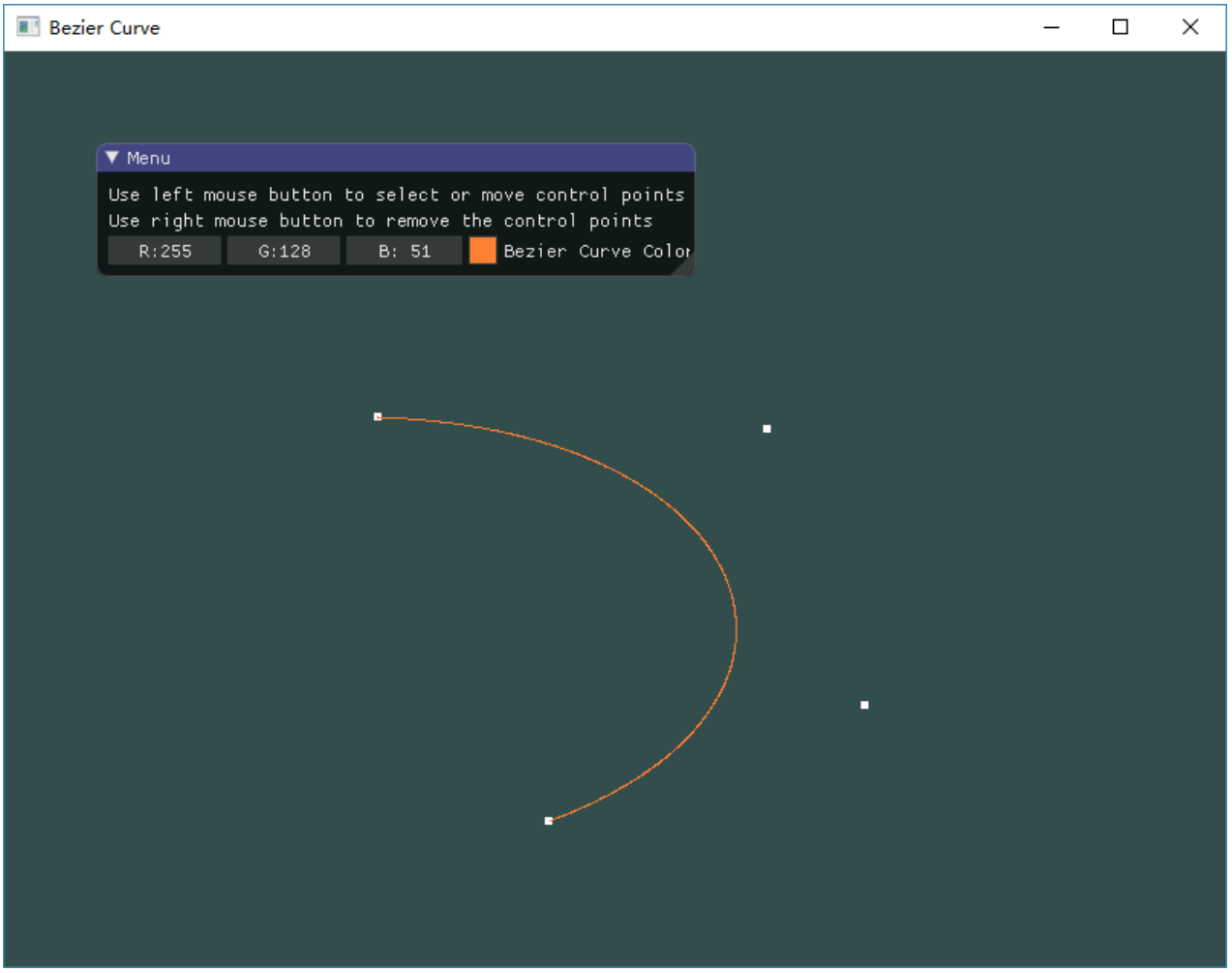
Homework 8 - Bezier Curve

> 点击鼠标左键添加控制点和拖拽已有的控制点
> 点击鼠标右键删除已有的控制点

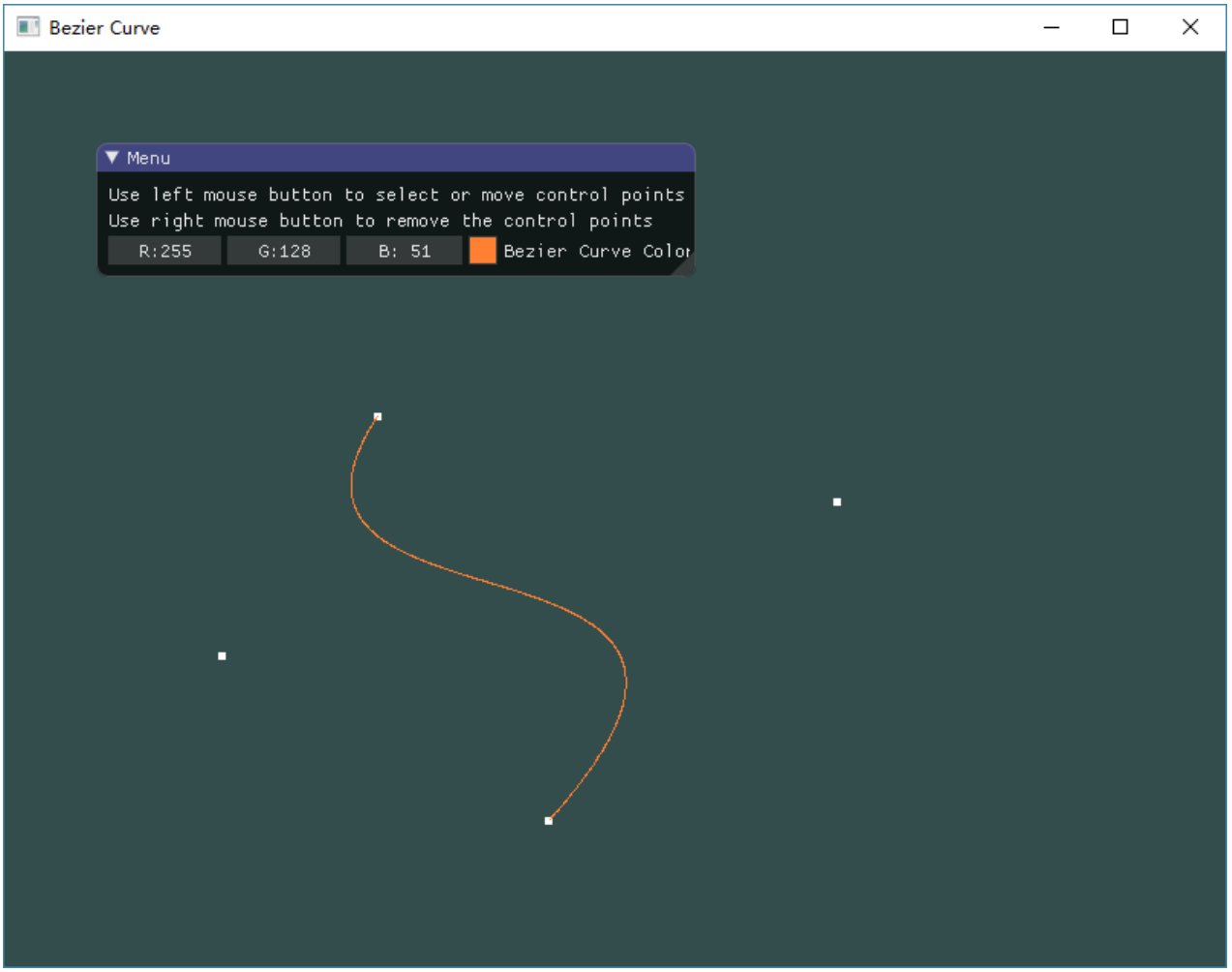
Basic

实验截图

1. 用户能在工具屏幕上画4个点（使用鼠标点击），然后工具会根据这4个点拟合出一条Bezier Curve（按照画点的顺序）



2. 实时调整 4 个点的位置



算法实现

1. Bezier Curve 的渲染

Bezier Curve 的实现只是一条公式的问题。

$$Q(t)=\sum_{i}^{3}P_iB_{i,3}(t)=P_0B_{0,3}(t)+P_1B_{1,3}(t)+P_2B_{2,3}(t)+P_3B_{3,3}(t),\quad t\in[0,1]$$
$$=(1-t)^3P_0+3t(1-t)^2P_1+3t^2(1-t)P_2+t^3P_3$$

