双边滤波算法的原理、流程、实现及效果

一、引言

双边滤波在图像处理领域中有着广泛的应用，比如去噪、去马赛克、光流估计等等，最近，比较流行的Non-Local算法也可以看成是双边滤波的一种扩展。自从Tomasi et al等人提出该算法那一天起，如何快速的实现他，一直是人们讨论和研究的焦点之一，在2011年及2012年Kunal N. Chaudhury等人发表的相关论文中，提出了基于三角函数关系的值域核算法，能有效而又准确的实现高效双边算法。本文主要对此论文提出的方法加以阐述。

双边滤波的边缘保持特性主要是通过在卷积的过程中组合空域函数和值域核函数来实现的，典型的核函数为高斯分布函数，如下所示：

$$\tilde{f}(x)=\eta^{-1}\int_{\Omega} w_{\sigma_s}(y)\phi_{\sigma_r}(f(y)-f(x))f(y)dy$$

(1)

其中:

$$\eta=\int_{\Omega} w_{\sigma_s}(y)\phi_{\sigma_r}(f(y)-f(x))dy$$

(2)

为归一化的作用。

σ_s为空域高斯函数的标准差，σ_r为值域高斯函数的标准差，Ω表示卷积的定义域。 可见，在图像的平坦区域，f(y)-f(x)的值变化很小，对应的值域权重接近于1，此时空域权重起主要作用，相当于直接对此区域进行高斯模糊，在边缘区域，f(y)-f(x)会有较大的差异，此时值域系数会下降，从而导

致此处整个核函数的分布的下降，而保持了边缘的细节信息。

直接的编码实现上述过程是相当耗时的，其时间复杂度为$O(\sigma_s^2)$，因此严重的限制住了该算法的推广和实际使用。不断有学者提出了解决的办法，其中Porikli基于一些假定对此过程进行了优化，比如我就实现过其中一种：空域函数为均值函数，值域为任何其他函数，此时可以用直方图技术进行处理，可减少计算量，但我的实践表明该算法那速度还是慢，并且效果也不好。

在2011的论文《Fast O(1) bilateral filtering using trigonometric range kernels》中，作者提出了用Raised cosines函数来逼近高斯值域函数，并利用一些特性把值域函数分解为一些列函数的叠加，从而实现函数的加速。下面我们重点描述下该过程。

二、推导

1、一些基础理论和常识。

(1) Cos函数在[-Pi/2,Pi/2]之间为非负、对称、在半周期内单调递增以及且有峰值的函数；

(2) 欧拉公式： exp(ix)=cos(x)+isin(x);

(3) 分配律： exp(a+b)=exp(a)*exp(b)；

(4) 图像的动态范围：[0,T]，比如对于灰度图像即为[0,255];

2、一些有用的论证

(1) 对于式子：

$$\phi(s)=\left[\cos(\gamma s)\right]^N \tag{3}$$

其中s是自变量，取值范围[-T,T]，令γ= Pi / 2T，则γs的值在[-Pi/2,Pi/2]内。此时，可以证明：

$$\phi(s)=\left[\cos(\gamma s)\right]^N=\sum_{n=0}^{N}2^{-N}\binom{N}{n}\exp(i(2n-N)\gamma s)$$

(4)

(2) 当N足够大时，有下式成立：

$$\left[\cos(\frac{\gamma s}{\rho\sqrt{N}})\right]^N\approx\exp(-\frac{s^2}{2\rho^2\gamma^{-2}})$$

(5)

如果令ρ=γσ，则上式就变为：

$$\left[\cos(\frac{\gamma s}{\rho\sqrt{N}})\right]^N\approx\exp(-\frac{s^2}{2\rho^2\gamma^{-2}})=\exp(\frac{-s^2}{2\sigma^2})$$

(6)

同样，上面成立的条件也必须有：

$$\frac{\gamma s}{\rho\sqrt{N}}\in\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$$

