《可视计算与交互概论》期末大作业报告

基于 OpenGL 与 ImGui 的高性能多功能数字绘画系统

姓名：[方山川]　学号：[2400011013]
专业：[信科]

2026 年 1 月 11 日

1　大作业选题 (Project Overview)

本项目选题属于“可视计算与交互”中的**图形渲染与交互系统开发**专题。本项目实现了一个高性能的数字绘画程序，旨在探索如何平衡实时图形交互中的“矢量灵活性”与“大规模渲染性能”。

在数字绘画中，随着笔画（Vector Strokes）数量增加，逐帧遍历数以万计的顶点会导致严重的计算负担。本项目通过实现 **离屏渲染（Off-screen Rendering）**与 **动态纹理烘焙（Bake）**技术，有效地解决了这一性能瓶颈，并实现了多种具有艺术表现力的复杂纹理笔刷。

2　系统架构 (System Architecture)

本项目采用模块化设计，将软件划分为四个核心层次，以保证系统的可扩展性与稳定性：

1. **Renderer (渲染层)**：管理 OpenGL 资源，负责 FBO (Frame Buffer Object) 的状态切换及纹理附件的维护。
2. **CanvasLogic (逻辑层)**：实现核心算法，包括笔刷采样逻辑、Catmull-Rom 样条插值、几何形体生成及矢量路径切割算法。
3. **AppUI (界面层)**：基于 Dear ImGui 搭建，处理侧边栏、画布布局、主题定制及用户交互响应。
4. **Common (公共层)**：定义全局数据结构（如 `Stroke` 结构体）和常量配置。

3　关键技术实现 (Key Implementation)

3.1　混合渲染与纹理烘焙

系统采用双层渲染机制。底层为一张 RGBA 格式的静态纹理（底片），顶层为实时生成的矢量线段。当用户触发 `Bake` 操作时，系统通过以下公式将矢量点集转化为像素：

$$C_{final} = \alpha_{src} \cdot C_{src} + (1 - \alpha_{src}) \cdot C_{dst} \tag{1}$$

通过离屏渲染技术，将 ImGui 的绘图命令重定向至 FBO，实现了矢量到光栅的无损转化，极大提升了渲染效率。

3.2 纹理盖章算法与笔刷动力学

为了模拟真实笔触（如蜡笔、水彩），本项目摒弃了传统的线段连接法，采用了**纹理盖章（Texture Stamping）**技术。系统在笔迹路径上进行高频采样，并引入随机性（Jitter）：

- **旋转抖动**：每个采样点随机生成 $\theta \in [0, 2\pi]$ 的旋转矩阵。
- **位置与缩放抖动**：模拟真实手绘中力度的不均匀性。

3.3 精确矢量切割 (Precise Eraser)

不同于普通的橡皮擦，本项目实现了“精确切割”功能。当橡皮擦经过某条笔划时，算法会遍历路径点，在碰撞点处断开连接，并将原笔划重构为两个独立的 `Stroke` 对象，保证了矢量编辑的灵活性。

4 实验结果展示 (Results)

