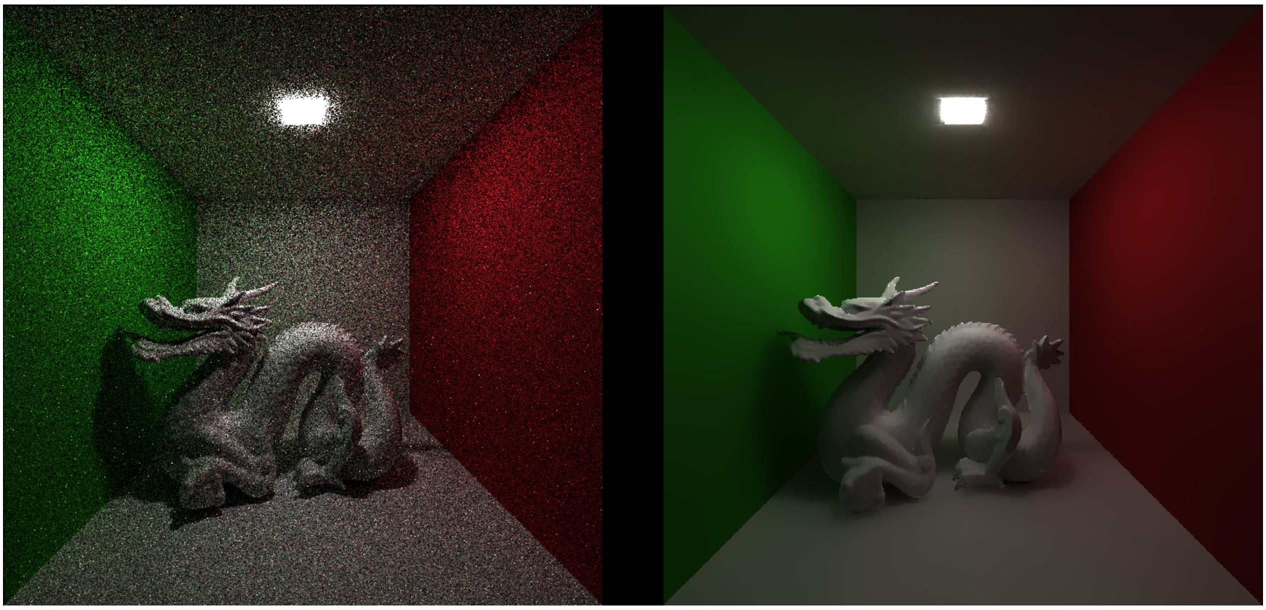
方差引导滤波器中的方差

Dezeming Family

2021 年 5 月 5 日



目录

一 总体介绍

最近也做了一段时间的去噪技术了，打算总结一下与方差引导滤波器有关的方差计算。对于较低样本数量的前提下，我们很难直接像统计学那样计算一堆样本的方差，因此需要一些特殊的处理手段。本文旨在总结和归纳方差计算的方法。

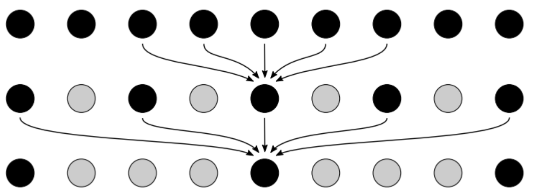
准备知识：交叉双边滤波器、引导滤波器、高斯滤波核（用来近似多孔小波）。

二 SVGF 方差计算

使用方差的宗旨：避免滤波器对噪声较小的位置进行滤波。

注意事项：空间计算的方差估计只能为噪声提供一个不完美的代理：噪声增加方差，但方差可以在没有噪声的情况下发生。

这里的多孔小波变换是用高斯核来近似的，我们可以理解为，用不断扩大核宽的高斯核来对图像进行迭代滤波，一般迭代 5 次。



只不过如上图所示，每次迭代，滤波半径变大，但会间隔 i 个样本。第一次滤波后，5 个样本的影响进入下一级，作为中间元素值，然后再依次迭代，论文中一般会迭代 5 次，因此滤波半径最后会扩大到 65 个元素（长宽都是 65 个元素：5+4*1+4*2+4*3+4*4+5*4=65）。

2 1 滤波器概述

下面描述一下滤波器的结构：

$$\hat{c}_{i+1}(p) = \frac{\sum_{q\in\Omega} h(q)w(p,q)\hat{c}_i(q)}{\sum_{q\in\Omega} h(q)w(p,q)} \tag{二.1}$$

h 就是滤波核，比如使用高斯核、多孔小波核等。

w 表示权重函数，这里的权重就类似于双边滤波器中的距离和颜色权重。SVGF 一共有三种权重：depth，world-space normals，luminance。深度权重（或者位置权重）以及法向量权重都很简单，这里不再介绍，我们重点关注的是基于 luminance 的方差权重部分：

$$w_i(p,q) = w_z \cdot w_n \cdot w_l \tag{二.2}$$

luminance 边缘停止函数是基于局部标准差，自动适应所有规模的 re-normalizing luminance。但是在低样本数下操作会在我们的方差和标准差估计中引入不稳定性，会引入 artifacts。为了避免这些问题，我

们使用 3×3 高斯核对计算得到的方差图像进行预滤波，这显著提高了重建质量。下面公式中，σ_l 表示我们设置的权重参数（常数）。

$$w_l = exp\left(-\frac{|l_i(p) - l_i(q)|}{\sigma_l|\sqrt{g_{3\times3}(Var(l_i(p)))} + \epsilon}\right) \tag{二.3}$$

该高斯预滤波器仅用于驱动 luminance 边缘停止函数，而不传播到下一次小波变换迭代的方差图像。我们可以从上式看到，随着滤波器不断迭代（多孔小波需要迭代多次），w_l 权重会不断减小，这是为了防止避免过度模糊。