当γs的值在[-Pi/2,Pi/2]时，因此只需要

$$\rho\sqrt{N}>1$$

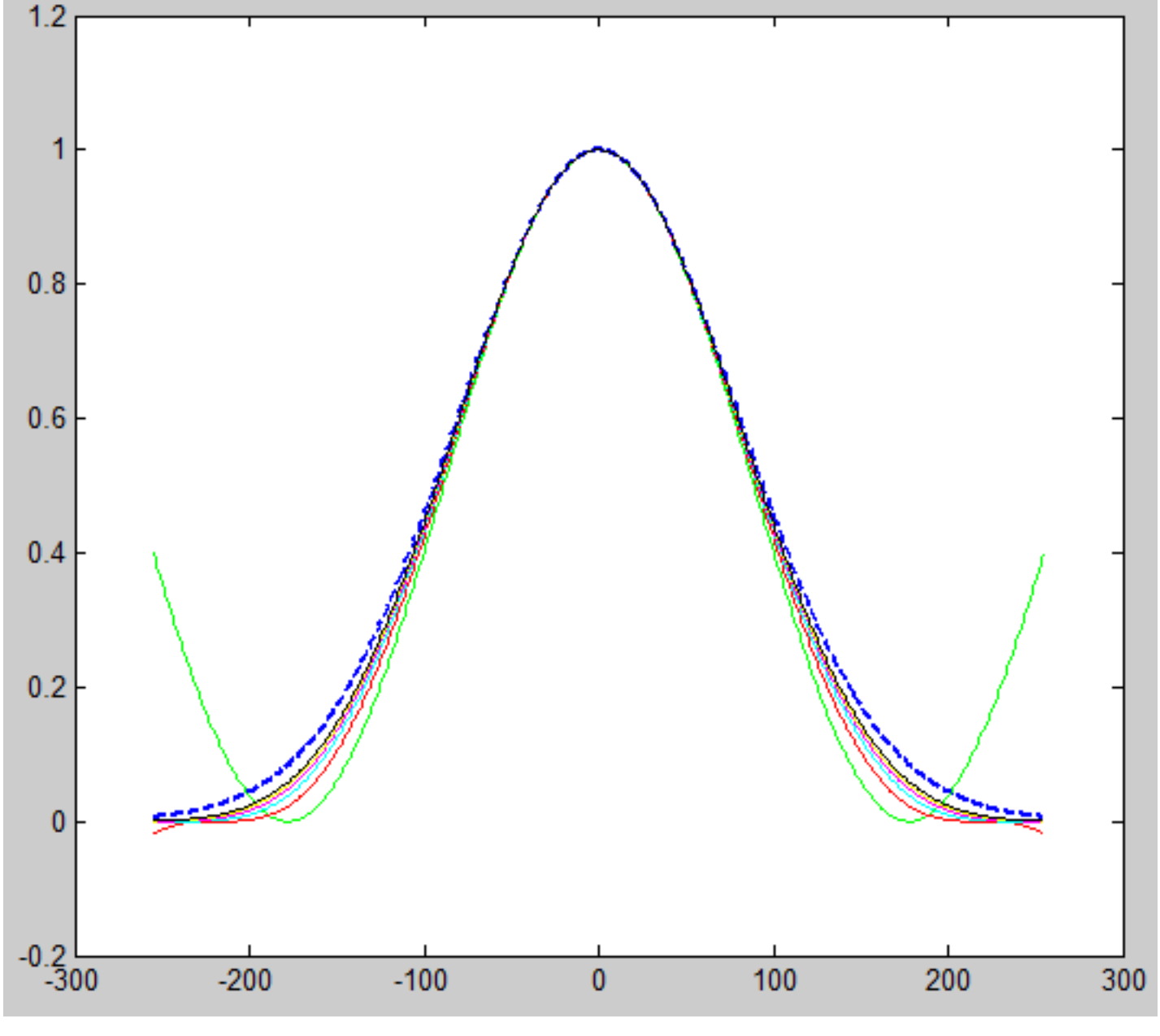
即可，此时要求

$$N>\rho^{-2}=\gamma\sigma^{-2}$$

;

式6中，最右侧部分即为高斯函数，此时说明，可以用 Raised cosines 函数来近似的模拟高斯函数，我们用一段matlab函数来验证该结果：

```
clc;
T=255;
Delta =80;
Gamma = pi/(2*T);
Rho= Gamma * Delta;
Color = ['b','g','r','c','m','y','k'];
x=-T:T;
y1=exp(-x.^2/(2*Delta*Delta));
plot(x,y1,'--b','LineWidth',2);
hold on;
for k=2:7
    y2= cos(Gamma .* x/ (Rho * sqrt(k))).^(k);
    plot(x,y2,Color(k));
end
```



从左图的曲线分布可见，N=2(绿色），N=3(红色）两条曲线明显不符合我们的定义域的要 求，分别出现了非单调递增和负值得情况。之后随着N的不断增大，曲线越来越接近高斯分布曲线（蓝色曲线)。 这从实际的角度证明了公式6的正确性。

