Homework 9 - Bezier Curve

16 级数媒 颜承橹 15322244

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 Basic 需要解决三个问题：获取鼠标的点击事件、根据点击事件动态修改绘制点、绘制 Bezier 曲线。

利用 glfwSetMouseButtonCallback(window, click_callback)可以设置鼠标点击后的响应函数，glfwSetCursorPosCallback(window, cursor_pos_callback)可以追踪光标。由以下代码：

```
void cursor_pos_callback(GLFWwindow* window, double x, double y) {
    bool hovered = ImGui::IsWindowHovered(ImGuiHoveredFlags_AnyWindow);
    cursorPosX = x;
    cursorPosY = y;
}

void click_callback(GLFWwindow* window, int button, int action, int mods) {
    bool hovered = ImGui::IsWindowHovered(ImGuiHoveredFlags_AnyWindow);
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS && !hovered) {
        glm::vec3 clickPos = standardize(cursorPosX, cursorPosY);
        Points.push_back(clickPos);
    }
    if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
        if (!Points.empty())
            Points.pop_back();
    }
}
```

可以实时获取光标坐标。坐标用 vector<glm::vec3>Points 存储。窗口内（不包括 GUI）点击左键后，会将当前坐标存储在 Points 中，hover 用于判断光标是否在 GUI 内。点击右键时，若 Points 不为空，则会将最后一个元素删除，从而达到删除点的效果。

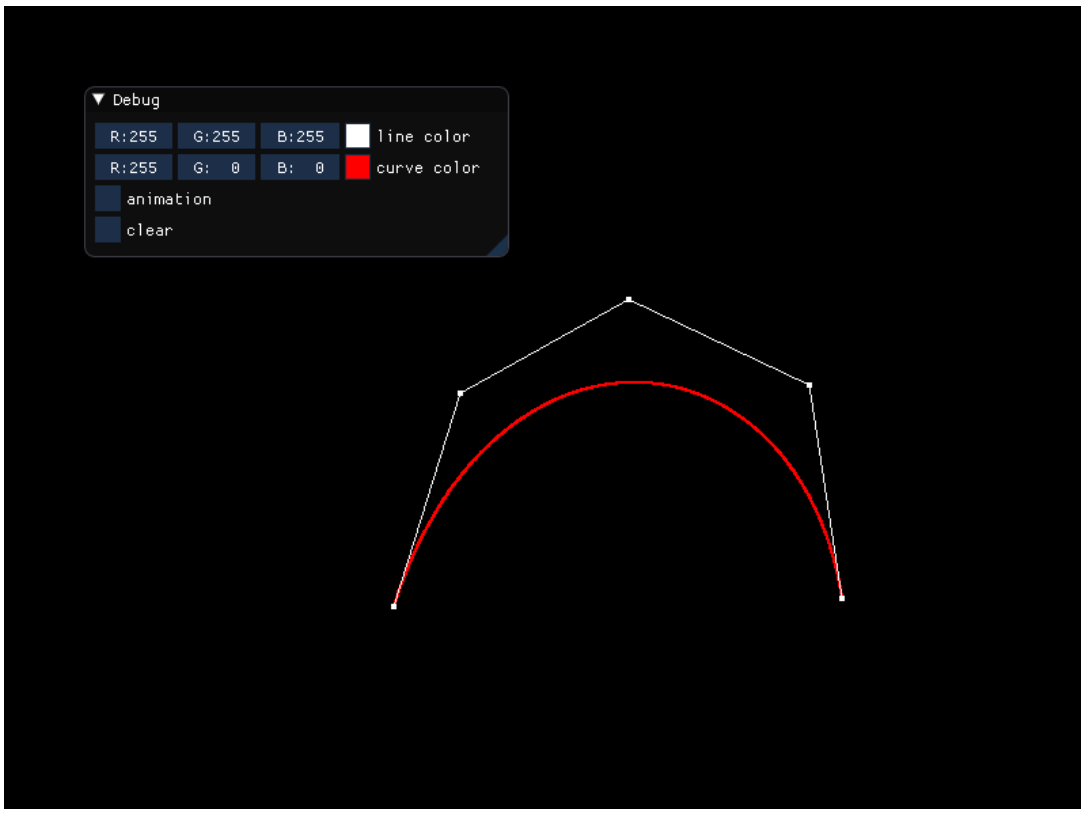
Bezier 曲线的绘制使用了公式：

$$B(t)=\sum_{i=0}^{n}\binom{n}{i}Pi(1-t)^{n-i}t^{i}=\binom{n}{0}P0(1-t)^{n}t^{0}+\binom{n}{1}P1(1-t)^{n-1}t^{1}+\ldots+\binom{n}{n-1}Pn-1(1-t)^{1}t^{n-1}+\binom{n}{n}Pn(1-t)^{0}t^{n},t\in[0,1]$$

Bezier 函数会根据 Points 动态生成曲线，并返回一个点集。代码如下：

```
vector<glm::vec3> Bezier() {
    vector<glm::vec3> result;
    int n = Points.size() - 1;
    for (float t = 0; t <= 1; t += 0.001) {
        glm::vec3 r(0,0,0);
        for (int i = 0; i <= n; i++) {
            float f1 = pow(t, i);
            float f2 = pow(1 - t, n - i);
            float cn = 1;
            for (int j = 0; j < i; j++) {
                cn = cn * (n-j) / (j+1);
            }
            r = r + cn * f1 * f2 * Points[i];
        }
        result.push_back(r);
    }
    return result;
}
```

最后呈现的结果如下:



