1 反走样

1.1 反走样的理论基础

1.1.1 采样定理

从连续的信号 (例如渲染管线中的场景或矢量图) 到离散的信号 (例如显示器上的像素), 我们需要对连续信号进行采样. 最简单的采样方法是在目标像素的中心点对连续信号进行采样. 然而, 离散的信号有时并不能正确地反应连续信号.

根据 Fourier 变换的性质, 可以得出: 对连续函数 $f_0(x)$ 以频率 f_s 进行采样, 最终得到的离散函数 $f_1(x)$ 的频谱 $F_1(f)$ 是把原函数的频谱 $F_0(f)$ 以 f_s 为周期进行周期延拓的结果.

记 $f_0(x)$ 的截止频率为 f_0(即其频域 $F_0(f)$ 在 $f > f_0$ 的区域上均为 0), 如果 $f_s > 2f_0$, 那么将 $F_0(f)$ 按照 f_s 为周期平移时就不会出现重叠, 也就可以准确地复现 $f_0(x)$. 否则, 当 $F_1(f)$ 中出现重叠时, 就出现了错误的频率信号, **走样 (Aliasing)** 就发生了.

定理 1.1 Nyquist-Shannon 采样定理 对于一个截止频率为 f 的连续信号, 如果以大于 $2f$ 的采样频率对其进行采样, 则可以准确重建该信号. 否则, 会发生混叠现象, 无法准确重建信号.

因此, 一切走样几乎都是因为采样频率 f_s 不足导致的.

1.1.2 反走样的基本策略

除了购买一块分辨率更高的屏幕之外, 我们主要有两种反走样的策略:

1. **预过滤 (Pre-filtering)**: 在采样前进行**低通滤波 (Low-pass Filtering)**, 截断高于 $f_s/2$ 的频率.
2. **超采样 (Supersampling)**: 以高于 f_s 的频率进行采样, 然后对采样结果进行低通滤波.

1.2 预过滤

所谓低通滤波, 在图像滤波中我们已经介绍过, 主要是通过各种低通滤波器 (主要是均值滤波器和高斯滤波器等) 来实现的. 然而, 预过滤中的低通滤波是对连续信号进行的, 在实现上稍有不同.

1.2.1 不加权与加权面积采样

我们主要介绍直线的采样. 考虑直线 $l : y = kx + b$, 并将它视作一个具有一定宽度的矩形[1]. 根据该矩形在某一像素内覆盖的面积为像素着色, 即

$$\boldsymbol{c}_{\text{pixel}} = \frac{S_{\text{overlap}}}{S_{\text{pixel}}} \boldsymbol{c}_{\text{line}}$$

[1] 似乎也有根据直线在像素内的长度进行着色的办法, 但各处查找到的定义并不相同. 此外, 这种办法的边缘更加尖锐, 效果不算太好.

这种方法称为**不加权面积采样 (Unweighted Area Sampling)**. 更好的办法是按照像素到直线的距离对像素内的各点进行加权, 即

$$\boldsymbol{c}_{\text{pixel}} = \frac{\iint_{\text{pixel}} w(d(x,y))\boldsymbol{c}_{\text{line}}\mathrm{d}S}{\iint_{\text{pixel}} w(d(x,y))\mathrm{d}S}$$

其中 $d(x,y)$ 为像素内点 (x,y) 到直线的距离. 这种方法称为**加权面积采样 (Weighted Area Sampling)**.

1.2.2 预滤波

考虑原始图像 $\boldsymbol{I}(x,y)$ 和输出图像 $\boldsymbol{J}[x,y]$. 最开始所述的点采样就是把 $\boldsymbol{J}$ 中各像素的中心点在 $\boldsymbol{I}$ 中对应的颜色直接赋值给整个像素:

$$\boldsymbol{J}[x,y] = \boldsymbol{I}\left(x+\frac{1}{2}, y+\frac{1}{2}\right)$$

如前所述, 这会引入走样.

