**实验报告：基于OpenGL的Bezier曲线建模与可视化**

**​引言**

在计算机图形学中，​Bezier曲线是描述平滑曲线的重要工具，广泛应用于游戏开发、动画设计、工业建模等领域。本次实验我通过OpenGL实现了一个交互式Bezier曲线绘制工具，使用鼠标点击生成控制点，右键完成输入后实时生成曲线。相比其他同学可能使用的Python或Web工具，我选择了OpenGL作为底层图形库，利用其高性能渲染能力，并结合Premake构建工具管理项目依赖，代码已开源至GitHub仓库，真正做到“开箱即用”。

​**一、Bezier曲线建模思想与应用领域**

​**1. 核心思想**

Bezier曲线通过控制点定义曲线形状，其数学本质是伯恩斯坦多项式的线性组合。曲线的起点和终点始终与第一个和最后一个控制点重合，而中间控制点通过权重影响曲线走向。实验中我采用递归德卡斯特里奥算法生成曲线点，逐层插值逼近最终曲线。

​**2. 应用场景**

* ​动画路径：如角色移动轨迹、摄像机运动路径。
* ​工业设计：汽车曲面、电子产品外观建模。
* ​字体渲染：TrueType字体轮廓的平滑描述。
* ​游戏开发：武器弹道、地形生成等。

​**二、实验实现与核心代码**

​**1. 工具与框架**

* ​OpenGL 4.6：核心渲染API，负责顶点着色与图元绘制。
* ​GLFW/GLAD：处理窗口创建、鼠标输入及OpenGL函数加载。
* ​Premake：跨平台项目构建工具，一键生成VS或Makefile项目，避免手动配置依赖库。

​**2. 核心代码解析**

​**① 鼠标交互与坐标转换**

void mouseButtonCallback(GLFWwindow\* window, int button, int action, int mods) {

if (button == GLFW\_MOUSE\_BUTTON\_LEFT && action == GLFW\_PRESS) {

*// 将屏幕坐标转换为NDC（OpenGL坐标系）*

float ndcX = (2.0f \* xpos) / width - 1.0f;

float ndcY = 1.0f - (2.0f \* ypos) / height;

coordinates.push\_back(ndcX); *// 存储控制点*

}

else if (button == GLFW\_MOUSE\_BUTTON\_RIGHT) acceptInput = false;

}

功能：左键点击记录控制点，右键结束输入。通过glfwGetCursorPos获取屏幕坐标，并转换为OpenGL的标准化设备坐标（NDC）。

​**② Bezier点生成算法**

std::vector<float> generateBezierPoint(const std::vector<float>& controlPoints, float t) {

if (controlPoints.size() > 3) {

std::vector<float> newPoints;

for (int i = 0; i < controlPoints.size() - 3; i += 3) {

*// 线性插值：P\_i = (1-t)\*P\_i + t\*P\_{i+1}*

float x = (1 - t) \* controlPoints[i] + t \* controlPoints[i + 3];

newPoints.push\_back(x); *// 同理计算y,z*

}

return generateBezierPoint(newPoints, t); *// 递归直至只剩一个点*

}

return controlPoints;

}

功能：递归实现德卡斯特里奥算法，每次迭代将控制点数量减一，最终得到曲线上某一点的坐标。

​**③ OpenGL渲染逻辑**

while (!glfwWindowShouldClose(window)) {

glClear(GL\_COLOR\_BUFFER\_BIT);

if (!acceptInput) {

std::vector<float> bezierPoints = generateBezierPoints(coordinates);

glBufferData(GL\_ARRAY\_BUFFER, bezierPoints.size() \* sizeof(float), bezierPoints.data(), GL\_STATIC\_DRAW);

glDrawArrays(GL\_LINE\_STRIP, 0, bezierPoints.size() / 3); *// 绘制曲线*

}

glfwSwapBuffers(window);

}

功能：右键点击后，生成100个曲线点并绑定到VBO，通过GL\_LINE\_STRIP绘制连续线段。

​**三、实验思考与个人心得**

​**1. 为什么选择OpenGL？**

* ​性能优势：直接操作GPU，适合实时渲染。
* ​跨平台性：代码可无缝运行于Windows、Linux、macOS。
* ​学习价值：深入理解图形管线、着色器、缓冲区等底层概念。

​**2. 遇到的坑与解决方案**

* ​初版曲线断裂：误用GL\_LINE而非GL\_LINE\_STRIP，导致点间无连接。
* ​顶点数计算错误：最初用控制点数而非贝塞尔点数渲染，仅显示零散点。
* ​递归越界：在generateBezierPoint中错误遍历控制点，引发内存崩溃。