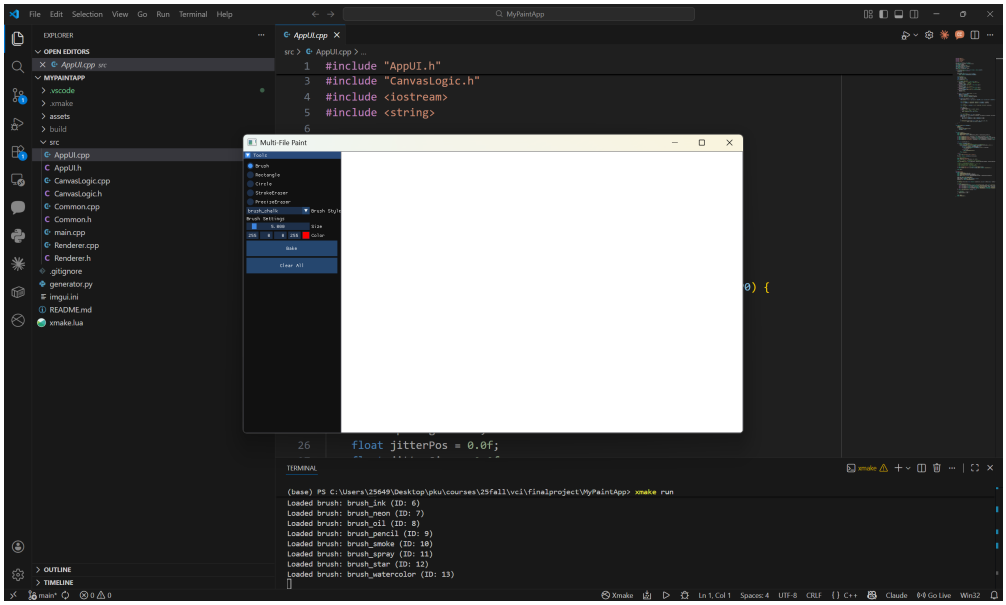

图 1: 软件主界面：展示了现代化的 UI 布局与侧边栏工具箱

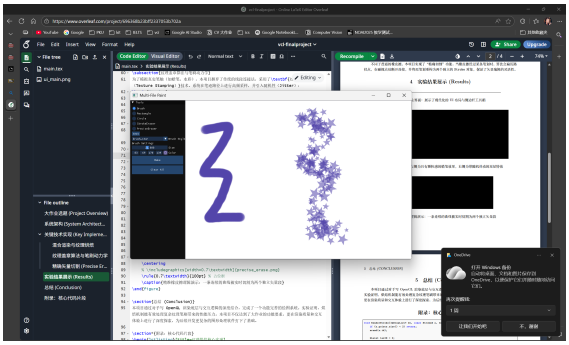

图 2: 笔刷效果展示：左侧为具有颗粒感的蜡笔效果，右侧为带随机抖动的星星特效

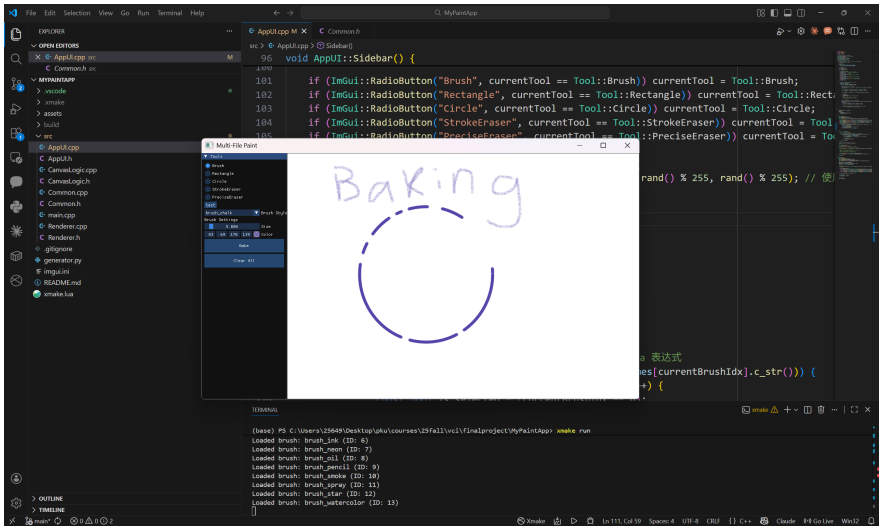

图 3: 精准橡皮擦逻辑演示：一条连续的曲线被实时切割为两个独立矢量段

5 总结 (Conclusion)

本项目通过对手写 OpenGL 渲染底层与交互逻辑的深度结合，完成了一个功能完善的绘图系统。实验证明，烘焙机制能有效处理复杂纹理笔刷带来的性能压力。本项目不仅达到了大作业的功能要求，更在渲染质量和交互体验上进行了深度探索，为后续开发更复杂的图形处理软件打下了基础。

附录：核心代码片段

```
void RenderStroke(ImDrawList* dl, const Stroke& s, ImVec2 canvasP0) {
    if (s.points.size() < 2) return;
    srand(s.id);

    GLuint texID = 0;
    if (Renderer::brushTextures.count(s.brushName))
        texID = Renderer::brushTextures[s.brushName];
```

```
    // 特殊配置参数
    float spacing = 0.1f;
    float jitterPos = 0.0f;
    float jitterSize = 0.0f;
    bool randomRotation = true;

    // 根据笔刷名字设置特殊表现
    if (s.brushName.find("star") != std::string::npos) {
        spacing = 0.8f;      // 星星要稀疏一点，才像撒出来的
        jitterPos = 2.0f;    // 坐标乱跳
        jitterSize = 0.5f;   // 大小不一
    } else if (s.brushName.find("smoke") != std::string::npos) {
        spacing = 0.2f;
        jitterSize = 1.2f;   // 烟雾忽大忽小
    } else if (s.brushName.find("grass") != std::string::npos) {
        spacing = 0.05f;
        randomRotation = false; // 草丛通常随鼠标方向，而不是乱转
    } else if (s.brushName.find("glitch") != std::string::npos) {
        spacing = 0.3f;
        jitterPos = 1.5f;
    }

    for (size_t i = 0; i < s.points.size() - 1; i++) {
        ImVec2 p1 = s.points[i];
        ImVec2 p2 = s.points[i+1];
        float dist = CanvasLogic::GetDistance(p1, p2);
        float step = fmax(1.0f, s.thickness * spacing);

        for (float d = 0; d < dist; d += step) {
            float t = d / dist;
            ImVec2 basePos = { p1.x + (p2.x-p1.x)*t + canvasP0.x, p1.y + (p2.y-p1.y)*t + canvasP0.y };

            // 1. 坐标抖动
            float px = basePos.x + (rand()%100 - 50)/50.0f * s.thickness * jitterPos;
            float py = basePos.y + (rand()%100 - 50)/50.0f * s.thickness * jitterPos;

            // 2. 随机大小
            float currentSize = s.thickness * (1.0f + (rand()%100 - 50)/50.0f * jitterSize);

            // 3. 旋转逻辑
            float angle = 0;
            if (randomRotation) {
                angle = (rand() % 360) * (3.1415f / 180.0f);
            } else {
                // 如果不随机旋转，就跟随鼠标移动方向
                angle = atan2f(p2.y - p1.y, p2.x - p1.x);
            }

            // 4. 准备绘制顶点
            float cosA = cosf(angle); float sinA = sinf(angle);
            ImVec2 corners[4] = {{-currentSize,-currentSize}, {currentSize,-currentSize}, {currentSize,
                currentSize}, {-currentSize,currentSize}};
            ImVec2 q[4];
            for (int n=0; n<4; n++) {
                q[n].x = px + (corners[n].x * cosA - corners[n].y * sinA);
```

```
                q[n].y = py + (corners[n].x * sinA + corners[n].y * cosA);
            }

            dl->AddImageQuad((ImTextureID)(intptr_t)texID, q[0], q[1], q[2], q[3],
                             {0,0}, {1,0}, {1,1}, {0,1}, s.color);
        }
    }
}
```

纹理烘焙核心实现