我们可以对 $\boldsymbol{I}$ 作 1×1 的低通滤波, 即对 $\boldsymbol{I}$ 中对应 1×1 像素的区域进行平均:

$$\hat{\boldsymbol{I}}(x,y) = \iint_{[x-1/2,x+1/2]\times[y-1/2,y+1/2]} \boldsymbol{I}(x,y)\mathrm{d}S$$

然后对卷积后的图像 $\hat{I}(x,y)$ 采样:

$$\boldsymbol{J}[x,y] = \hat{\boldsymbol{I}}\left(x+\frac{1}{2}, y+\frac{1}{2}\right)$$

这就相当于 $\boldsymbol{J}$ 中每个像素是 $\boldsymbol{I}$ 中对应 1×1 区域的平均值, 从而避免了走样. 对于渲染中最常见的三角形, 即

$$\boldsymbol{J}[x,y] = \frac{S_{\text{overlap}}}{S_{\text{pixel}}}\cdot \boldsymbol{c}$$

其中 $\boldsymbol{c}$ 为三角形的颜色, S_{pixel} 为像素面积, S_{overlap} 为三角形与像素的重叠面积.

1.2.3 纹理采样与 MIP 映射

在渲染三维场景时, 我们需要把二维图像 (纹理) 贴到三维物体上. 当视点远离物体时 (即对图像进行拉伸缩放等操作), 如果我们直接对纹理进行采样, 会出现明显的走样[2].

解决这一问题的最简单办法是在渲染时进行预过滤. 例如, 考虑一张 1024×1024 的纹理, 在最终贴图中对应于一个 32×32 的区域. 我们可以对纹理做 32×32 的均值滤波, 然后对滤波后的图像进行采样. 然而, 在渲染复杂场景, 贴图数量众多的情况下, 这种方法的计算量过大, 不太现实.

我们需要一种更高效合理的方法. 考虑对上述纹理 (记作 Level0) 进行预先采样, 每次进行 2×2 的均值滤波, 依次得到 512×512, 256×256 等 (分别记作 Level $1,2,\cdots$) 尺寸图像. 在实际渲染时, 我们考虑目标区域的尺寸介于两张贴图的尺寸之间. 对于每个待确定的像素 i, 通过一定的算法 (这在三维渲染中应该会介绍) 得出层次细节指数 λ_i, 选取要用到的两张贴图:

$$D_0 = \lfloor \lambda_i \rfloor, \quad D_1 = \lceil \lambda_i \rceil$$

[2]极端一点, 如果我们打算在一个像素内渲染人像, 如果不小心采样到头发上, 就能看见一个黑色的头. 这显然是不合适的.

于是需要使用 Level D_0, D_1 两张贴图. 接下来先确定像素 i 的采样点在 uv 坐标中的位置 (u_i, v_i), 然后用该位置在两张贴图中对应的像素进行线性插值:

$$u_{i0} = \lfloor u_i \rfloor, \quad u_{i1} = \lceil u_i \rceil, \quad v_{i0} = \lfloor v_i \rfloor, \quad v_{i1} = \lceil v_i, \quad \delta_u = u_i - u_{i0}, \quad \delta_v = v_i - v_{i0} \rceil$$

$$\boldsymbol{c}_i = (1-\delta_u)(1-\delta_v)\boldsymbol{I}[u_{i0}, v_{i0}] + \delta_u(1-\delta_v)\boldsymbol{I}[u_{i1}, v_{i0}] + (1-\delta_u)\delta_v\boldsymbol{I}[u_{i0}, v_{i1}] + \delta_u\delta_v\boldsymbol{I}[u_{i1}, v_{i1}]$$

在 Level D_0, D_1 两张贴图中分别可以确定 $\boldsymbol{c}_{i0}, \boldsymbol{c}_{i1}$ 两个颜色. 最终, 在这两个颜色之间做线性插值可得最终像素 i 的颜色为

$$\boldsymbol{c}_i = (1-\lambda)\boldsymbol{c}_{i0} + \lambda\boldsymbol{c}_{i1}$$

这就是**多级渐远纹理映射 (MIP Mapping)** 的基本原理. 不难得出, 我们仅仅增加了 1/3 的空间开销, 就能大大加快渲染过程并减少走样发生.