​**3. Premake的便捷性**

传统CMake配置复杂，而Premake通过Lua脚本一键生成项目文件，依赖管理极其简单：

bash

premake5 vs2022 *# 生成Visual Studio工程*

premake5 gmake2 *# 生成Makefile（Linux/macOS）*

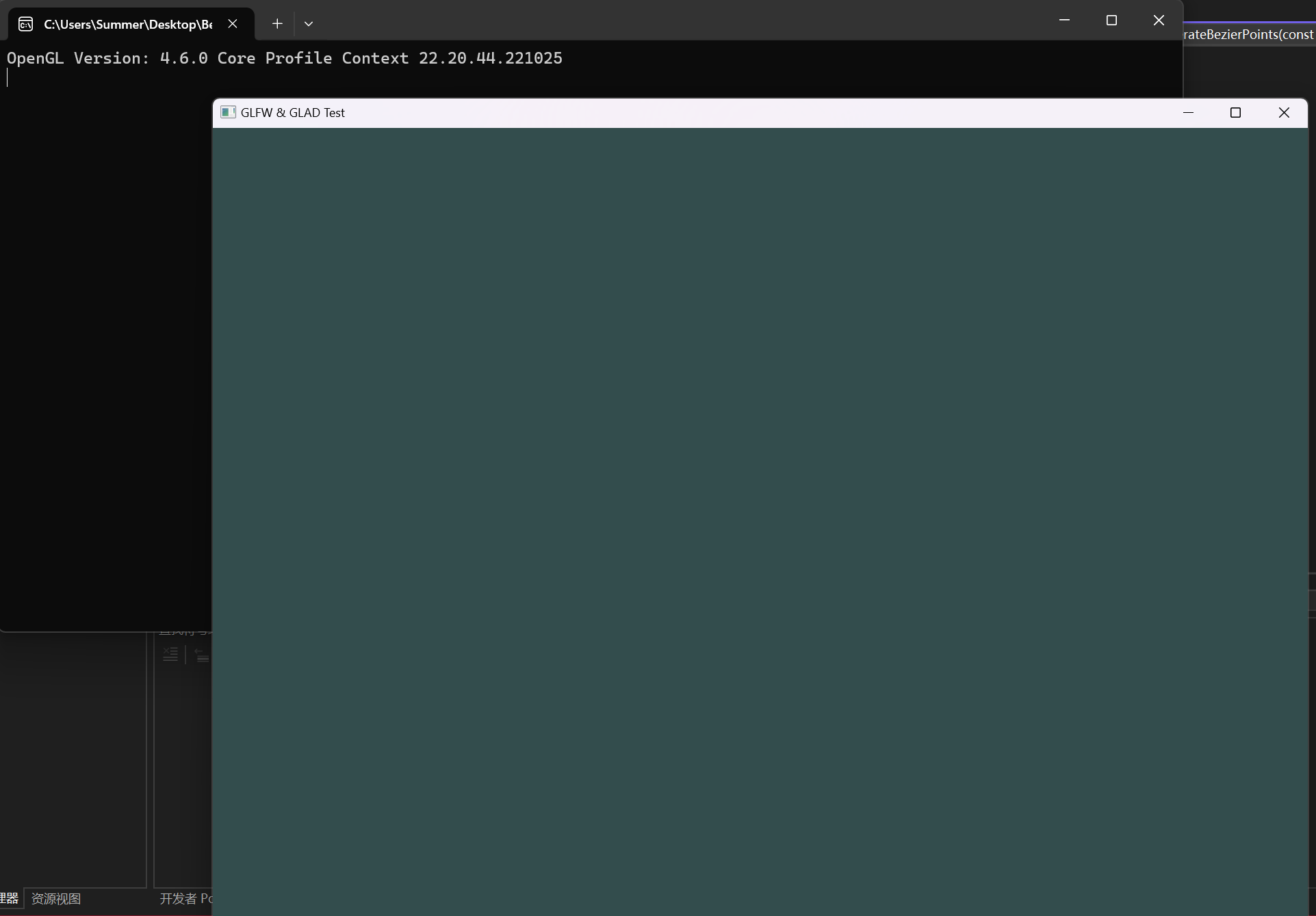
​**4. 鼠标交互的直观性**

通过实时坐标转换，用户可直接在窗口中“所见即所得”地调整曲线，相比输入坐标值的传统方式，体验更流畅。

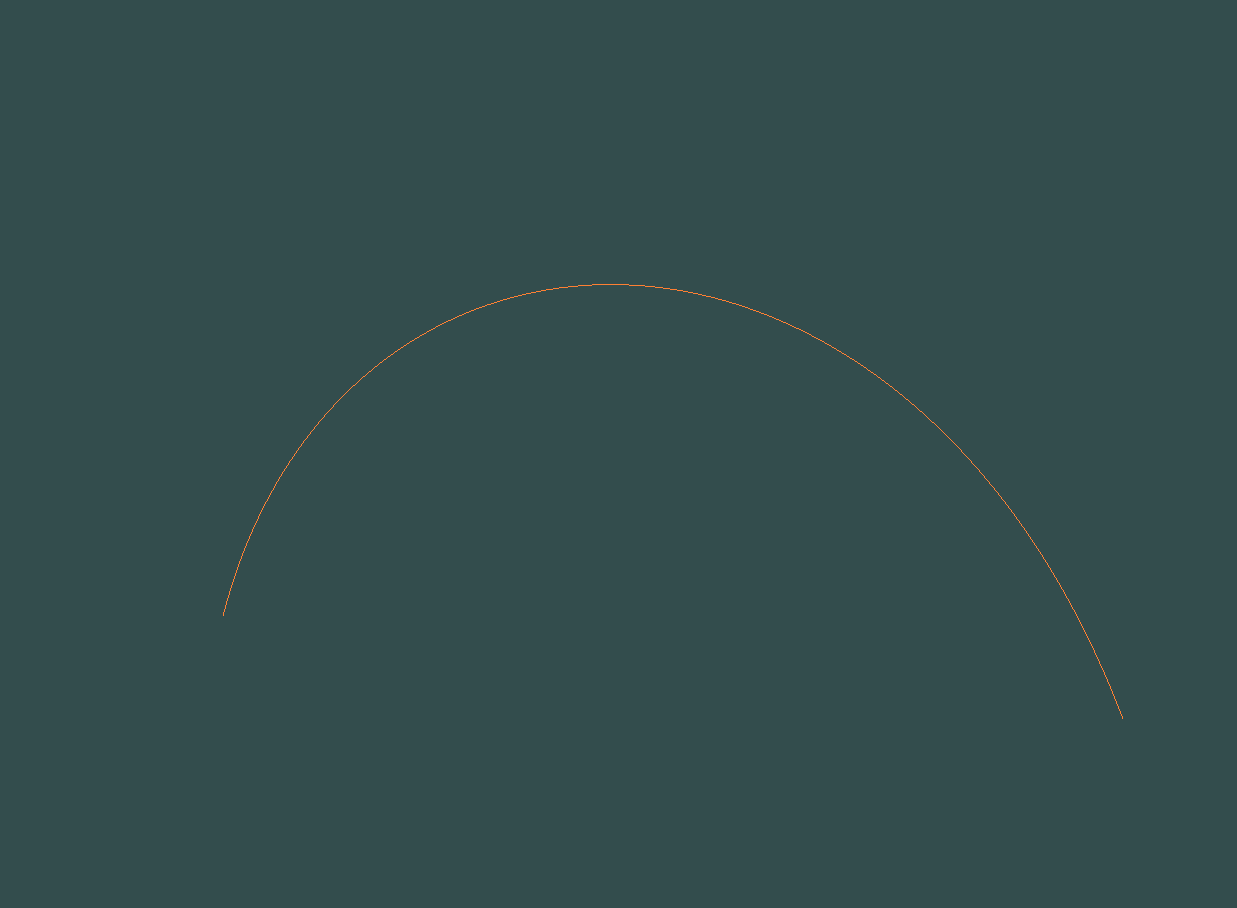
​**四、项目开源与使用说明**