因为每改变一次控制点都需要对曲线进行一次新的渲染，所以为了提高渲染效率，我决定把计算 Bezier Curve 的过程放在 shader 里面去做，**这样的话点的数据只是需要传入 t 的值即可**， 每次渲染改变的只是 4 个 uniform 变量用来存 4 个控制点的坐标位置。

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

uniform vec3 p0;
uniform vec3 p1;
uniform vec3 p2;
uniform vec3 p3;


void main()
{
   vec3 qt = pow((1-t), 3) * p0 + 3 * t * (1-t) * (1-t) * p1 + 3 * t * t * (1-t) * p2 + t * t * t * p3;
   gl_Position = vec4(qt.x, qt.y, qt.z, 1.0);
}
```

```
// 生成顶点数据 t
float step = 0.001;
vector<float> data;
data.resize(int(1 / step));
for (int i = 0; i < data.size(); ++i) {
    data[i] = i * step;
}
```

2. 控制点的存储

使用一个 size 为 4 的全局的 `vector<glm::vec3> p` 来存储控制点， 如果该控制点仍没有被选择，则默认设置为 (-100, -100, -100) 。使用 GLFW 注册监听鼠标点击的回掉函数 `mouse_button_callback` ，当点击鼠标左键，如果当前控制点没有选满且当前选的点不是控制点，则把当前鼠标点击的点添加到控制点 vector 中。如果当前控制点选满了，则尝试进入拖拽模式。

```
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 (isNeedControlPoints() && !isPointInVector(xpos, ypos)) {
                // add the selected point
                addPoint(xpos, ypos);
#ifdef DEBUG
                cout << "add point" << xpos << "  " << ypos << endl;
#endif // DEBUG
            }
        }

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

对于拖拽模式，由于 GLFW 中监听鼠标输入的回调函数在鼠标按下后只会执行一次，无法进行实时渲染，所以需要一个全局的控制变量 `isLeftButtonPressed` 和 `currPointIter` 来辅助进行实时渲染，**对于控制点坐标的实时改变需要放在 Render Loop 中而不是回调函数中。**

```
// Render Control Points
if (isLeftButtonPressed) {
    double xpos, ypos;
    glfwGetCursorPos(window, &xpos, &ypos);
    if (!isNeedControlPoints()) {
        // record the selected point index
        // 尝试进入控制模式
        currPointIter = findPointCanControlled(xpos, ypos, 180);
        if (currPointIter != p.end())
            *currPointIter = glm::vec3(xpos, ypos, 0.0f);
    }
}
```

在确认一个点是否应该被选中时，因为完全精确点击一个已有的点的坐标是非常困难的，所以使用一个相对的范围来进行选点：**在阈值 threshold 内，离点击点最近的一个控制点将会被选择。** 该逻辑封装在 `findPointCanControlled` 函数内

```
vector<glm::vec3>::iterator findPointCanControlled(const float xpos, const float ypos, const float threshold) {
    // 粗略查找点击范围中是否有可以控制的点
    vector<glm::vec3>::iterator  res = p.end();
    auto dist = [&xpos, &ypos](const vector<glm::vec3>::iterator iter) -> float {
        return pow((xpos - iter->x), 2) + pow((ypos - iter->y), 2);
    };
    for (auto iter = p.begin(); iter != p.end(); ++iter) {
        auto dis = dist(iter);
        if (dis < threshold) {
            if (res == p.end()) { res = iter; }
            else { res = (dist(res) < dis) ? res : iter; }
        }
    }

    return res;
}
```

GLFW 的鼠标拾取的坐标所采用的坐标系和 NDC 中所采用的坐标系有不同，所以在改变 Shader 中 4 个控制点坐标 uniform 变量前需要对坐标进行一个变换处理。

```
auto glfwPos2nocPos = [](const glm::vec3 p) -> glm::vec3 {
    glm::vec3 res;
    res.x = (2 * p.x) / SCR_WIDTH - 1;
    res.y = 1 - (2 * p.y) / SCR_HEIGHT;
    res.z = 0.0f;
    return res;
};
```

