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）
21301147	常杰	7	23	16	34	80

缺少对现有工作的梳理，参考文献较少

北京交通大學

计算机图形学课程报告

快速近似反走样算法 FXAA 调研与实现

学院 软件学院

班级 2106

学号 21301147

姓名 常杰

2024 年 6 月 23 日

摘　要

走样是由于对连续信息进行离散采样、存储或表示时采样频率低于信息变化频率所导致的一种信号失真现象, 反走样是计算机图形学研究的基本问题之一。图形反走样可以有效地重建出几何、纹理、运动等各种细节, 从而提高绘制图形和生成动画的质量, 帮助用户获得更好的视觉体验。因此, 反走样一直是计算机图形学研究中的重要课题, 在三维计算机游戏、计算机动画、虚拟现实、影视后期制作等领域具有广泛和重要的应用。

随着计算机及算力的快速发展, 出现了多种反走样算法, 包括: 快速近似反走样 (Fast Approximate Anti-Aliasing)、超采样反走样 (Super Sampling Anti-Aliasing)、多采样反走样 (Multi-Sampling Anti-Aliasing)、增强型子像素形态学反走样 (Enhanced Subpixel Morphological Antialiasing)、形态学反走样 (Morphological Antialiasing)、时间反走样 (Time Anti-Aliasing)、深度学习超采样 (Deep Learning Super Sampling) 等.

这些反走样技术在性能和反走样效果上各异，本文实现其中的 FXAA，并对其算法和效果进行分析。

关键词：反走样，FXAA，SSAA，MSAA，SMAA

目录

1 引言

在数字图像处理和计算机图形学领域，反走样技术 (Anti-Aliasing) 是提高图像质量提升用户观感的关键技术之一。随着计算机显示技术和算力的提高，人们对图片的清晰度和图像观感上的实时性要求也越来越高，反走样技术也越来越受重视。

走样是由于对连续信息进行离散采样、存储或表示时采样频率低于信息变化频率 (及低频采样或不完全采样) 所导致的一种信号失真现象。这种现象通常以伪影的形式展现，具体包括摩尔纹，锯齿状边缘等现象，走样现象不仅会出现在静态图片上，在渲染动态视频的时候也会出现走样、伪影等现象，严重影响观感。因此反走样技术应运而生，出现了 SSAA、MSAA、DLSS 等普遍应用的反走样技术。

本文课程实验论文旨在独立实现性能消耗较小的反走样算法 FXAA，并对其反走样效果进行分析。

2 相关工作介绍

目前的各种反走样技术已经应用广泛，在一些 GPU 上直接内嵌了整套的反走样算法。而本次实验使用的第二代 Web 图形库 (WebGL2) 引入了多重采样渲染缓冲对象技术 (Multisamples Renderbuffer)，支持创建一个多重采样缓冲区，进而可以在帧缓冲区的渲染缓冲区上进行多次采样，使用多个采样点来计算每个像素点颜色，求平均值进而实现 MSAA。而 WebGL 内嵌了反走样技术 SSAA，给本次实验带来便捷。而其余反走样技术则需要通过实现片段着色器来达到反走样效果。同时 CSDN 上存在着 FXAA 的算法讲解，unity 方面的技术人员 CatlikeCoding 也撰写了 FXAA 的算法讲解。

3 方法描述

经过调研，走样现象在图像边缘位置尤为明显，而这一部分再频域上属于高频分量，但基于采样的反走样技术都有一个共同点，就是会在非边缘部分浪费大量的计算资源，其中 SSAA 尤为明显。

而 FXAA 利用计算机视觉领域常用的图像边缘检测技术，提取出图像中的边缘部分，然后做抗锯齿。

具体算法流程如下：

1. 利用 NTSC 1953 经验公式将 RGB 图片转换成亮度图：

$$0.299R + 0.587G + 0.114B = Gray \tag{1}$$

2. 通过亮度图，计算当前像素点与东西南北四个领域像素的亮度差值，如果当前像素点局部最大和最小像素差值小于阈值，则不是边缘像素，保持原亮度值：

其中 lumaM, lumaN, lumaW, lumaS, lumaE 分别表示原像素点与其本身、北、西、南、东的亮度值，而 range 表示原像素点的局部对比度；