* ​GitHub仓库：https://github.com/Firefly-Star/Bezier
* ​操作指南：左键添加控制点，右键生成曲线，ESC退出。

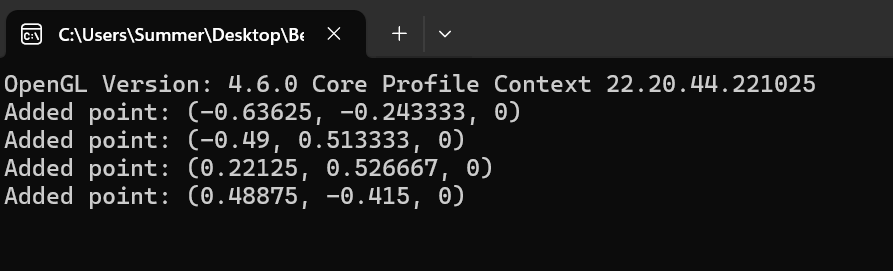
**五、代码运行截图**



初始窗口



Bezier曲线的绘制



控制点的NDC坐标输出

​**五、总结**

这次实验让我深刻体会到OpenGL在图形编程中的强大与灵活。通过亲手实现Bezier曲线生成算法，不仅巩固了递归与插值的数学知识，更掌握了现代图形API的核心开发流程。未来计划扩展为Bezier曲面建模，并加入曲线编辑功能（如拖动控制点），进一步提升工具的实用性。