Bilateral Filtering(双边滤波) for SSAO

1. 简介

图像平滑是一个重要的操作，而且有多种成熟的算法。这里主要简单介绍一下Bilateral方法（双边滤波），这主要是由于前段时间做了SSAO，需要用bilateral blur 算法进行降噪。Bilateral blur相对于传统的高斯blur来说很重要的一个特性即可可以保持边缘（Edge Perseving），这个特点对于一些图像模糊来说很有用。一般的高斯模糊在进行采样时主要考虑了像素间的空间距离关系，但是却并没有考虑像素值之间的相似程度，因此这样我们得到的模糊结果通常是整张图片一团模糊。Bilateral blur的改进就在于在采样时不仅考虑像素在空间距离上的关系，同时加入了像素间的相似程度考虑，因而可以保持原始图像的大体分块进而保持边缘。在于游戏引擎的post blur算法中，bilateral blur常常被用到，比如对SSAO的降噪。

2. 原理

滤波算法中，目标点上的像素值通常是由其所在位置上的周围的一个小局部邻居像素的值所决定。在2D高斯滤波中的具体实现就是对周围的一定范围内的像素值分别赋以不同的高斯权重值，并在加权平均后得到当前点的最终结果。而这里的高斯权重因子是利用两个像素之间的空间距离（在图像中为2D）关系来生成。通过高斯分布的曲线可以发现，离目标像素越近的点对最终结果的贡献越大，反之则越小。其公式化的描述一般如下所述：

$$\mathbf{h}(\mathbf{x}) = k_d^{-1}(\mathbf{x}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{f}(\xi) c(\xi, \mathbf{x}) d\xi$$

$$k_d(\mathbf{x}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{x}) d\xi$$

其中的c即为基于空间距离的高斯权重，而

$$k_d(\mathbf{x})$$

用来对结果进行单位化。

高斯滤波在低通滤波算法中有不错的表现，但是其却有另外一个问题，那就

是只考虑了像素间的空间位置上的关系，因此滤波的结果会丢失边缘的信息。这里的边缘主要是指图像中主要的不同颜色区域（比如蓝色的天空，黑色的头发等），而Bilateral就是在Gaussian blur中加入了另外的一个权重分部来解决这一问题。Bilateral滤波中对于边缘的保持通过下述表达式来实现：

$$\mathbf{h}(\mathbf{x}) = k_r^{-1}(\mathbf{x}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{f}(\xi) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x})) d\xi$$

$$k_r(\mathbf{x}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(\mathbf{f}(\xi), \mathbf{f}(\xi)) d\xi$$

其中的s为基于像素间相似程度的高斯权重，

$$k_r(\mathbf{x})$$

同样用来对结果进行单位化。对两者进行结合即可以得到基于空间距离、相似程度综合考量的Bilateral滤波：

$$\mathbf{h}(\mathbf{x}) = k^{-1}(\mathbf{x}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{f}(\xi) c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\xi)) d\xi$$

$$k(\mathbf{x}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x})) d\xi$$

上式中的单位化分部

$$k(\mathbf{x})$$

综合了两种高斯权重于一起而得到，其中的c与s计算可以详细描述如下：

$$c(\xi, \mathbf{x}) = e^{-\frac{1}{2}(\frac{d(\xi, \mathbf{x})}{\sigma_d})^2}$$

且有

$$d(\xi, \mathbf{x}) = d(\xi - \mathbf{x}) = \|\xi - \mathbf{x}\|$$

$$s(\xi, \mathbf{x}) = e^{\frac{1}{2}(\frac{\sigma(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))}{\sigma_r})^2}$$

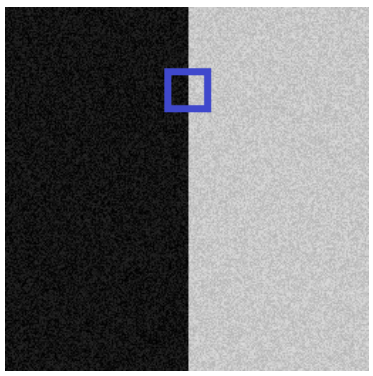
且有

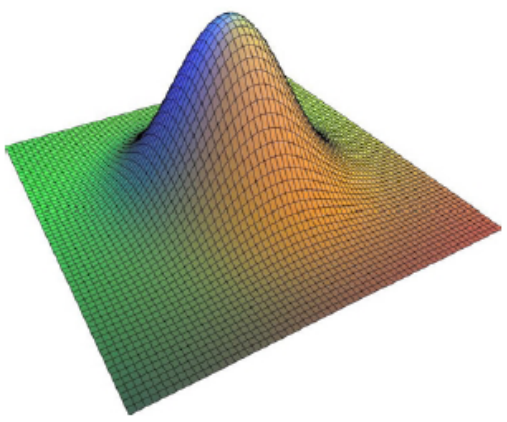
$$\sigma(\phi, \mathbf{f}) = \sigma(\phi - \mathbf{f}) = \|\phi - \mathbf{f}\|$$