Algorithm 1 Calculate range of luminance

float rangeMin = min(lumaM, min(min(lumaN, lumaW), min(lumaS, lumaE)));
float rangeMax = max(lumaM, max(max(lumaN, lumaW), max(lumaS, lumaE)));
float range = rangeMax − rangeMin;

3. 计算原像素点四周的像素点的平均值，及通过一个低通滤波器，然后求与中间亮度的绝对差异，以此得到像素对比度。随后用像素对比度与局部对比度的比率来作为亚像素锯齿的检测，若比值接近 1.0 说明在考察点周围亮度变化均匀，及存在亚像素锯齿；若接近 0.0 则说明亮度变化分布在更大范围内，锯齿不明显。而该比值在后续会作为低通滤波器的强度；

4. 使用 3x3 的 BOX 滤波器平滑子像素处理，以此减少高频噪声，增强图像整体视觉质量；

Algorithm 2 3x3 BOX 滤波器的实现

输入: 图像纹理 tex, 像素位置 pos
输出: 平滑处理后的像素颜色 $rgbM$
procedure BoxFilter(tex, pos)
    $rgbN \leftarrow$ FxaaTextureOffset($tex, pos, (0, -1)$)
    $rgbW \leftarrow$ FxaaTextureOffset($tex, pos, (-1, 0)$)
    $rgbM \leftarrow$ FxaaTextureOffset($tex, pos, (0, 0)$)
    $rgbE \leftarrow$ FxaaTextureOffset($tex, pos, (1, 0)$)
    $rgbS \leftarrow$ FxaaTextureOffset($tex, pos, (0, 1)$)
    $rgbNW \leftarrow$ FxaaTextureOffset($tex, pos, (-1, -1)$)
    $rgbNE \leftarrow$ FxaaTextureOffset($tex, pos, (1, -1)$)
    $rgbSW \leftarrow$ FxaaTextureOffset($tex, pos, (-1, 1)$)
    $rgbSE \leftarrow$ FxaaTextureOffset($tex, pos, (1, 1)$)
    $rgbL \leftarrow rgbN + rgbW + rgbM + rgbE + rgbS + rgbNW + rgbNE + rgbSW + rgbSE$
    $rgbL \leftarrow rgbL \times (1.0/9.0)$ ▷ 计算 9 个像素的平均值
    return $rgbL$
end procedure

5. 取局部 3x3 邻域的行和列的亮度的加权平均幅度作为局部边缘程度的参考；计算垂直边缘强度时，对于中心列 lumaN,、lumaM、lumaS、lumaW、lumaE 其加权系数分别为-0.5、-1.0、-0.5、0.5、0.5，而其余的侧边列全为 0.25；计算水平边缘强度时分别为 0.5、-1.0、0.5、-0.5、-0.5；最后两者相比较判断更偏向于水平还是垂直；

6. 确定局部边缘方向后，选出与其方向垂直且对比度最高的像素对，随后沿着该垂直方向的正方向或反方向搜素，直到到达边缘或沿着边缘移动的像素对平均亮度足以表示边缘。接下来原点会与搜索到的片元颜色进行混合。

7. 端点搜索：确定端点位置，根据端点与原点的距离和像素值来决定原点与其他片元颜色混合时的比例；确定端点时计算每个点的梯度，若他的梯度绝对值大于原点梯度绝对值的 1/4，则其就是端点。这里的梯度是指两点之间的亮度差异。