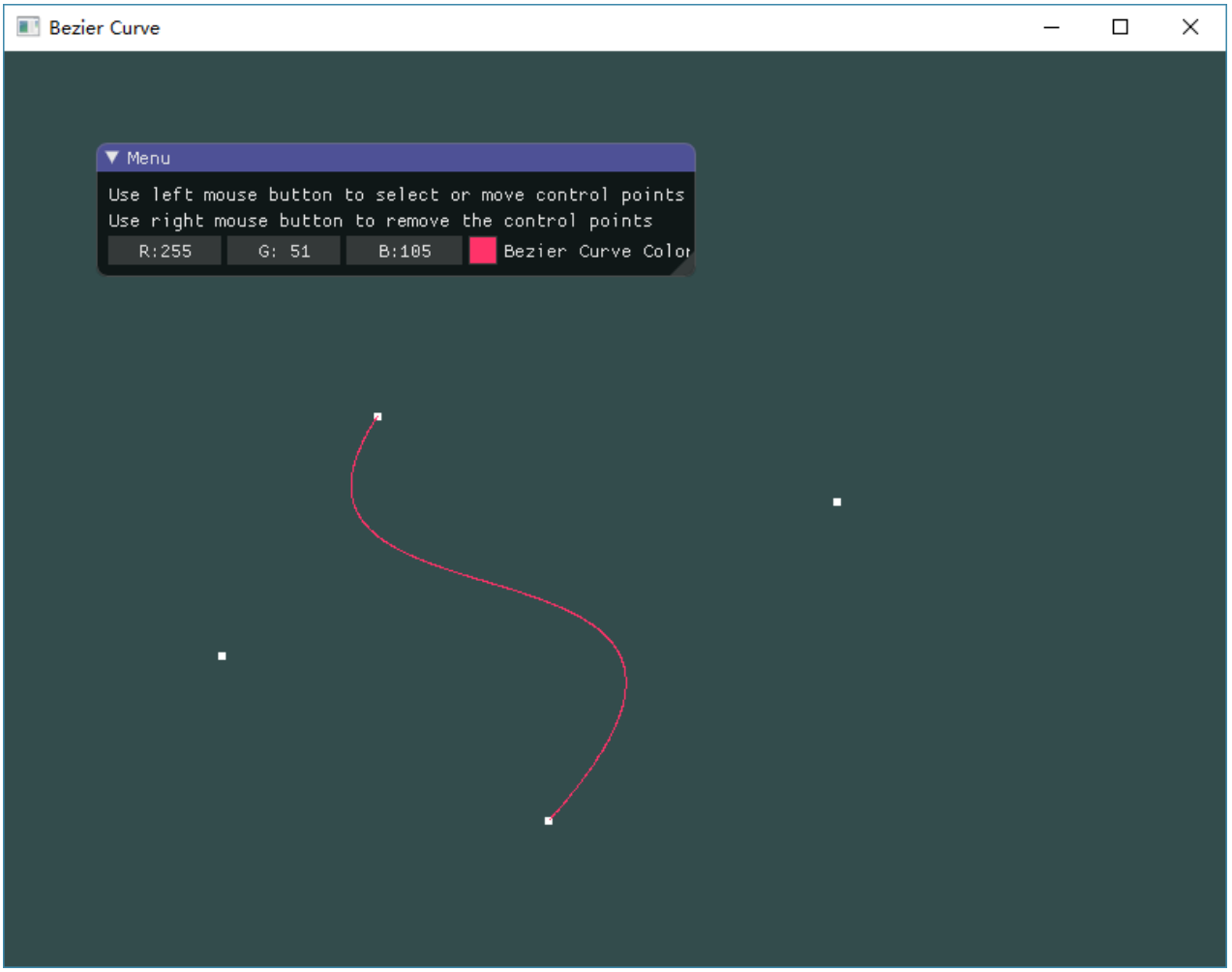
为了防止 ImGui 把我设置的 GLFW 回调函数给覆盖掉，需要把 ImGui 中自己注册的 GLFW 过程给注释掉

```
//if (install_callbacks)
//    ImGui_ImplGlfw_InstallCallbacks(window);
```

