homework 8

16340118 李杰泓

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。

实现思路

- 添加控制点

添加控制点需要监听鼠标左键点击事件，使用 glfwSetMouseButtonCallback() 函数绑定相应的回调函数，然后判断当前鼠标点击位置是否已经存在控制点，如果没有则在控制点队列末尾添加一个控制点。这里我使用的是 vector<glm::vec3> 这样的数据结构来保存每一个控制点的坐标。这里还要注意的是，我将拿到的鼠标位置从窗口坐标转换为渲染坐标（归一化），方便后面可以直接将值传给顶点着色器。

```
// 鼠标点击回调
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    double xpos, ypos;
    glfwGetCursorPos(window, &xpos, &ypos);
    glm::vec3 mousePos = glfwPos2nocPos(glm::vec3(xpos, ypos, 0.0f));

    if (button == GLFW_MOUSE_BUTTON_LEFT) {
        if (action == GLFW_PRESS) {
            // 点击左键判断是否已有控制点
            isLeftButtonPressed = true;
            if (!isPointInVector(mousePos.x, mousePos.y)) {
                // 若没有则新增一个控制点
                addPoint(mousePos.x, mousePos.y);
            }
        }

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

```
}

// 坐标归一化
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;
};

// 判断点击位置是否已存在控制点
bool isPointInVector(const float xpos, const float ypos) {
    return find(p.begin(), p.end(), glm::vec3(xpos, ypos, 0.0f)) !=
p.end();
}
```

- 删除控制点

删除比较简单，在上面的鼠标点击事件回调函数里面判断是否是右键点击，然后判断控制点队列长度是否大于0，如果是则从中删除最后一个控制点

```
  // 点击右键时删除最后一个控制点
  if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
      if (p.size() > 0) {
          p.pop_back();
      }
 }
```

- 渲染控制点

控制点的渲染主要关注点的坐标。这里我将每一个控制点的3维坐标转为1维的 float 数组，并且将所有点都保存在一个 vector<float> 对象中。设置好顶点着色器属性后，利用 glDrawArrays(GL_POINTS,) 的方式来实现画点。

```
    glBindVertexArray(pVAO);
    glBindBuffer(GL_ARRAY_BUFFER, pVBO);

    auto controlPoints2dataVector = []() -> vector<GLfloat> {
        vector<GLfloat> res;
        res.clear();
        for (int i = 0; i < p.size(); ++i) {
            res.push_back(p[i].x);
            res.push_back(p[i].y);
            res.push_back(p[i].z);
        }
        return res;
    };

    // 将屏幕坐标记录的控制点渲染出来
    auto pointData = controlPoints2dataVector();
```

```
    glBufferData(GL_ARRAY_BUFFER, pointData.size() * sizeof(GLfloat), pointData.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    pointShader.use();
    glPointSize(5.0f);
    glDrawArrays(GL_POINTS, 0, pointData.size() / 3);
    glBindVertexArray(0);
```

- 渲染贝塞尔曲线

由上课知识可以知道，贝塞尔曲线上的每一点由每个控制点与伯恩斯坦基函数相乘得到，公式如下：

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

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

通过 uniform 变量，将控制点个数 n 和控制点坐标数组 p 传到顶点着色器中，在着色器内部分别实现 阶乘函数 fac() 和 伯恩斯坦基函数，从而实现贝塞尔曲线上点的渲染。

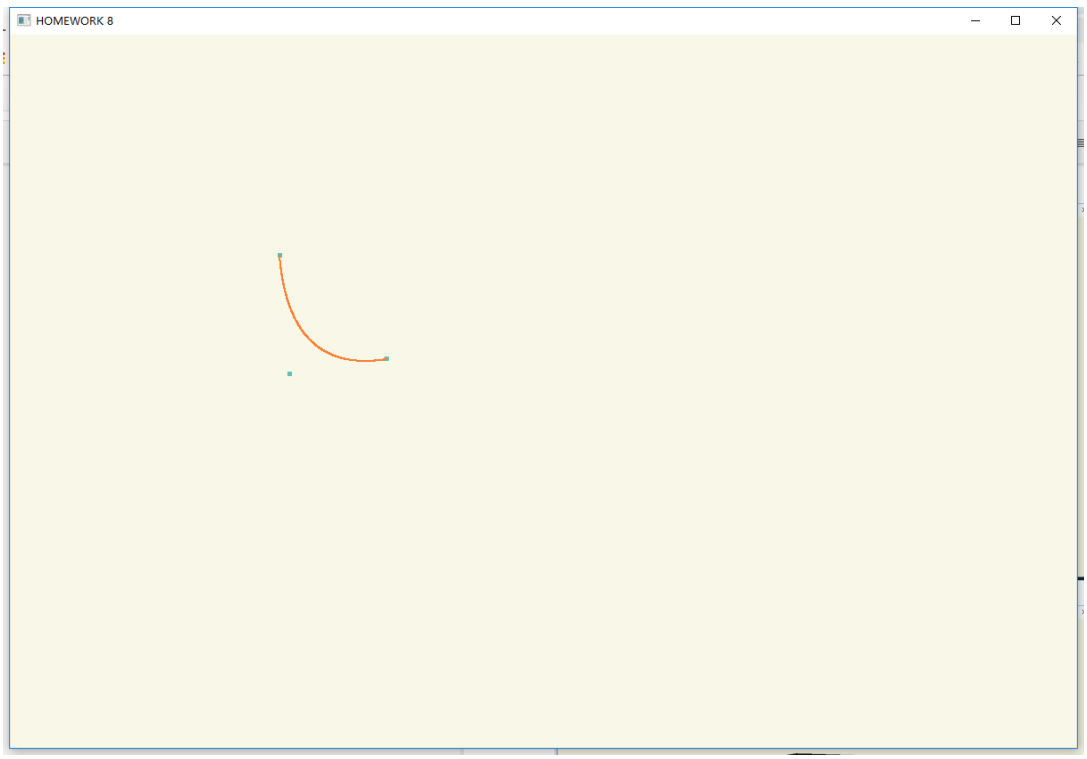
```
#version 330 core
layout (location = 0) in float t;

