学号	姓名	论文规范性（10）	问题分析与调研（30）	方案创新性（20）	实验结果分析与讨论（40）	结课论文总成绩（100）
21301096	黄一圃	8	21	17	36	82

缺乏相关工作梳理

北京交通大学

计算机图形学期末论文

基于 FXAA 技术的抗锯齿研究

A Study on Anti-Aliasing Techniques Utilizing FXAA Technology

学　　院：软件学院

专　　业：软件工程

学生姓名：黄一圃

学　　号：21301096

指导教师：

北京交通大学

2024 年 6 月 23 日

摘要

抗锯齿技术在计算机图形学中扮演着重要角色，旨在减少图像中由于分辨率不足而产生的锯齿效应。本文聚焦于一种流行的抗锯齿算法——快速近似抗锯齿（Fast Approximate Anti-Aliasing, FXAA），对其原理、实现方法以及应用效果进行深入研究和分析。通过实验比较不同场景下 FXAA 开启与关闭的渲染效果，探讨其优势和不足，最终得出结论。

关键词：抗锯齿 FXAA

目　　录

1 引言

在计算机图形学中，锯齿效应是由于图像分辨率不足而导致的边缘不平滑现象。这种效应严重影响图像质量，尤其在高对比度场景中尤为明显。为了减少锯齿效应，研究人员提出了多种抗锯齿技术，如多重采样抗锯齿（MSAA）、超采样抗锯齿（SSAA）和快速近似抗锯齿（FXAA）。本文选择 FXAA 作为研究对象，旨在了解其技术特点、实现方式及实际应用效果。

2 FXAA 算法

FXAA 由 NVIDIA 的 Timothy Lottes 于 2011 年首次提出。它是一种后处理算法，不需要多次采样，因此计算速度快，适用于实时应用[1]。与 MSAA 和 SSAA 等抗锯齿算法相比，FXAA 的计算和内存开销较低，对硬件要求较低。且 FXAA 可以处理各种分辨率和图像类型，无需调整参数即可获得较好的抗锯齿效果[2]。

FXAA 算法的具体通常需要以下几个步骤：

1. 对比度计算：首先，我们从输入的非线性 RGB 图像中计算亮度（luminance）。通常，我们使用以下公式来计算亮度：

$$L = 0.299 * R + 0.587 * G + 0.114 * B$$

2. 其中，R, G, B 分别为像素的红、绿、蓝分量，计算后的效果如图一所示。接下来对于每一个采样点，计算其周围五个像素点最高亮度与最低亮度的差值，作为该采样点的对比度。当对比度高于阈值时，则认为该采样点需要进行锯齿处理。

图 1 亮度计算结果示例

3. 混合系数计算：根据目标像素与其邻居之间的平均亮度差异来计算混合权重，为使结果更加精确，需对对角线上的四个点进行采样并计算亮度值。因为对角像素距离中心像素比较远，所以计算平均亮度值时的权重会略微低一些，因此计

算时各采样点的权重如图 2 所示。

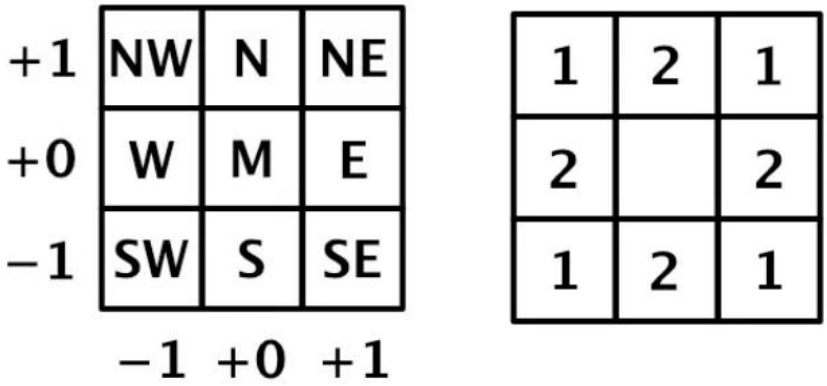


图 2 采样点位置与权重

4. 计算混合方向：这一步需要确定进行混合计算的方向，锯齿边界通常不会是刚好水平或者垂直的，需要寻找一个最接近的方向。因此通过计算水平与竖直方向的变化程度并比较以确定混合方向，变化程度计算如下：

$$\text{Vertical} = 2 * \text{abs}(\text{N} + \text{S} - 2\text{M}) + \text{abs}(\text{NE} + \text{SE} - 2\text{E}) + \text{abs}(\text{NW} + \text{SW} - 2\text{W})$$
$$\text{Horizon} = 2 * \text{abs}(\text{E} + \text{W} - 2\text{M}) + \text{abs}(\text{NE} + \text{NW} - 2\text{N}) + \text{abs}(\text{SE} + \text{SW} - 2\text{S})$$