1.3 超采样

1.3.1 超采样反走样 (SSAA)

超采样的基本思想是以高于显示器分辨率的频率对场景进行采样, 然后对采样结果进行低通滤波. 例如, 对于一个 800×600 的显示器, 我们可以以 1600×1200 的频率对场景进行采样, 然后对采样结果进行 2×2 的均值滤波, 最后将滤波后的图像下采样到 800×600. 这种方法称为**超采样反走样 (Supersampling Anti-Aliasing, SSAA)**.

定义 1.2 SSAA **超采样反走样**是指以高于显示器分辨率的频率对场景进行采样, 然后对采样结果进行低通滤波, 最后将滤波后的图像缩小到显示器分辨率的过程.

SSAA 等价于在输出图像 $\boldsymbol{J}$ 的每个像素对应于原始图像 $\boldsymbol{I}$ 的区域内进行多次采样, 然后取平均值. 不难看出, 在采样方式比较规则时, 这等价于扩大分辨率后下采样至输出图像[3].

SSAA 的实现简单, 效果也最好, 但开销是最大的. 在实际应用中很少如此使用.

1.3.2 多重采样反走样 (MSAA)

SSAA 的低效主要来源于对每个采样点颜色的单独计算, 这意味着运行时间成倍的提升. **多重采样反走样 (Multisample Anti-Aliasing, MSAA)** 就是一种更高效的办法.

仍然考虑目标图像 $\boldsymbol{J}$ 的一个像素 $\boldsymbol{J}[x, y]$. 和 SSAA 一样, 我们先取定 $\boldsymbol{J}[x, y]$ 中的 n 个采样点, 不妨记作 $S_1, \cdots, S_n$. MSAA 的步骤如下:

1. **几何测试**: 对每个采样点 S_i 做三角形覆盖测试, 确定其最终被哪个三角形覆盖.
2. **着色计算**: 对某一三角形, 按照重心插值方法在 $\boldsymbol{J}[x, y]$ 中心计算一次颜色, 然后把该颜色赋值给所有被该三角形覆盖的采样点.

[3]但实际超采样时通常会避免将像素简单地划为规则排列的格子, 而更常用低差异采样序列, 即按一定随机形式排布的采样点. 这能更有效地避免走样.

3. 重复 **2.**, 直到所有采样点 S_i 都被处理完毕.

4. **合并颜色**: 对像素 $\boldsymbol{J}[x,y]$ 内的所有采样点的颜色取平均, 作为 $\boldsymbol{J}[x,y]$ 的颜色.

定义 1.3 MSAA **多重采样反走样**是指对输出图像的每个像素取多个采样点, 对每个采样点做几何测试, 但只计算像素中心的颜色对采样点赋值, 最后对采样点颜色取平均作为像素颜色的过程.

无疑, MSAA 相比 SSAA 大大减少了着色器的调用次数, 尤其是在三角形的内部. 此外, 它还需要成倍的内存开销来存储采样点的信息. 随着现代渲染场景的复杂化, MSAA 也逐渐被更高效的反走样方法所取代.

1.3.3 时域反走样 (TAA)

一般而言, 渲染图像都是在时间上是连续的. 因此, 除了从空间上优化采样方式, 还可以从时间上进行优化. **时域反走样 (Temporal Anti-Aliasing, TAA)** 就是一种利用时间信息进行反走样的方法.

