CG 第八次作业

软件工程数字媒体 15331101 洪鹏圳

Basic:

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）

（1）通过鼠标点击画 4 个点

首先声明一个 vector 类型的 Points 用来保存这四个点：

```
vector<pair<GLfloat, GLfloat> > Points;//确定Bezier曲线的四个点
```

接着声明一个布尔类型的 isLeftClick 用来保存鼠标是否点击，并声明一个 click_xpos 和 click_ypos 用来保存鼠标每一时刻的位置：

```
bool isLeftClick = false;//判断鼠标是否点击左键
int click_xpos;//鼠标点击的x坐标
int click_ypos;//鼠标点击的y坐标
```

在鼠标移动回调函数中更新 click_xpos 和 click_ypos：

```
//在每次鼠标移动时调用回调函数
void mouse_callback(GLFWwindow* window, double xpos, double ypos) {
    click_xpos = xpos;
    click_ypos = ypos;
}
```

在鼠标点击回调函数中，如果按下鼠标左键则将 isLeftClick 置为 true：

```
//当点击鼠标时的回调函数
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods) {
    if (action == GLFW_PRESS && button == GLFW_MOUSE_BUTTON_LEFT) {
        isLeftClick = true;
        //cout << "leftClick" << endl;
    }
```

在主函数中注册这两个回调函数

```
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
glfwSetMouseButtonCallback(window, mouse_button_callback);//GLFW注册回调函数，当点击鼠标mouse_button_callback函数就会被调用
glfwSetCursorPosCallback(window, mouse_callback);//GLFW注册回调函数，当鼠标一移动mouse_callback函数就会被调用
```

如果 Points 的点数少于 4 个，则每次点击将点击时的鼠标坐标保存到 Points 中：

```
    //左键添加点或者移动点的位置
    if (isLeftClick) {
        //cout << "LeftClick:" << click_xpos << " " << click_ypos << endl;
        //如果点数目少于4添加点
        if (Points.size() < 4) {
            Points.push_back(make_pair(GLfloat(click_xpos), GLfloat(click_ypos)));
        }
```

在绘制点之前，需要将点的坐标转换到视口坐标[-1,1]的范围内：

```
//将x转换成视口坐标系的x
GLfloat x_convert(int x) {
    return (x - screenWidth / 2.0) * 2.0 / screenWidth;
}

//将y转换成视口坐标系的y
GLfloat y_convert(int y) {
    return (screenHeight / 2.0 - y) * 2.0 / screenHeight;
}
```

因为对于窗口坐标来说原点在左上角，对于视口坐标来说原点在中间，因此需要进行上述的转换。

绘制 Points 中已经转换成视口坐标的每个点：

```
    //画点
    for (int i = 0; i < Points.size(); i++) {
        //cout << Points[i].first << " " << Points[i].second << endl;
        drawPoints(x_convert(Points[i].first), y_convert(Points[i].second), pointShader);
    }
```

```
//画点函数
void drawPoints(float fx, float fy, Shader shader) {
    float vertices[] = {
        fx, fy, 0.0f
    };
    unsigned int points_VBO;//顶点缓冲对象
    unsigned int points_VAO;//顶点数组对象
    glGenVertexArrays(1, &points_VAO);//生成一个VAO对象
    glGenBuffers(1, &points_VBO);//生成一个VBO对象
    glBindVertexArray(points_VAO);//绑定VAO
    //把顶点数组复制到缓冲中供OpengGL使用
    glBindBuffer(GL_ARRAY_BUFFER, points_VBO);//把新创建的缓冲VBO绑定到GL_ARRAY_BUFFER目标上
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);//把之前定义的顶点数据points_vertices复制到缓冲的内存中

    //链接顶点属性
    //位置属性，值为0
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);//解析顶点数据
    glEnableVertexAttribArray(0);
    shader.use();//激活着色器程序对象
    glBindVertexArray(points_VAO);//绑定VAO
    glPointSize(5);
    glDrawArrays(GL_POINTS, 0, 1);//绘制图元
    //glBindVertexArray(0);
    glDeleteVertexArrays(1, &points_VAO);
    glDeleteBuffers(1, &points_VBO);
}
```