8. 计算混合比例：找到最近端点之后，用其与原点之间的距离的最小值除以左右两端端点距离和，在这里因为越远，他们的混合比例越小，使混合后原点颜色更接近与其接近的片元颜色，同时这里比值在 0 0.5 之间，故使用 0.5 减去比值来作为混合比例，具体计算公式如下：

$$L1 = |\text{LeftEnd} - \text{Start}| \tag{2}$$

$$L2 = |\text{LeftRight} - \text{Start}| \tag{3}$$

$$\alpha = -1.0 \times \frac{\min(L1, L2)}{L1 + L2} + 0.5 \tag{4}$$

其中 LeftEnd、LeftRight、Start 分别表示左端点、右端点和原点的位置；同时考虑若最近的端点与原点颜色相近，则不混合，防止产生混合的对称性，导致两端点都参与混合。

8. 低通滤波：最后计算原点与其周围点平均亮度的差值，再通过一个经验公式将差异映射成混合比例。

Algorithm 3 Subpixel Offset Calculation

$luma\_average\_center \leftarrow 0.0$

$average\_weight\_mat \leftarrow [1.0, 2.0, 1.0, 2.0, 0.0, 2.0, 1.0, 2.0, 1.0]$

for $i \leftarrow 0$ to 8 **do**

    $luma\_average\_center \leftarrow luma\_average\_center + average\_weight\_mat[i] \times luma\_mat[i]$

end for

$luma\_average\_center \leftarrow luma\_average\_center/12.0$

$subpixel\_luma\_range \leftarrow \text{clamp}(|luma\_average\_center - luma\_mat[\text{CENTER}]|\,/(luma\_max - luma\_min), 0.0, 1.0)$

$subpixel\_offset \leftarrow (-2.0 \times subpixel\_luma\_range + 3.0) \times subpixel\_luma\_range \times subpixel\_luma\_range$

$subpixel\_offset \leftarrow subpixel\_offset \times subpixel\_offset \times \text{SUBPIXEL\_QUALITY}$

$pixel\_offset \leftarrow \max(pixel\_offset, subpixel\_offset)$

其中 luma_max、luma_min 是该区域内的亮度的最大、最小值，clamp 函数将最中取值映射到 0.0 到 1.0 之间；SUBPIXEL_QUALITY 是一个超参数，这个值越大最终的抗锯齿效果越好，但也会消去高光效果。

4 实验结果分析

图 1: 测试原图

图 2: FXAA 处理后

从图中可以看出在门框附近、救生圈周围、水面和桅杆等的锯齿状走样现象得到明

显改善，但是由于 FXAA 使用基于像素邻域的平均化方法来平滑边缘，这也引入模糊效果，导致图像中的细节变得模糊。例如，图像中窗户下的文字变模糊，发生了失真的情况。仔细看水面、船板、船壁周围也出现了失真的情况，如图四。回顾 FXAA 确定边缘和计算原点和其他片元混合比例时会根据端点距离和颜色来确定，因此原点的颜色也会受到邻近像素颜色影响，特别在细节比较多的地方，这也导致了颜色漂移的现象。

图 3: 局部原图

图 4: FXAA 处理后

5 结论

通过本次实验，实现了一种资源消耗较少的反走样算法——快速近似反走样算法 FXAA，同时也对处理后的图片进行了分析，了解到 FXAA 算法的缺陷，即会导致颜色漂移现象，同时也容易消去高光部分；同时在调研学习过程中也了解到 FXAA 算法的优点和减轻 FXAA 带来的模糊现象，可以通过添加一个锐化后处理步骤，即添加一个锐化卷积核，如 Unsharp Mask；或者使用一些锐化算法，比如 Laplacian 和高通滤波器，但这些实现起来都比较难，故本次实验并没有做锐化后处理工作，这也是接下来的调研和学习的方向

参考文献

[1]杜文俊. 基于几何的实时绘制反走样[D].浙江大学,2016.

[2]Catlike Coding, Advanced Rendering, FXAA Smoothing Pixels