5. 搜索与混合：在确定亮度梯度的方向后，沿着边界两侧的方向进行搜索，直到找到锯齿边界，判断边界的方式为计算两侧亮度值的差，是否与当前的亮度变化梯度值符合，若不符合，则为一个边界。若在指定的搜索步数中未找到边界，则将边界距离设置为默认值。若两边界的距离分别为 PDis 与 NDis，则混合系数为：

$$\text{EdgeBlend} = \begin{cases} 0.5 - \text{PDis}/(\text{PDis} + \text{NDis}), & \text{PDis} < \text{NDis} \\ 0.5 - \text{NDis}/(\text{PDis} + \text{NDis}), & \text{PDis} > \text{NDis} \end{cases}$$

最终通过混合系数对颜色进行混合即可得到如图 3 抗锯齿处理后的图像。

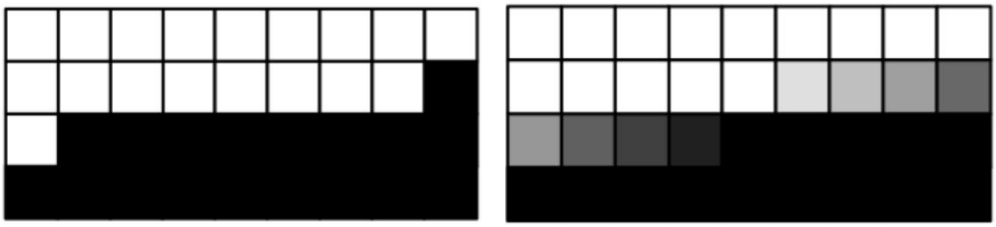


图 3 抗锯齿处理前后效果对比

3 实验设计

3.1 实验目的

本实验将通过编写程序实现 FXAA，以评估 FXAA 算法在实际应用中的效果。通过对比不同场景下启用和未启用 FXAA 的图像质量，验证其在减少锯齿效应方面的有效性。

3.2 实验环境

操作系统	Windows11
显卡	NVIDIA GeForce RTX 2060
开发语言	C++
图形库	OpenGL
开发工具	Visual Studio Code

3.3 实验步骤

3.3.1 测试场景搭建

首先，启动 OpenGL 环境的初始化流程。通过调用 OpenGL 的窗口系统 API，创建一个具有特定尺寸、位置和标题的图形渲染窗口。随后，为该窗口初始化一个 OpenGL 渲染上下文，接着设置视口的大小以匹配窗口的物理尺寸。为了准备后续的渲染工作，使用 glClearColor 和 glClearDepth 函数设置清除颜色和深度缓冲区的初始值。

基础图形的绘制不再赘述，接下来主要是复杂模型的加载与初始化。选择较为复杂的模型，利用模型加载库 Assimp 加载模型的顶点坐标、纹理坐标、法线等数据，并将其存储到 OpenGL 可识别的数据结构中。根据加载的模型数据，创建 OpenGL 的顶点数组对象（VAO）、顶点缓冲区对象（VBO）和纹理对象等，为后续的渲染工作做好准备。

完成模型的加载与初始化后，进行相机和光源的设置。定义相机的位置、朝向和视野角等参数，以确定观察场景的角度和范围。根据相机的参数和视口大小，计算透视投影矩阵，并将其传递给着色器程序。

最后，进行基础图像的渲染。将用于基础渲染的顶点着色器和片段着色器绑定到 OpenGL 的渲染管线上。这些着色器程序将负责处理模型的顶点变换、光照计算和像素着色等任务。随后，调用 OpenGL 的绘制函数，根据加载的模型数据和设置的相机参数，

将模型绘制到屏幕上。在绘制完成后，将渲染得到的图像保存到磁盘上，以供后续与启用 FXAA 后的图像进行对比。

3.3.2 FXAA 算法实现

在这一过程，首先，针对 FXAA 算法的特性，创建一组着色器，包括顶点着色器和片段着色器。顶点着色器专注于处理场景中的顶点数据，执行必要的变换和投影操作，以确保顶点能够在屏幕空间中被正确绘制。片段着色器则扮演着更为核心的角色，负责在像素级别上应用 FXAA 算法，减少图像中的锯齿现象。

在片段着色器的实现中，主要包括三个关键步骤：边缘检测、子像素处理和像素混合。边缘检测是首要步骤，它通过分析像素间的颜色和亮度差异来识别图像中的边缘区域。这些边缘区域往往是锯齿现象最为明显的部分，因此是 FXAA 算法需要重点关注的对象。