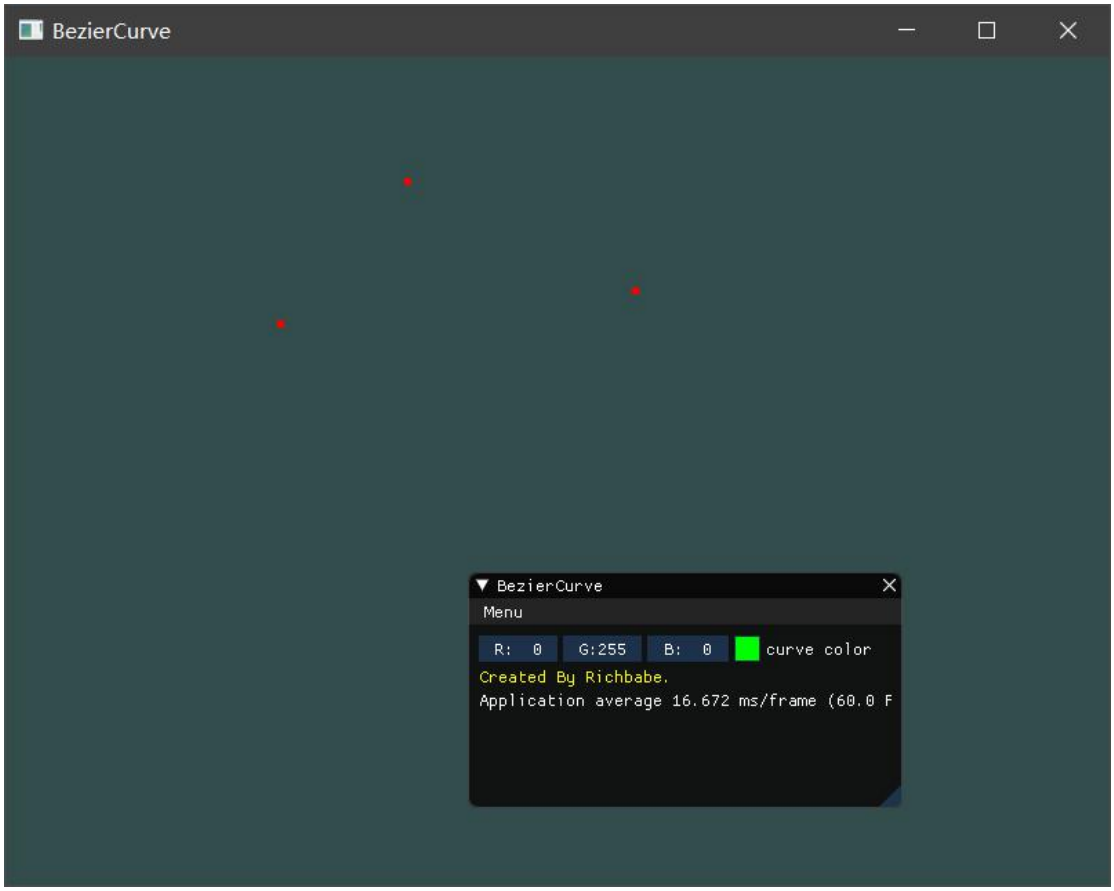
我在点的片段着色器中将点的颜色设置成了红色，区分下面的曲线的颜色：

```
//点片段着色器源代码
#version 330 core
out vec4 FragColor;

//uniform vec3 pointColor;//从顶点着色器传来的输入变量(名称相同、类型相同)

void main()
{
    FragColor = vec4(1.0f, 0.0f, 0.0f, 1.0f);
}
```

运行效果：




（2）画 Bezier Curve 曲线

Bezier Curve，即为贝塞尔曲线。它是一些曲线几何的总称。我们经常使用到的 Bezier Curve 曲线有三种，分别是 Linear Bezier Curve（一次曲线），Quadratic Bezier Curve（二次曲线），Cubic Bezier Curve（三次曲线）。因为要求用 4 个点来拟合一条 Bezier Curve 曲线，因此我选用 Cubic Bezier Curve，其计算公式如下：