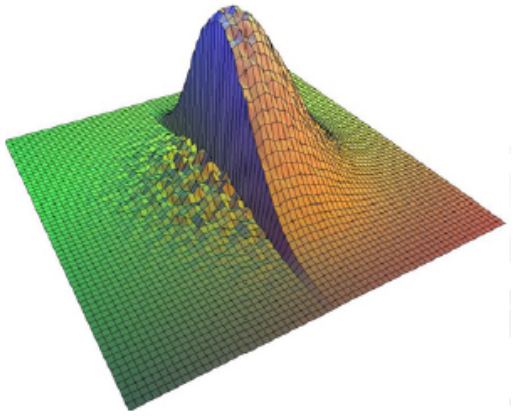
上述给出的表达式均是在空间上的无限积分，而在像素化的图像中当然无法这么做，而且也没必要如此做，因而在使用前需要对其进行离散化。而且也不需要对于每个局部像素从整张图像上进行加权操作，距离超过一定程度的像素实际上对当前的目标像素影响很小，可以忽略的。限定局部子区域后的离散化公就可以简化为如下形式：

$$\mathbf{h}(\mathbf{x}) = k^{-1}(\mathbf{x}) \sum_{\Omega} \mathbf{f}(\xi) c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\xi))$$

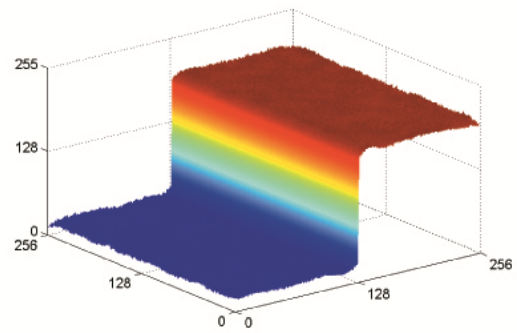
上述理论公式就构成了Bilateral滤波实现的基础。为了直观地了解高斯滤波与双边滤波的区别，我们可以从下列图示中看出依据。假设目标源图像为下述左右区域分明的带有噪声的图像（由程序自动生成），蓝色框的中心即为目标像素所在的位置，那么当前像素处所对应的高斯权重与双边权重因子3D可视化后的形状如后边两图所示：

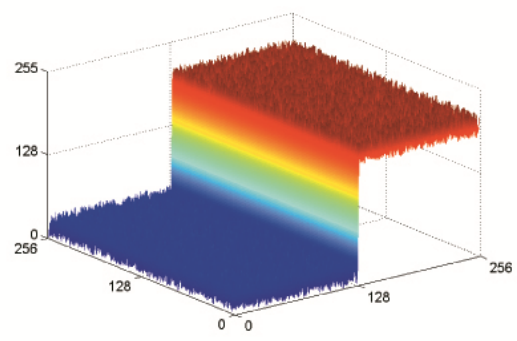


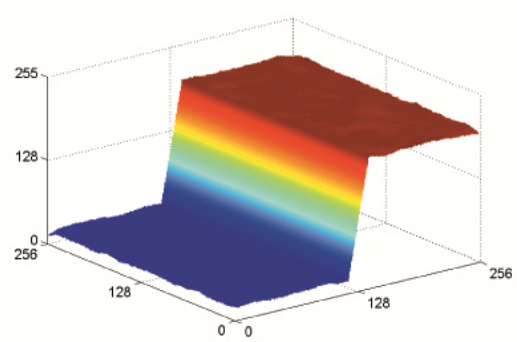




左图为原始的噪声图像；中间为高斯采样的权重；右图为Bilateral采样的权重。从图中可以看出Bilateral加入了相似程度分部以后可以将源图像左侧那些跟当前像素差值过大的点给滤去，这样就很好地保持了边缘。为了更加形象地观察两者间的区别，使用Matlab将该图在两种不同方式下的高度图3D绘制出来，如下：







上述三图从左到右依次为：双边滤波，原始图像，高斯滤波。从高度图中可以明显看出Bilateral和Gaussian两种方法的区别，前者较好地保持了边缘处的梯度，而在高斯滤波中，由于其在边缘处的变化是线性的，因而就使用连

累的梯度呈现出渐变的状态，而这表现在图像中的话就是边界的丢失（图像的示例可见于后述）。

3. 代码实现

有了上述理论以后实现Bilateral Filter就比较简单了，其实它也与普通的Gaussian Blur没有太大的区别。这里主要包括3部分的操作: 基于空间距离的权重因子生成；基于相似度的权重因子的生成；最终filter颜色的计算。