假设方差样本不相关：

$$Var(\hat{c}_{i+1}(p)) = \frac{\sum_{q\in\Omega} h(q)^2 w(p,q)^2 Var(\hat{c}_i(q))}{(\sum_{q\in\Omega} h(q)w(p,q))^2} \tag{二.4}$$

2 2 方差计算

每个像素的 luminance 方差的估计使用一阶矩 μ_{1_i} 和二阶矩 μ_{2_i}（first and second raw moments）来估计，

通过积累矩（integrated moments）μ'_{1_i} 和 μ'_{2_i} 计算时序方差，$\sigma'^2_i = \mu'_{2_i} - \mu'^2_{1_i}$。

当时序样本的数量有限时，我们估计空间方差。

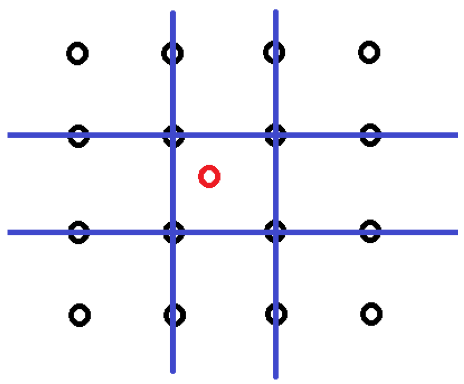
2 3 实现方法

具体实现过程我们参考源码 [2]。

在处理当前帧时，首先需要根据反投影技术，投影到先前帧的位置，计算先前帧的信息，定义：

```
//先前帧的矩
glm::vec2 prevMoments = glm::vec2(0.0f);
//先前帧历史样本总长度
float prevHistoryLength = 0.0f;
```

因为当前帧的点反投影到的先前帧位置很可能在四个像素点的中间，因此需要对各个点的贡献进行插值（一般用双线性插值）。



如果没有历史样本，则自己设置一个方差常数，并定义初始矩。

```
//计算luminance
float luminance = 0.2126 * sample.x + 0.7152 * sample.y + 0.0722 * sample.z;
moment_acc[p] = glm::vec2(luminance, luminance * luminance);
variacne_out[p] = 100.0f;
```

如果有历史样本，就根据历史样本计算新的矩：

```
//把反投影位置的特定半径内的有效像素样本计算历史矩
prevMoments += moment_history[q];
cnt += 1.0f;
......
prevMoments /= cnt;
prevHistoryLength /= cnt;
//计算指数平均的衰减参数，源码中是除以N+1，但是我认为不应该是当前帧位置的历史值，而应该是投影后位置的历史值，因此我改为了prevHistoryLength+1
float moment_alpha = max(1.0f / (float)(prevHistoryLength + 1), moment_alpha_min);
//计算积累矩
float first_moment = moment_alpha * prevMoments.x + (1.0f - moment_alpha) * luminance;
float second_moment = moment_alpha * prevMoments.y + (1.0f - moment_alpha) * luminance * luminance;
moment_acc[p] = glm::vec2(first_moment, second_moment);
// 计算方差，为了让方差有一定影响效果这里乘以了100
float variance = second_moment - first_moment * first_moment;
variacne_out[p] = variance > 0.0f ? 100.0f * variance : 0.0f;
```

在启动多孔小波后，首先对于滤波半径里的方差图像进行滤波（注意多孔算法里，每次滤波半径都会变大，但因为有孔，所以每次滤波的样本数都是固定的）：

```
sum += gaussian[sampleIdx] * variance[loc.x + loc.y * res.x];
sumw += gaussian[sampleIdx];
......
var = max(sum / sumw, 0.0f);
```

然后会使用该方差计算相对应的边缘停止函数：

```
float wl = expf(-glm::distance(lp, lq) / (sqrt(var) * sigma_c + 1e-6));
```

我们可以看到，[2] 中并没有估计空间方差。其时序方差计算式其实就是最简单的方差计算方法：

$$\sigma^2 = \frac{\sum(x-\hat{x})^2}{n} \tag{二.5}$$

其中，$\hat{x}$ 表示样本均值。上式可以进行化简：

$$\begin{aligned}
\sigma^2 = \frac{\sum(x-\hat{x})^2}{n} &= \frac{(x_1-\hat{x})^2+(x_2-\hat{x})^2+...}{n} && \text{(二.6)}\\
&= \frac{1}{n}(x_1^2+x_2^2+...+n\hat{x}^2-2\hat{x}(x_1+x_2+...)) && \text{(二.7)}\\
&= \frac{1}{n}(\sum x^2)-\hat{x}^2 && \text{(二.8)}
\end{aligned}$$

计算方法

当我们拥有源源不断的样本输入时，应该如何计算样本方差呢？

我们定义两个变量分别进行记录，变量 1：s^2 记录当前所有的 x_i^2 的和，初始为 0；变量 2：s 记录当

前所有的 x_i 的和，初始为 0。当第一个样本输入后，很明显，方差为：

$$s^2 += x_1^2 \tag{二.9}$$

$$s += x_1 \tag{二.10}$$

$$\sigma^2 = \frac{s^2}{1} - \left(\frac{s}{1}\right)^2 = 0 \tag{二.11}$$

当输入第二个样本后：

$$s^2 += x_2^2 \tag{二.12}$$

$$s += x_2 \tag{二.13}$$

$$\sigma^2 = \frac{s^2}{2} - \left(\frac{s}{2}\right)^2 \tag{二.14}$$

以此类推即可。

参考文献

[1] Schied C , Salvi M , Kaplanyan A , et al. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination[C]// High Performance Graphics. ACM, 2017.

[2] https://github.com/ZheyuanXie/CUDA-Path-Tracer-Denoising