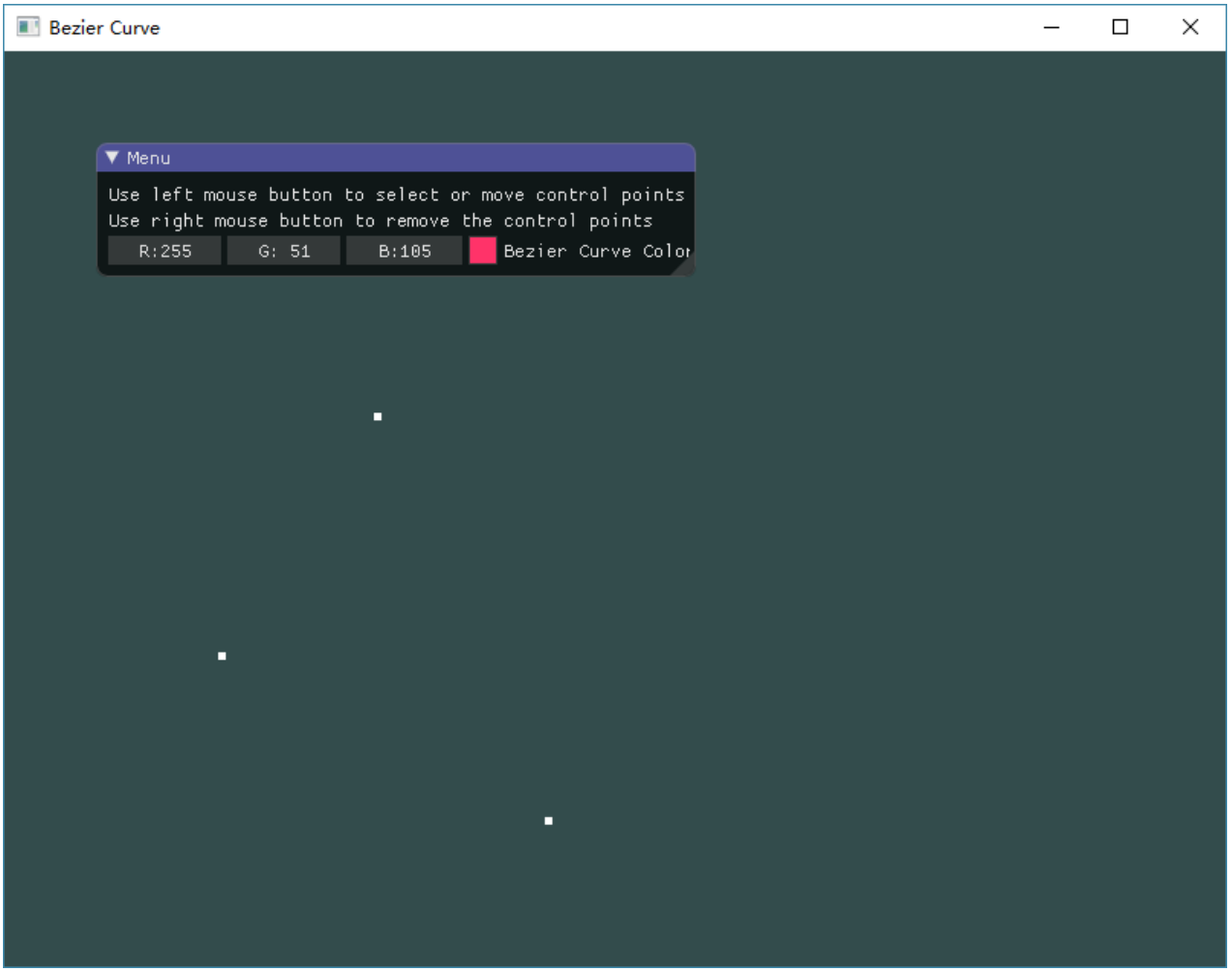
Bonus

实验截图

1. 可以改变 Bezier Curve 的颜色



2. 可以消除某个点。（使用鼠标右键进行删除）



算法实现

1. 在片段着色器里面新增一个 uniform 来控制渲染的颜色即可。然后在 ImGui 中修改一个 `float[3]` 并不断给 uniform 更新。

```
ImGui::ColorEdit3("Bezier Curve Color", col1);
```

2. 使用右键进行点的删除，**在注册的鼠标回调函数里面添加右键点击时候要进行的操作即可。**

```
if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
    // delete one point on the canvas
    auto templter = findPointCanControlled(xpos, ypos, 80);
    if (templter != p.end()) {
        *templter = glm::vec3(-100.0f, -100.0f, -100.0f);
    }
}
```