3、推导

将公式（4）带入公式（6）中，得到：

$$\phi(s)=\exp(\frac{-s^2}{2\sigma^2})\approx\left[\cos(\frac{\gamma s}{\rho\sqrt{N}})\right]^N=\sum_{n=0}^{N}2^{-N}\binom{N}{n}\exp(-i(2n-N)\frac{\gamma}{\rho\sqrt{N}}s$$

(7)

将公式（7）带入原始的双边滤波的公式（1）中，有：

$$f(x)=\eta^{-1}\int_{\Omega}w(y)\phi(f(y)-f(x))f(y)dy$$

$$=\eta^{-1}\int_{\Omega}w(y)(\sum_{n=0}^{N}2^{-N}\binom{N}{n}\exp(i(2n-N)\frac{\gamma}{\rho\sqrt{N}}(f(y)-f(x)))f(y)dy$$

$$=\eta^{-1}\sum_{n=0}^{N}2^{-N}\binom{N}{n}\exp(-i(2n-N)\frac{\gamma}{\rho\sqrt{N}}f(x))*\int_{\Omega}w(y)(\sum_{n=0}^{N}2^{-N}\binom{N}{n}\exp(i(2n-N)\frac{\gamma}{\rho\sqrt{N}}f(y)))f(y)dy$$

$$=\eta^{-1}\sum_{n=0}^{N}2^{-N}\binom{N}{n}((\cos(\theta)+i\sin(\theta))*\int_{\Omega}w(y)(\cos(\beta)+i\sin(\beta))f(y)dy)$$

$$=\eta^{-1}\sum_{n=0}^{N}2^{-N}\binom{N}{n}((\cos(\theta)+i\sin(\theta))*(\int_{\Omega}w(y)\cos(\beta)f(y)dy+i\int_{\Omega}w(y)\sin(\beta)f(y)dy))$$

(8)

其中：

$$\theta=-(2n-N)\frac{\gamma}{\rho\sqrt{N}}f(x)$$

$$\beta=(2n-N)\frac{\gamma}{\rho\sqrt{N}}f(x)$$

式（8）的第三行利用了前面基础理论中的第三条：分配率。注意式中的积分是对y积分，因此可以把f(x)相关部分提取到积分符号的外围。

同样，对于η,可以推导得到：

$$\eta = \sum_{n=0}^{N} 2^{-N} \binom{N}{n} ((\cos(\theta) + i\sin(\theta)) * (\int_{\Omega} w(y)\cos(\beta)dy + i\int_{\Omega} w(y)\sin(\beta)dy))$$

(9)

注意（8)和(9)两式中后的后半部分，比如

$$\int_{\Omega} w(y)\cos(\beta)dy$$

这个，可以看出这个实际上就是对cos(β)这幅图像进行高斯模糊，而高斯模糊可以通过FFT或者回溯算法快速O(1)实现，这样两式8/9两式就分别转换为对f(y)cos(β)、f(y)sin(β)、cos(β)以及sin(β)图像进行一系列高斯模糊的过程了。

至此，所有的推导工作完成，那么我们在理一下算法的执行步骤吧：

（1）：已知条件：输入图像f(x),动态范围[-T,T],空域和值域方差σ_s、σ_r。

（2）：设定γ = Pi / 2T,ρ=γσ_r,N=1/(γσ_r*γσ_r)；

（3）: for (0≤n≤N),获取f(y)cos(β)、f(y)sin(β)、cos(β)以及sin(β)所对应的图像数据（浮点类型）；

（4）：利用O(1)高斯模糊算法对上述四个图像数据进行标准差为σ_s的高斯模糊并累计。

（5）：对累加后的数据进行除法操作，获得最终的结果图像。

注意：式8和式9中的乘法最后会有虚部的数据出现，在处理时可以直接丢弃掉。

三、实现及效果

以上算法在论文Fast O(1) bilateral filtering using trigonometric range kernels中有着较为详细的论述，论文中还给出了**Java**代码实现的链接，但是该链接已经失效，需要JAVA代码做参考的可从此处下载：BilateralFilter-src.rar，其中的BilateralFilter_.jar可在ImageJ中作为插件加载，而这篇论文的对应代码在解压后的bilateralfilterinstant文件夹中。注意，这个ImageJ的插件写的似乎有问题，运行时点plugins-->BilateralFilters-->Bilateral Filter Instant 后弹出的对话框中，不要勾选多线程才能对输入的任意参数进行处理，否则图像无任何反映。

