2D图形绘制与图像处理实验报告

一、实验概述

Lab1主要实现了一系列2D图形绘制与图像处理算法，包括图像抖动、图像滤波、图像修复、直线绘制、三角形填充、图像超采样以及贝塞尔曲线绘制等功能。通过这些实验，深入理解了数字图像处理的基本原理和图形绘制的核心算法。

二、各功能模块实现思路

1. 图像抖动（Image Dithering）

- **随机均匀抖动（DitheringRandomUniform）**
  - 原理：在阈值判断前给像素值添加一个[-0.5, 0.5]的随机扰动
  - 实现：使用随机数生成器产生均匀分布的扰动值，添加到每个像素的RGB分量后再做阈值判断
- **蓝噪声抖动（DitheringRandomBlueNoise）**
  - 原理：使用预先计算的蓝噪声图案替代随机噪声，产生更自然的抖动效果
  - 实现：通过使用噪声图像，将噪声值标准化后添加到像素值
- **有序抖动（DitheringOrdered）**
  - 原理：使用3x3的抖动矩阵，将每个像素扩展为3x3区域，根据矩阵值决定每个子像素的颜色
  - 实现：将输入像素值与抖动矩阵归一化后的值比较，生成3倍分辨率的输出图像
- **误差扩散抖动（DitheringErrorDiffuse）**
  - 原理：将当前像素的量化误差按比例扩散到周围未处理的像素，使整体误差最小化
  - 实现：采用Floyd-Steinberg扩散算法，将7/16误差传到右邻像素，3/16传到左下，5/16传到下邻，1/16传到右下

2. 图像滤波（Image Filtering）

- **模糊滤波（Blur）**
  - 原理：使用3x3的均值卷积核对图像进行卷积操作，平滑图像细节
  - 实现：对每个像素，计算其3x3邻域内像素的加权平均（权重均为1/9），边界采用 clamping 处理

```
float kernel[3][3] = {
    {1.0f / 9, 1.0f / 9, 1.0f / 9},
    {1.0f / 9, 1.0f / 9, 1.0f / 9},
    {1.0f / 9, 1.0f / 9, 1.0f / 9}
};
```

- **边缘检测（Edge）**
  - 原理：使用Sobel算子（x方向和y方向）计算图像梯度，梯度大小表示边缘强度
  - 实现：分别用x和y方向的卷积核计算梯度，再通过 sqrt(dx²+dy²) 计算最终边缘强度

3. 图像修复（Image Inpainting）

- 原理：基于泊松方程的图像融合方法，通过求解偏微分方程将前景图像自然地融合到背景图像中
- 实现：
  i. 设置边界条件，使前景图像边界与背景图像对应位置一致
  ii. 使用Jacobi迭代法求解泊松方程（8000次迭代）
  iii. 将求解结果与前景图像相加，得到融合后的图像

```
// 边界条件设置
g[y * width] = inputBack.At(offset.x + 0, offset.y + y) - inputFront.At(0, y);
g[y * width + width - 1] = inputBack.At(offset.x + width - 1, offset.y + y) - inputFront.At(widt

// Jacobi迭代
g[y * width + x] = (g[(y - 1) * width + x] + g[(y + 1) * width + x] + g[y * width + x - 1] + g[y
```

4. 直线绘制（DrawLine）

- 原理：实现Bresenham直线算法，通过整数运算高效绘制直线
- 实现：
  i. 特殊处理水平和垂直直线
  ii. 对于斜线，根据斜率绝对值分为两类（|k|≤1和|k|>1）
  iii. 使用误差累积变量决定下一个像素位置，避免浮点运算

5. 三角形填充（DrawTriangleFilled）

- 原理：采用扫描线算法，通过判断点是否在三角形内部来填充三角形
- 实现：
  i. 计算三角形的 bounding box，限定扫描范围
  ii. 对每个像素，使用叉积法判断是否在三角形内部

iii. 内部像素设置为指定颜色

```
// 叉积法判断点是否在三角形内
int a1=(p1.x-p0.x)*(y-p0.y)-(p1.y-p0.y)*(x-p0.x);
int a2=(p2.x-p1.x)*(y-p1.y)-(p2.y-p1.y)*(x-p1.x);
int a3=(p0.x-p2.x)*(y-p2.y)-(p0.y-p2.y)*(x-p2.x);
return (a1>=0&&a2>=0&&a3>=0)||(a1<=0&&a2<=0&&a3<=0);
```

6. 图像超采样（Supersample）

- 原理：通过在每个输出像素内进行多次采样并平均，减少锯齿效应，提高图像质量
- 实现：
  i. 根据采样率（rate）在每个输出像素内生成rate×rate个采样点
  ii. 对每个采样点使用双线性插值计算颜色
  iii. 平均所有采样点颜色作为输出像素值

7. 贝塞尔曲线（Bezier Curve）

- 原理：使用德卡斯特里奥（de Casteljau）算法计算贝塞尔曲线上的点
- 实现：
  i. 递归地对控制点进行线性插值
  ii. 取不同的参数t计算曲线上的点
  iii. 通过绘制线段连接这些点近似贝塞尔曲线

```
// 德卡斯特里奥算法
std::vector<glm::vec2> points_copy = std::vector<glm::vec2>(points.begin(), points.end());
for(int i=1;i<points_copy.size();++i)
    for(int j=0;j<points_copy.size()-i;++j)
        points_copy[j]=(1.0f-t)*points_copy[j]+t*points_copy[j+1];
return points_copy[0];
```