曲线的参数形式为：

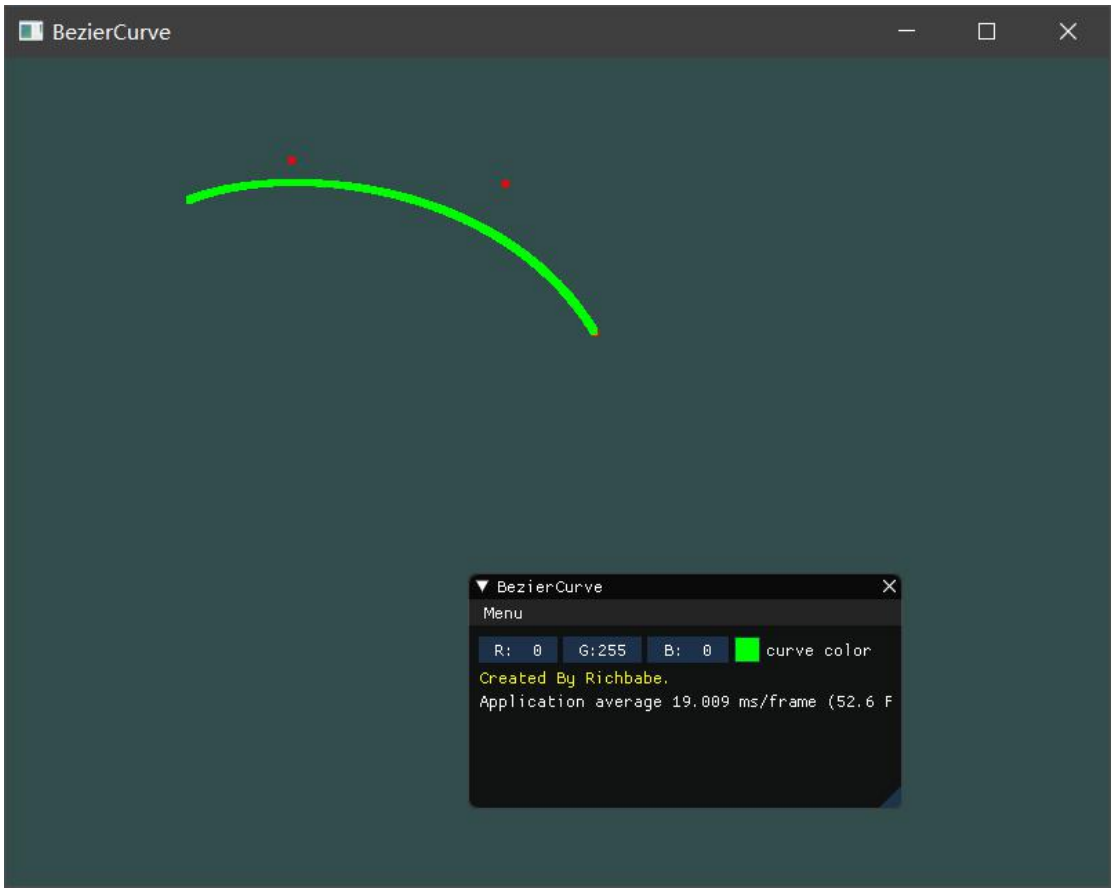
$$B(t) = P_0(1-t)^3 + 3P_1 t(1-t)^2 + 3P_2 t^2(1-t) + P_3 t^3, t \in [0,1]$$

其中 B(t)为曲线上的点，P0、P1、P2、P3 为确定曲线的四个点，t 为[0,1]范围内的浮点数，t 的值取得越小，曲线连续性越高，因此我们可以构造一个循环使得 t 从 0 每次只增加 0.001 直到 t 等于 1，在这个循环内计算曲线上的每个点，通过画点函数便可以画出曲线：

```
//画贝塞尔曲线函数
void drawCurve(Shader shader) {
    for (GLfloat t = 0; t <= 1.0; t += 0.001) {
        //计算曲线上每个点的x坐标和y坐标
        GLfloat x = Points[0].first * pow(1.0f - t, 3) + 3 * Points[1].first * t * pow(1.0f - t, 2)
            + 3 * Points[2].first * t * t * (1.0f - t) + Points[3].first * pow(t, 3);
        GLfloat y = Points[0].second * pow(1.0f - t, 3) + 3 * Points[1].second * t * pow(1.0f - t, 2)
            + 3 * Points[2].second * t * t * (1.0f - t) + Points[3].second * pow(t, 3);

        //画曲线
        drawPoints(x_convert(x), y_convert(y), shader);
    }
}
```