3.1 Spatial Weight

这就是通常的Gaussian Blur中使用的计算高斯权重的方法，其主要通过两个pixel之间的距离并使用如下公式计算而来：

$$c(\xi, \mathbf{x}) = e^{-\frac{1}{2}(\frac{d(\xi,\mathbf{x})}{\sigma_d})^2} = e^{-\frac{1}{2}(\frac{\|\xi-\mathbf{x}\|}{\sigma_d})^2}$$

其中的

$$\xi - \mathbf{x}$$

就表示两个像素间的距离，比如当前像素与其右边紧邻的一个像素之间的距离我们就可以用

$$\|\{0,0\} - \{0,1\}\|$$

来计算，也即两个二维向量*{0 , 0}*以及*{0 , 1}*之间的欧氏距离。直接计算一个区域上的高斯权重并单位化后就可以进行高斯模糊了。

3.2 Similarity Weight

与基于距离的高斯权重计算类似，只不过此处不再根据两个pixel之间的空间距离，而是根据其相似程度（或者两个pixel的值之间的距离）。

$$s(\xi, \mathrm{x}) = e^{-\frac{1}{2}(\frac{\sigma(\mathrm{f}(\xi),\mathrm{f}(\mathrm{x}))}{\sigma_r})^2}$$

其中的

$$\sigma(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))$$

表示两个像素值之间的距离，可以直接使用其灰度值之间的差值或者RGB向量之间的欧氏距离。

3.3 Color Filtering

有了上述两部分所必需的权重因子之后，那么具体的双边滤波的实现即与普通的高斯滤波无异。主要部分代码如下述：

```
1.  UCHAR3 BBColor(int posX , int posY)
2.  {
3.      int centerItemIndex = posY * picWidth4 + posX * 3 , neighbourItem
    Index;
4.      int weightIndex;
5.      double gsAccumWeight = 0;
6.      double accumColor = 0;
7.
8.
9.      for(int i = -number ; i <= number ; ++i)
10.     {
11.         for(int j = -number ; j <= number ; ++j)
12.         {
13.             weightIndex = (i + number) * (number * 2 + 1) + (j + number);
14.             neighbourItemIndex = min(noiseImageHeight - 1 , max(0 , pos
    Y + j * radius)) * picWidth4 +
15.                         min(noiseImageWidth - 1  , max(0 , posX + i * radius)
    ) * 3;
16.
17.             pCSWeight[weightIndex] = LookupGSWeightTable(pSrcDataB
    uffer[neighbourItemIndex] , pSrcDataBuffer[centerItemIndex]);
18.             pCSWeight[weightIndex] = pGSWeight[weightIndex] * pGCWe
    ight[weightIndex];
19.             gsAccumWeight += pCSWeight[weightIndex];
20.         }
21.     }
22.
23.
24.     gsAccumWeight = 1 / gsAccumWeight;
25.     for(int i = -number ; i <= number ; ++i)
```

```
26.     {
27.         for(int j = -number ; j <= number ; ++j)
28.         {
29.             weightIndex = (i + number) * (number * 2 + 1) + (j + number);
30.             pCSWeight[weightIndex] *= gsAccumWeight;
31.         }
32.     }
33.
34.
35.     for(int i = -number ; i <= number ; ++i)
36.     {
37.         for(int j = -number ; j <= number ; ++j)
38.         {
39.             weightIndex = (i + number) * (number * 2 + 1) + (j + number);
40.             neighbourItemIndex = min(noiseImageHeight - 1 , max(0 , pos
    Y + j * radius)) * picWidth4 +
41.                                  min(noiseImageWidth - 1  , max(0 , posX + i * radi
    us)) * 3;
42.                 accumColor += pSrcDataBuffer[neighbourItemIndex + 0] * pC
    SWeight[weightIndex];
43.         }
44.     }
45.
46.     return UCHAR3(accumColor , accumColor , accumColor);
47. }
```