TAA 的步骤如下:

1. **抖动采样位置**: 在 MSAA 中, 我们是在一帧的一个像素内取多个位置不同的采样点. 在 TAA 中, 我们则是在连续的多帧的一个像素内取位置不同的采样点, 这就需要对采样点的位置进行轻微移动.

2. **着色计算**: 正常执行渲染管线, 计算当前帧当前采样点的颜色.

3. **回溯采样点**: 对于当前帧的采样点 S_{current}, 通过一定方法[4]找到其在上一帧中对应区域的采样点 S_{history}.

4. **合并颜色**: 对当前帧的采样点 S_{current} 和上一帧的采样点 S_{history} 的颜色进行加权平均:

$$\boldsymbol{c}_{\text{result}} = (1-\alpha)\boldsymbol{c}_{\text{history}} + \alpha\boldsymbol{c}_{\text{current}}$$

其中 α 为权重系数, 一般取 0.05 到 0.2 左右. 将上述过程展开如下 (下标 t 表示时间):

$$\boldsymbol{c}_{t,\text{result}} = \alpha\boldsymbol{c}_t + (1-\alpha)\boldsymbol{c}_{t-1,\text{result}} = \alpha\sum_{k=0}^{\infty}(1-\alpha)^k\boldsymbol{c}_{k-t}$$

可以看到当前帧的颜色是对历史帧加权平均的结果.

在实际动态图像的渲染中, TAA 的算法更为复杂 (例如根据移动的程度决定混合权重系数等), 但基本思路是一致的.

定义 1.4 TAA **时域反走样**是指在连续的多帧中对同一像素取位置不同的采样点, 对当前帧的采样点进行着色计算, 然后通过回溯找到上一帧对应区域的采样点, 最后对当前帧和上一帧的采样点颜色进行加权平均作为当前帧像素颜色的过程.

TAA 的优点是开销较小 (每一帧采样着色的次数明显小于 SSAA 和 MSAA), 但缺点也很明显, 即会引入模糊和拖影现象. 这在动态范围较大时更为明显.

[4]这在后面的渲染的课程会介绍.

1.3.4 基于深度学习的反走样

这里主要以 NVIDIA 的**深度学习超级采样 (Deep Learning Super Sampling, DLSS)** 为例进行介绍. DLSS 的基本思路是利用深度学习的方法, 通过对大量高质量图像的训练, 让神经网络学习如何将低分辨率图像转换为高分辨率图像.

具体而言, NVIDIA 通过大量数据 (包括一个低分辨率的图像作为输入和对应的高分辨率图像作为输出) 预训练模型, 然后将参数写入显卡中. 在实际渲染时, 只需通过采样得到一个低分辨率图像, 然后通过神经网络进行推理, 即可输出对应的高分辨率图像, 达到超采样的效果.

DLSS 技术和前面讲的几种反走样技术在实现原理上是不同的, 它纯粹是力大砖飞的提高了分辨率. 此外, 模型还可以通过渲染帧的数据推理预测它们之间的画面, 实现插帧的效果. 在 2K 和 4K 分辨率下, DLSS 对于帧率和画质的提升是飞跃级的, 几乎能达到四倍于原始帧率的程度.

当然, DLSS 也有明显的缺点, 例如需要一张 NVIDIA 的 Geforce RTX 20 系列以上的, 具有 Tensor Core 的显卡. 最新的 DLSS 4.0 只能在 Geforce RTX 50 系列显卡使用. 此外, 在高动态的场景下, DLSS 仍会出现模糊和拖影现象.

另一种无需硬件支持的技术是 AMD 推出的**超分辨率锐画技术 (FidelityFX Super Resolution, FSR)**. FSR 的原理和 DLSS 类似, 但不依赖 Tensor Core 等硬件, 可以在大多数显卡上使用. 然而, FSR 的效果不如 DLSS.

总而言之, 基于深度学习的反走样技术是目前最先进的反走样技术, 它几乎战胜了以往所有的反走样技术, 成为现代渲染中反走样和提高帧率的主要技术.