运行效果：




2. 用户画完第一条 Bezier Curve 之后，可以调整 4 个点的位置。工具会根据调整位置后的点 实时更新曲线的样子。

因为我们绘制点和绘制线的代码都在 While 渲染循环中，因此每次更改 Point 中点的坐标值都会实时更新点的位置和曲线的样子。要使得点的位置能够更新，只需要在画满 4 个点后，给每个点添加一个边长为 50 的可修改位置的正方形区域，每次左键点击该区域，该区域所包含的点的位置将会更新为你鼠标点击的位置：

```
//左键添加点或者移动点的位置
if (isLeftClick) {
    //cout << "LeftClick:" << click_xpos << " " << click_ypos << endl;
    //如果点数目少于4添加点
    if (Points.size() < 4) {
        Points.push_back(make_pair(GLfloat(click_xpos), GLfloat(click_ypos)));
    }
    //如果点数目大于等于4移动点的位置
    else{
        for (int i = 0; i < Points.size(); i++) {
            //拖动作用域在【-50, 50】
            if (Points[i].first <= click_xpos + 50 && Points[i].first >= click_xpos - 50 && Points[i].second <= click_ypos + 50 && Points[i].second >= click_ypos - 50) {
                //更新点的位置
                Points[i].first = GLfloat(click_xpos);
                Points[i].second = GLfloat(click_ypos);
            }
        }
    }
    isLeftClick = false;
}
```

这里需要注意每次在处理完鼠标的左键单击后要将 isLeftClick 置为 false，否则会一直判断为正在鼠标左键单击的状态。

Bonus:

1. 在 GUI 里添加菜单栏，用户可以选择 Bezier Curve 的颜色。

可以在曲线片段着色器中声明一个 uniform 的颜色值，通过在 Imgui 中修改该颜色值实现曲线颜色变换的效果：

```
//曲线片段着色器源代码
#version 330 core
out vec4 FragColor;

uniform vec3 curveColor;//从顶点着色器传来的输入变量(名称相同、类型相同)

void main()
{
    FragColor = vec4(curveColor, 1.0f);
}
```

```
// GUI设置参数
ImGui::ColorEdit3("curve color", (float*)&change_curve_color); // 设置曲线颜色
curveShader.setVec3("curveColor", change_curve_color);
```

2. 用户画点时，可以把画出的某个点消除（消除的方式自定义，并在报告说明）。
我实现了在点上面点击右键将画出的某个点消除，具体做法如下：
（1）声明一个布尔型变量 isRightClick 判断鼠标是否点击右键

```
bool isRightClick = false;//判断鼠标是否点击右键
```

在鼠标点击回调函数中，如果点击了右键，则将 isRightClick 置为 true：

```
//当点击鼠标时的回调函数
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods) {
    if (action == GLFW_PRESS && button == GLFW_MOUSE_BUTTON_LEFT) {
        isLeftClick = true;
        //cout << "leftClick" << endl;
    }
    else if (action == GLFW_PRESS && button == GLFW_MOUSE_BUTTON_RIGHT) {
        isRightClick = true;
        //cout << "rightClick" << endl;
    }
    return;
}
```

（2）在主函数中，如果 isRightClick 为 true，则遍历 Points 看其元素中哪个点被右键点击了，如果该点被右键点击则将该点从 Points 中删除：

```
//右键去除点
if (isRightClick) {
    //cout << "rightClick:" << click_xpos << " " << click_ypos << endl;
    if (Points.size() > 0) {
        for (int i = 0; i < Points.size(); i++) {
            if (Points[i].first <= click_xpos + 3 && Points[i].first >= click_xpos - 3 && Points[i].second <= click_ypos + 3 && Points[i].second >= click_ypos - 3) {
                Points.erase(Points.begin() + i);
                //cout << "delete!" << endl;
            }
        }
    }
    isRightClick = false;
}
```

需要注意的是因为点的位置很小，因此我给点击的坐标加上了-3 到 3 的误差，同时在处理完每次右键点击事件的时候需要记得将 isRightClick 置为 false。

具体的运行效果见演示视频。