三、实验效果图

1. 图像抖动

- 随机均匀抖动

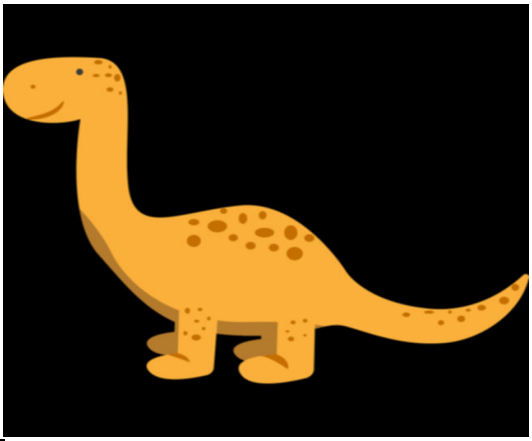
- 蓝噪声

- 有序抖动
(缩放后)

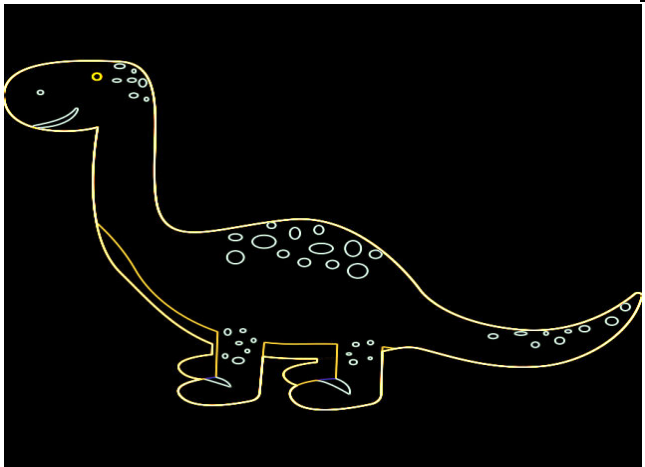
- error difuse

2. 图像滤波效果

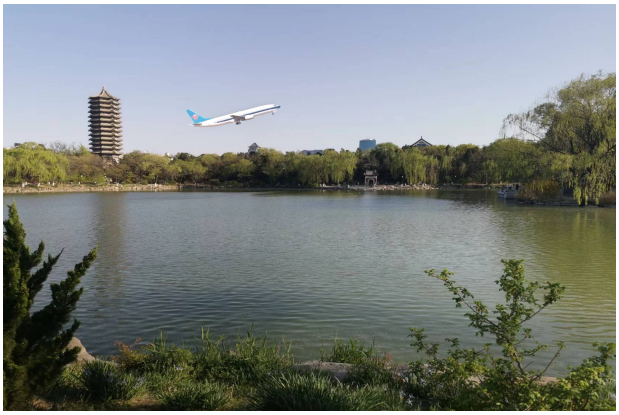
- 模糊滤波



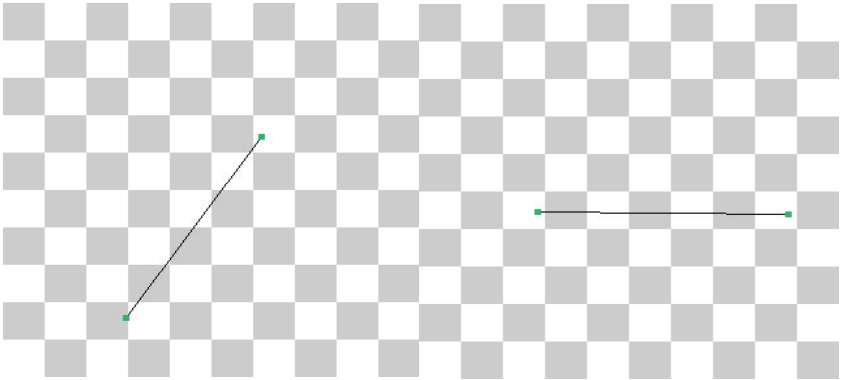
- 边缘检测



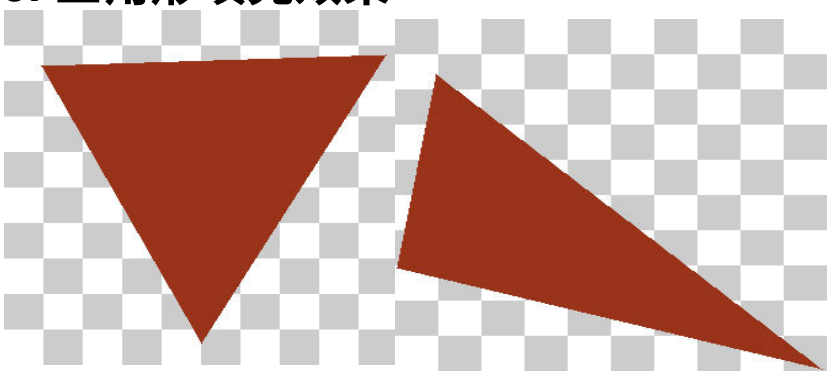
3. 图像修复效果



4. 直线绘制效果



5. 三角形填充效果



6. 超采样效果

- 未采样

- rate=5

- rate =10

7. 贝塞尔曲线效果