Bonus：

1. 可以动态地呈现 Bezier 曲线的生成过程。

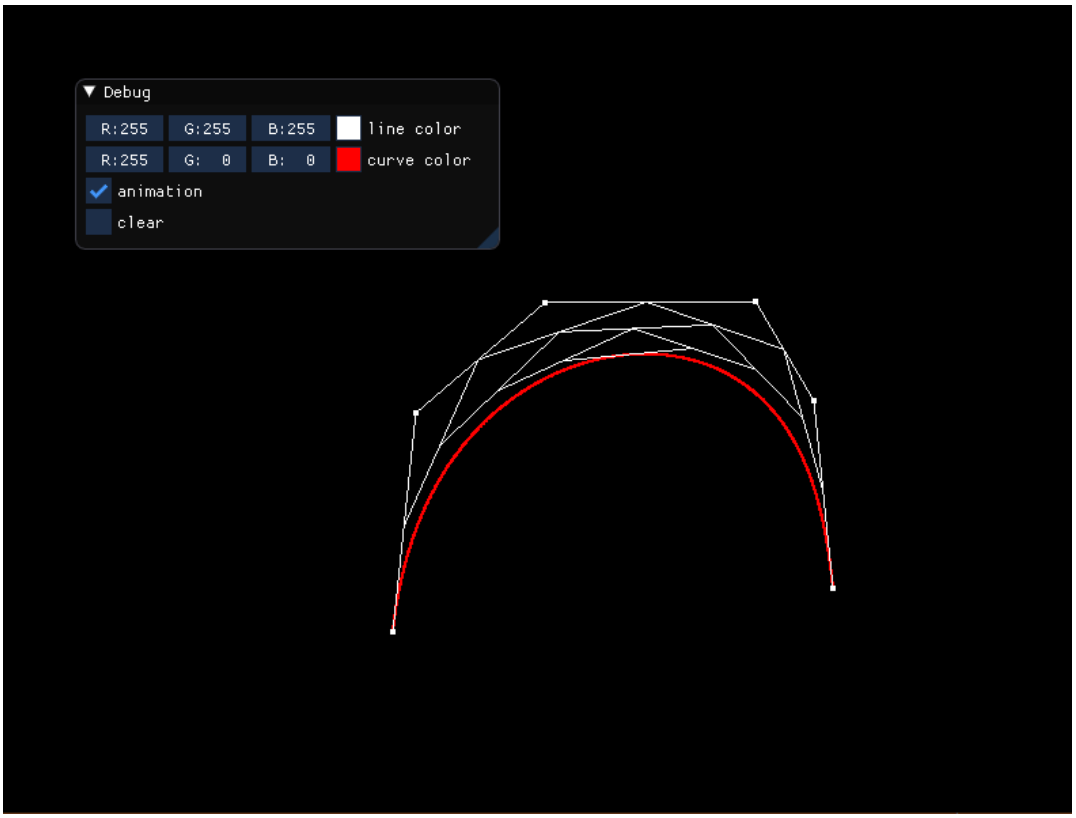
该部分代码如下：

```
153     if (animation) {
154         if (flag) {
155             lastFrame = glfwGetTime();
156             flag = false;
157         }
158         else {
159             currentTime = glfwGetTime();
160             float deltaTime = (currentTime - lastFrame) / 3;
161             if (deltaTime >= 1) {
162                 //animation = false;
163                 flag = true;
164             }
165             else {
166                 vector<glm::vec3> TempPoints = Points;
167                 vector<glm::vec3> TempPoints2;
168                 int count = 5;
169                 while (TempPoints.size() > 2) {
170                     for (int i = 0; i < TempPoints.size() - 1; i ++) {
171                         glm::vec3 P0 = (1 - deltaTime) * TempPoints[i] + deltaTime * TempPoints[i + 1];
172                         TempPoints2.push_back(P0);
173                     }
174                     for (int i = 0; i < TempPoints2.size() - 1; i++) {
175                         float line[] = {
176                             TempPoints2[i].x, TempPoints2[i].y,TempPoints2[i].z, color[0], color[1], color[2],
177                             TempPoints2[i+1].x, TempPoints2[i + 1].y, TempPoints2[i + 1].z, color[0], color[1], color[2],
178                         };
179 
180                         unsigned int lineVAO2 = count;
181                         count++;
182                         glBindVertexArray(lineVAO2);
183                         glBufferData(GL_ARRAY_BUFFER, sizeof(line), line, GL_STATIC_DRAW);
184                         // position
185                         glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
186                         glEnableVertexAttribArray(0);
187                         //color
188                         glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
189                         glEnableVertexAttribArray(1);
190                         glBindVertexArray(lineVAO2);
191 
192                         glDrawArrays(GL_LINE_STRIP, 0, 2);
193                     }
194                     TempPoints.clear();
195                     TempPoints = TempPoints2;
196                     TempPoints2.clear();
197                 }
198             }
```

时间元素：该动画以 3 秒为周期，deltaTime 参数表示动画的某一时刻，为方便后续计算，将它除的值以 3，使其在 0~1 之间变化。

绘图元素：先使用 TempPoints 复制 Points，假设 Points 中有 n 个点，则每次循环后，都会产生 n-1 个新点并存入 TempPoints2 中，点的生成见第 171 行代码。然后每次选 TempPoints2 中相邻的两点绘制直线，共有 n-2 条新的直线。接着再用 TempPoints2 覆盖 TempPoints，重新循环直到 TempPoints.size()小于等于 2。

最终得到的效果如下：


Debug
R:255
G:255
B:255
line color
R:255
G: 0
B: 0
curve color
animation
clear