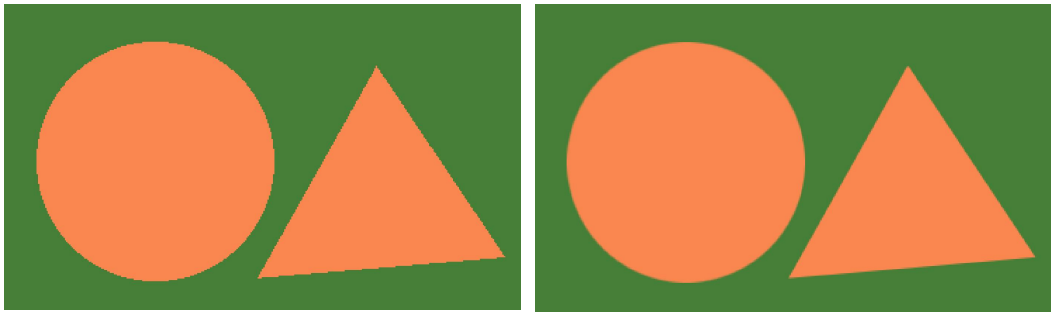
一旦边缘区域被成功检测出来，算法进入子像素处理阶段。在这一阶段，算法对检测到的边缘区域进行平滑处理，以减少锯齿现象。最后，算法执行像素混合步骤。这一步骤将经过平滑处理的像素与原始像素进行混合，以生成最终的抗锯齿图像。

在完成了 FXAA 着色器的编写后，将这些着色器程序加载到图形处理单元中，并进行编译和链接。随后，将这些着色器绑定到渲染管线上，以便在后续的渲染过程中使用。这样，每当场景被渲染时，FXAA 着色器就会自动对图像进行抗锯齿处理。

最后，在启用 FXAA 算法的情况下，重新渲染测试场景，并将结果保存下来。

3.4 实验结果与分析

如图 4 所示为简单几何图形在关闭与开启 FXAA 时的渲染结果。如图 5 所示为复杂纹理在关闭与开启 FXAA 时的渲染结果。



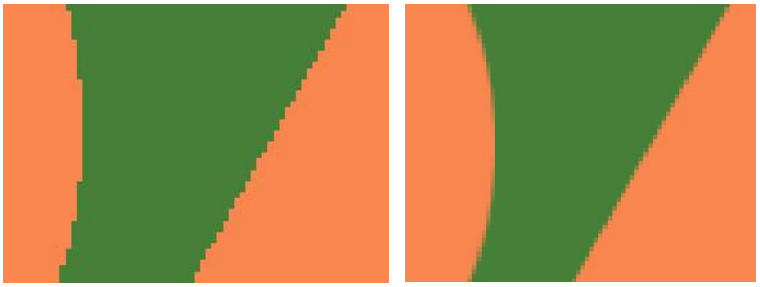


图 4 简单几何图形在关闭（左）与开启（右）FXAA 时的渲染结果

图 5 复杂场景在关闭（左）与开启（右）FXAA 时的渲染结果

通过对比启用 FXAA 前后的图像，我们可以清晰地看到 FXAA 算法对图像质量的改善。在未启用 FXAA 时，图像中的边缘区域存在明显的锯齿现象，特别是在对比度较大的几何边界处。而启用 FXAA 后，这些锯齿现象得到了有效的抑制，图像的边缘更加平滑，图像整体更加自然、连贯，视觉体验更佳。

4 总结

本文深入探讨了计算机图形学中的锯齿效应问题，并专注于研究快速近似抗锯齿（FXAA）算法。文中详细介绍了 FXAA 算法的实现步骤，包括对比度计算、混合系数计算、混合方向确定以及搜索与混合等关键过程。通过这些步骤，FXAA 能够识别并处理图像中的锯齿边界，从而提升图像质量。为验证 FXAA 算法在实际应用中的效果，本文设计了一系列实验，通过对比不同场景下启用和未启用 FXAA 的图像质量，验证其在减少锯齿效应方面的有效性。实验结果表明，FXAA 算法在减少锯齿效应方面效果显著，能够

显著提升图像质量。尤其是在高对比度场景中，FXAA 能够有效地平滑边缘，使图像看起来更加自然和清晰。此外，由于 FXAA 算法的计算速度快、内存开销低，因此在实际应用中具有广泛的适用性。

参考文献

[1] NVIDIA. (2010). “Fast Approximate Anti-Aliasing (FXAA).” NVIDIA Whitepaper.

[2] Akenine-Möller, T., Haines, E., & Hoffman, N. (2018). “Real-Time Rendering.” CRC Press.