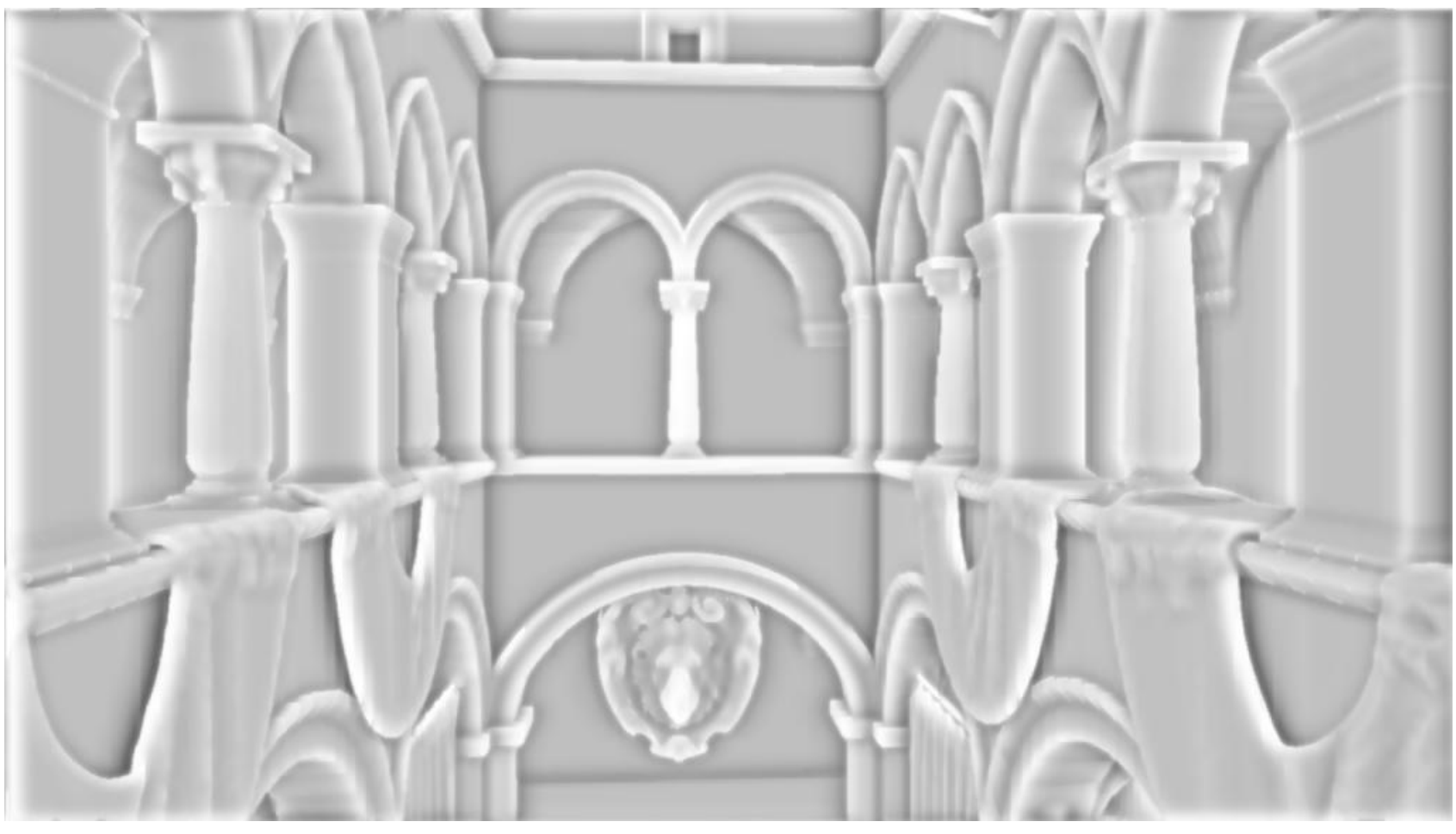
其中的相似度分部的权重**s**主要根据两个Pixel之间的颜色差值计算面来。对于灰度图而言，这个差值的范围是可以预知的，即[-255, 255]，因而为了提高计算的效率我们可以将该部分权重因子预计算生成并存表，在使用时快速查询即可。使用上述实现的算法对几张带有噪声的图像进行滤波后的结果如下所示：

上图从左到右分别为：双边滤波；原始图像；高斯滤波。从图片中可以较为明显地看出两种算法的区别，最直观的感受差别就是使用高斯算法后整张图片都是一团模糊的状态；而双边滤波则可以较好地保持原始图像中的区域信息，看起来仍然嘴是嘴、眼是眼（特别是在第一张美女图像上的效果！看来PS是灰常重要啊~~^o^）。

4. 在SSAO中的使用

在上述实现中的边缘判定函数主要是通过两个像素值之间的差异来决定，这也是我们观察普通图片的一种普遍感知方式。当然，也可以根据使用的需求情况来使用其它的方式判断其它定义下的边缘，比如使用场景的depth或是normal。比如在对SSAO进行滤波时可以直接使用Depth值来行边缘判断。首先，设置一个深度的阈值，在作边缘检测时比较两点间的depth差值，如果差值大于阈值，则认为其属于不同的区域，则此处就应为边界。使用此方法得到的效果可见于下图所示：

高斯滤波



双边滤波

在得到滤波之后的SSAO图像之后，与原始图像进行直接的整合就可以得到最终的渲染效果，如下图所示：

SSAO关闭

SSAO开启

后记： 崭新的2012年自己以一篇博文开始，感觉也不错，加油~！~!