uniform int n;
uniform vec3 p[];

int fac(int x) {
    int res = 1;
    for(int k = 1; k <= x; k++) {
        res = res * k;
    }
    return res;
}

float B(int i) {
    return fac(n-1) / (fac(i) * fac(n-1-i)) * pow(t, i) * pow((1-t), n-1-i);
}

void main()
{
    vec3 qt = vec3(0.0f, 0.0f, 0.0f);

    for (int i = 0; i < n; ++i) {
        qt += p[i] * B(i);
    }

    gl_Position = vec4(qt.x, qt.y, qt.z, 1.0);
```

```
28  }
```

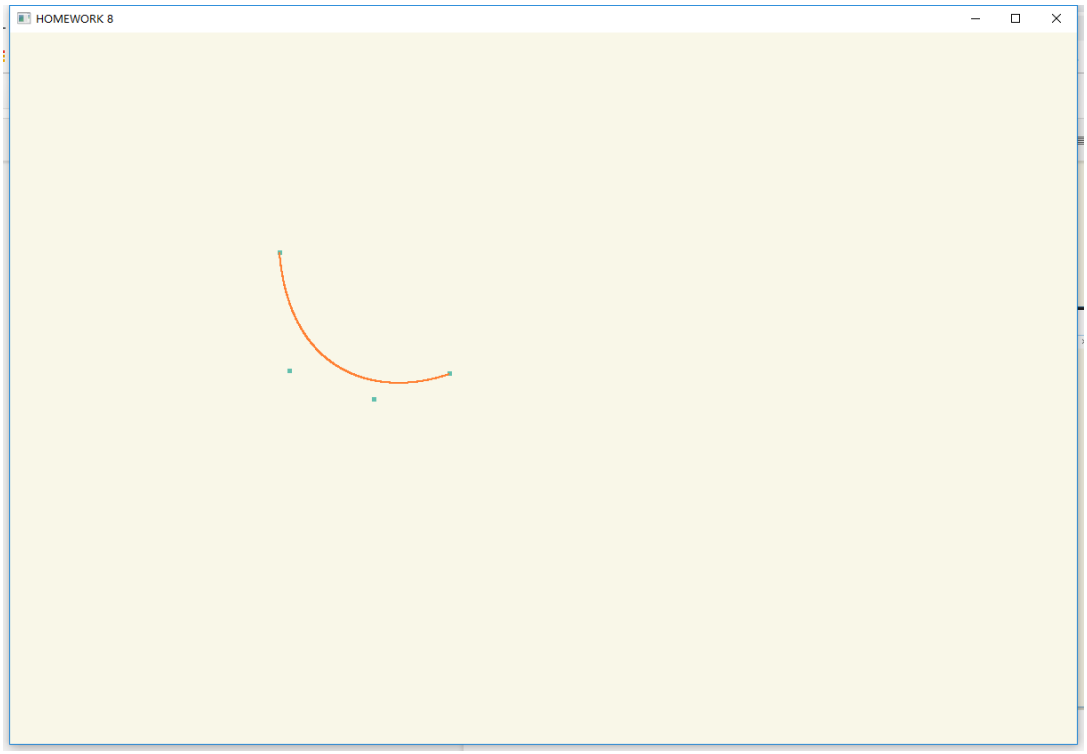
运行效果

两个控制点：

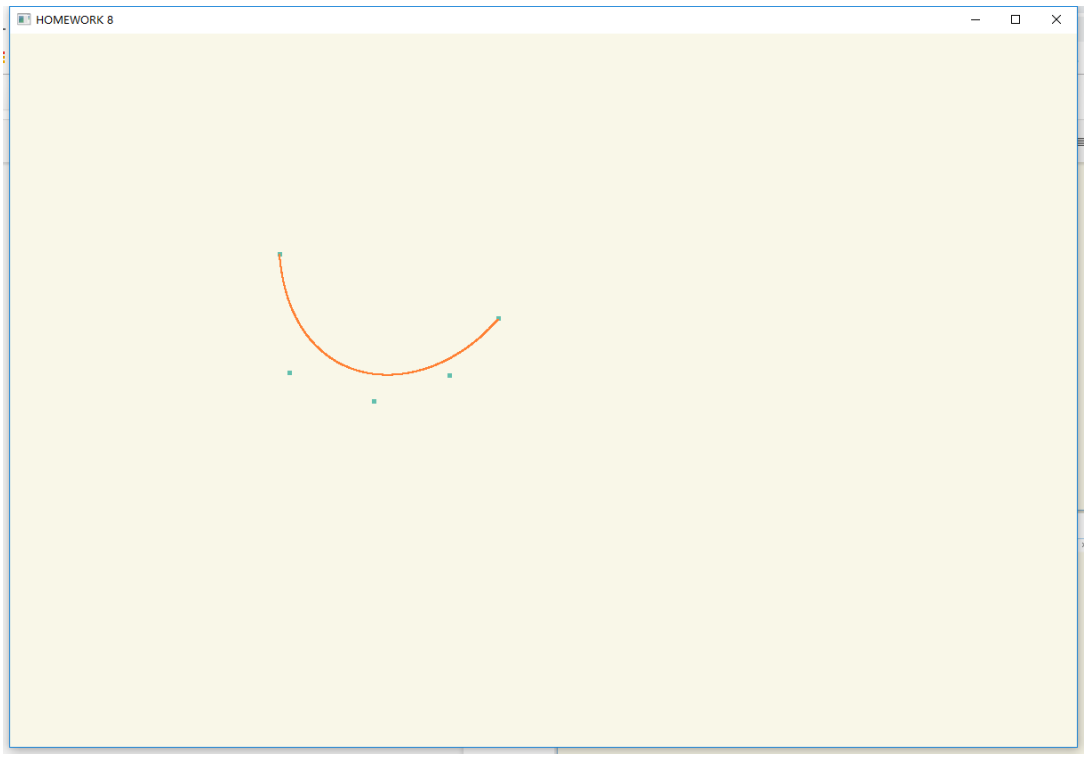
三个控制点：



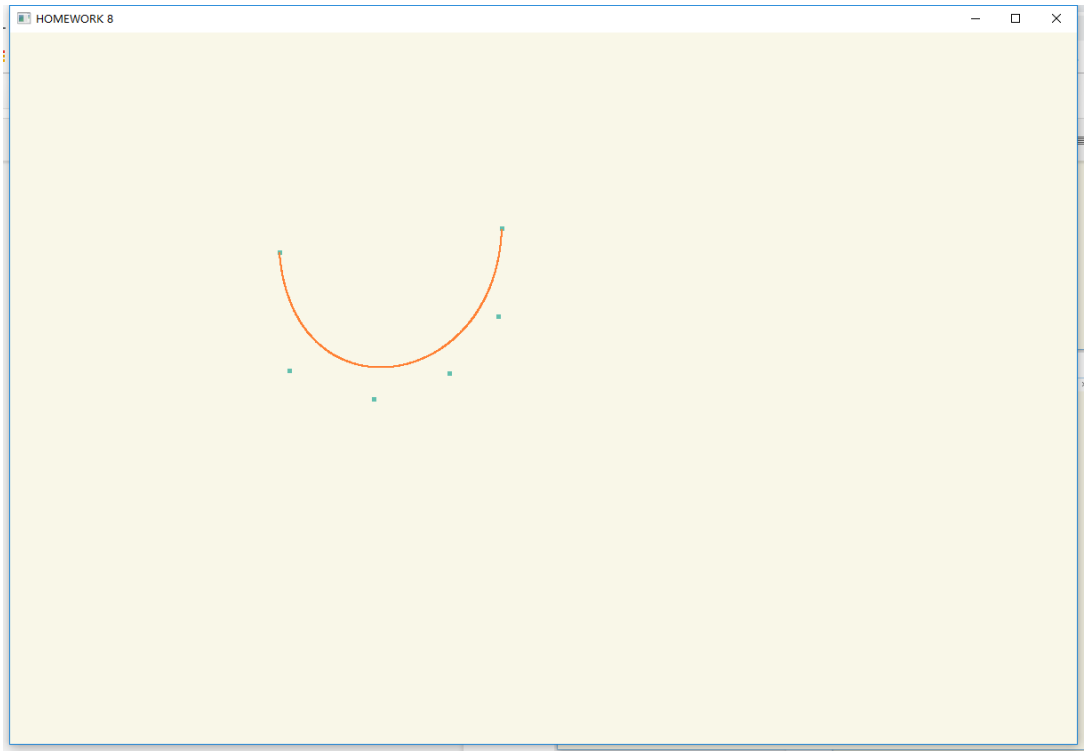
四个控制点：



五个控制点：



六个控制点：



鼠标拖动改变控制点的位置：