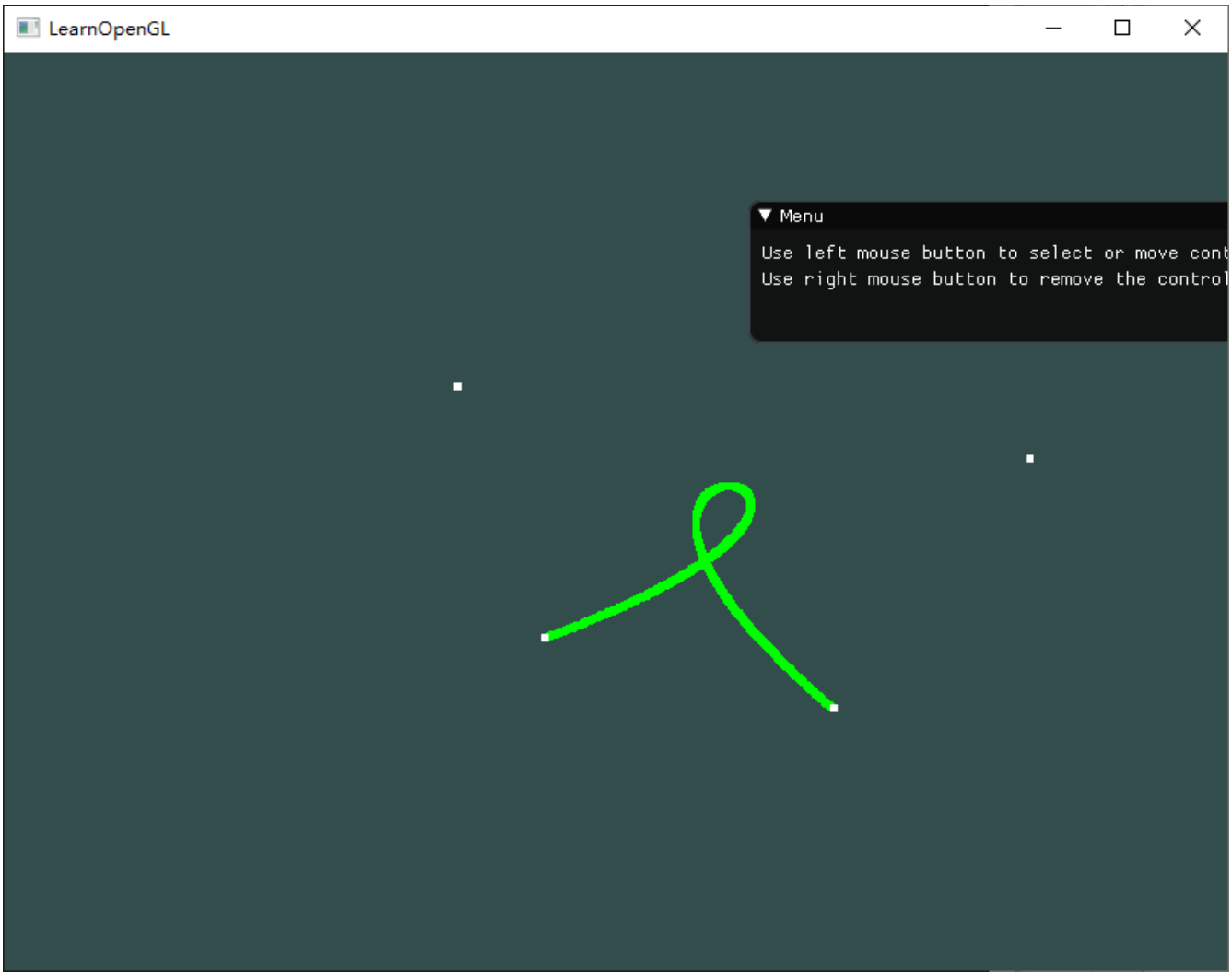
Homework 8

实验要求

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。

实验截图



算法描述

Bezier曲线绘制公式

$$\binom{n}{i} Pi(1-t)^{n-i} t^{i}$$

将贝塞尔曲线一般参数公式中的表达式用如下方式表示： 设有常数 a,b 和 c，则该表达式可统一表示为：
$a * (1-t)^{b} * t^{c} * Pn$

- N = 3: $P = (1-t)^2 * P0 + 2 * (1-t) * t * P1 + t^2 * P2$

  > a: 1 2 1 b: 2 1 0 c: 0 1 2

- N = 4: $P = (1-t)^3 * P0 + 3 * (1-t)^2 * t * P1 + 3(1-t) * t^2 * P2 + t^3 * P3$

  > a: 1 3 3 1 b: 3 2 1 0 c: 0 1 2 3

- N = 5: $P = (1-t)^4 * P0 + 4 * (1-t)^3 * t * P1 + 6(1-t)2 * t2 * P2 + 4 * (1-t) * t^3 * P3 + t^4 * P4$

  > a: 1 4 6 4 1 b: 4 3 2 1 0 c: 0 1 2 3 4

根据上面的分析就可以总结出 a,b,c 对应的取值规则：

> b: (N - 1) 递减到 0 (b 为 1-t 的幂)
>
> c: 0 递增到 (N - 1) (c 为 t 的幂)
>
> a: 在 N 分别为 1,2,3,4,5 时将其值用如下形式表示： N=1:———1 N=2:——-1 1 N=3:——1 2 1 N=4:—-1 3 3 1 N=5:—1 4 6 4 1 a 值的改变规则为： 杨辉三角

c++代码实现

```
/**
* @param poss       贝塞尔曲线控制点坐标
* @param precision 精度，需要计算的该条贝塞尔曲线上的点的数目
* @return 该条贝塞尔曲线上的点（二维坐标）
*/
vector<vector<float>> calculate(vector<glm::vec3> poss, int precision) {

    //维度，坐标轴数（二维坐标，三维坐标...）
    int dimersion = 2;

    //贝塞尔曲线控制点数（阶数）
    int number = poss.size();

    //控制点数不小于 2 ，至少为二维坐标系
    if (number < 2 || dimersion < 2)
        return vector<vector<float>>();

    vector<vector<float>> result;
    result.resize(precision);

    for (int i = 0; i < precision; ++i) {
        result[i].resize(dimersion);
    }

    //计算杨辉三角
    int* mi = new int[number];
    mi[0] = mi[1] = 1;
    for (int i = 3; i <= number; i++) {

        int* t = new int[i - 1];
        for (int j = 0; j < i-1; j++) {
            t[j] = mi[j];
        }
```

```
        mi[0] = mi[i - 1] = 1;
        for (int j = 0; j < i - 2; j++) {
            mi[j + 1] = t[j] + t[j + 1];
        }
    }

    //计算坐标点
    for (int i = 0; i < precision; i++) {
        float t = (float)i / precision;
        for (int j = 0; j < dimersion; j++) {
            float temp = 0.0f;
            for (int k = 0; k < number; k++) {
                temp += pow(1 - t, number - k - 1) * poss[k][j] * pow(t, k) * mi[k];
            }
            result[i][j] = temp;
        }
    }

    return result;
}
```

使用一个vector poss来保存控制点，在渲染循环中，只需要将控制点传入上述函数中，就可以得到曲线的点坐标，再传入shader中渲染即可，不过要把坐标转换为-1到1之间。

```
auto po = calculate(poss, 500)
auto controlPoints2dataVector = [po]() -> vector<GLfloat> {
    vector<GLfloat> res;
    res.clear();
    for (auto c : po) {
        res.push_back(c[0]);
        res.push_back(c[1]);
        res.push_back(0);
    }
    // 将数据归一化到[-1, 1]
    for (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;
};
auto pointData = controlPoints2dataVector()
glBindVertexArray(pVAO);
glBindBuffer(GL_ARRAY_BUFFER, pVBO);
glBufferData(GL_ARRAY_BUFFER, pointData.size() * sizeof(GLfloat), pointData.data(),
GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_TRUE, 3 * sizeof(GLfloat), (void*)0);
glEnableVertexAttribArray(0)
pointShader.use();
pointShader.setVec4("ourColor", glm::vec4(0.0f, 1.0f, 0.0f, 1.0f));
glBindVertexArray(pVAO);
```

```
glPointSize(5.0f);
glDrawArrays(GL_POINTS, 0, pointData.size() / 3);
glBindVertexArray(0);
```

注册一个鼠标按钮监听事件回调函数，如果左键按下，那么将当前坐标添加到控制点中，如果右键按下，那么删除该点。

```
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) {
            if (!isPointInVector(xpos, ypos)) {
                // add the selected point
                addPoint(xpos, ypos);
                cout << "add point" << xpos << "  " << ypos << endl;
            }
        }
    }

    if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
        // delete one point on the canvas
        poss.erase(findPointCanControlled(xpos, ypos, 80));
    }
}
```