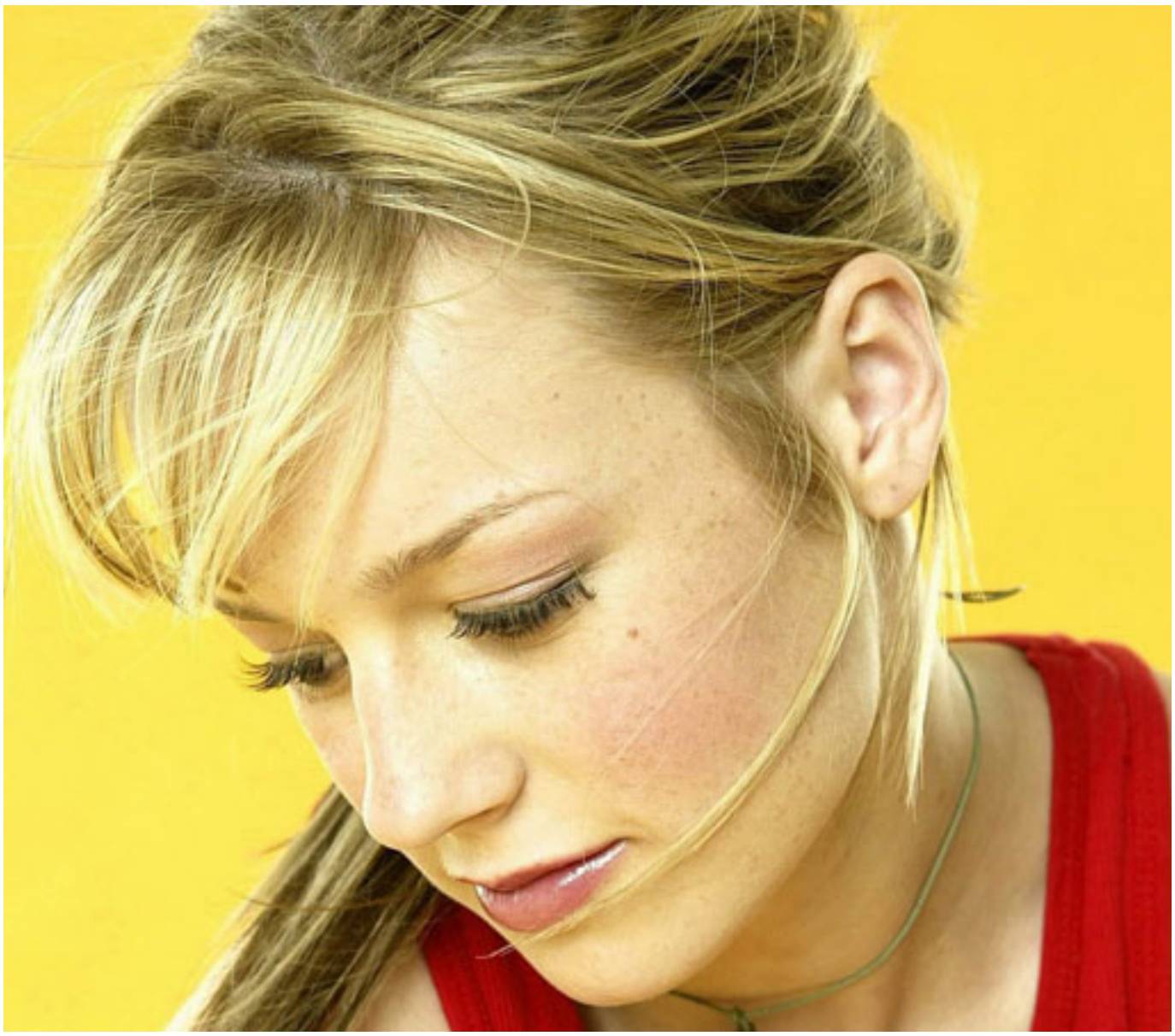
在第三步和第四步的处理，N+1次循环之间时没有任何关系的，因此，只要内存许可，各循环之间可以并行的执行，这对于现在的2核和4核的CPU的有一定的意义，不过相比GPU来说，可能意义更大吧。

按上述过程编制代码，测试效果测试如下:

（1） 去除噪音

lena图增加标准差为20的高斯噪音，使用σ_s=20、σ_r=40双边滤波后的结果

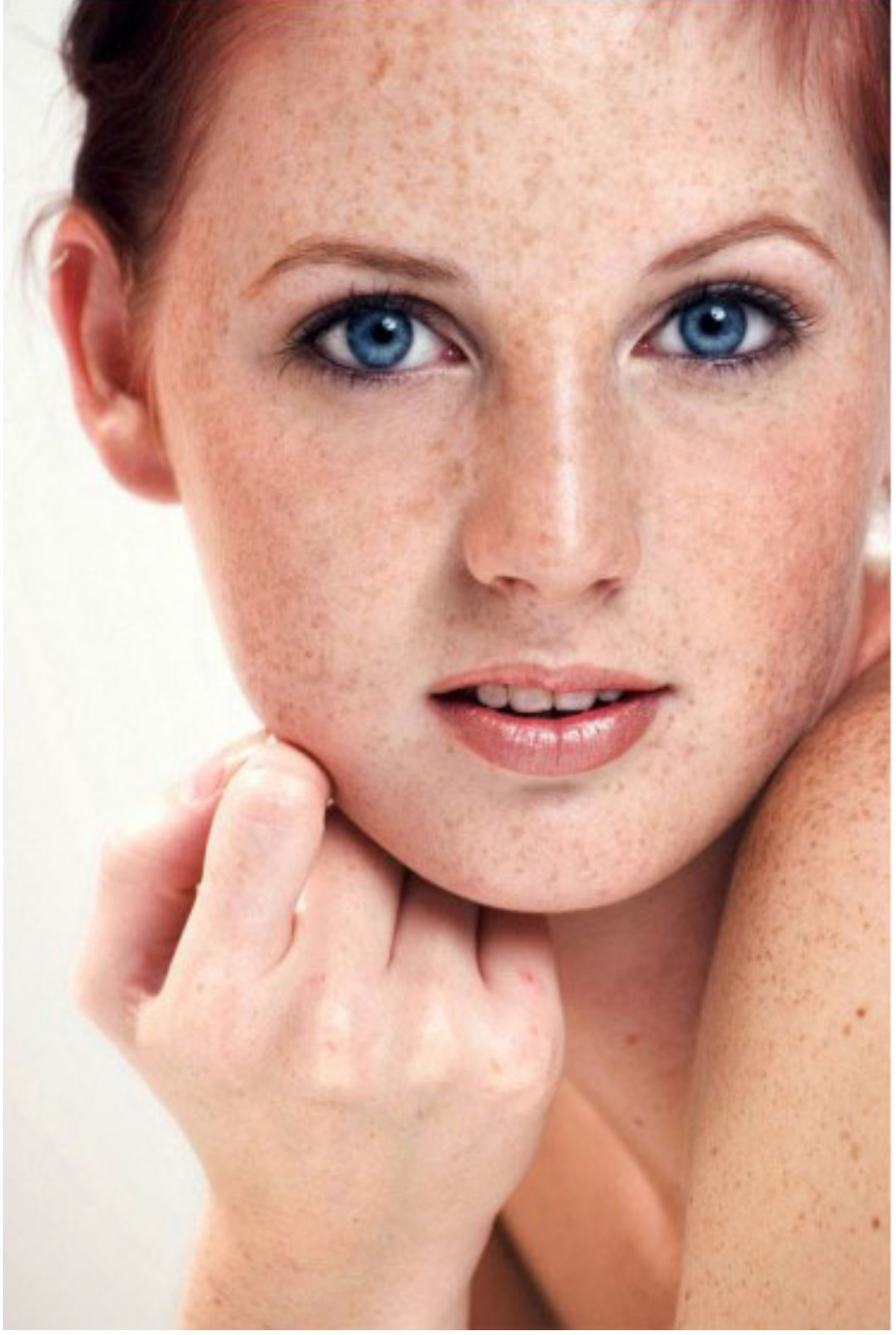
（2）普通图像的边缘保持结果



带有噪音的美女图

1

使用σ_s=10、σ_r=35双边滤波后的结果



带有噪音的美女图2

使用σ_s=15、σ_r=25双边滤波后的结果　　使用σ_s=10、σ_r=35双边滤波后的结果

美女1 细腻的头发在滤波后依然有着飘然的感觉，而面部的斑点也已经悄然消失。美女2的面部也得到了恰当的护理，而其湛蓝的双眼依旧是那么有神。

四、效率分析及进一步优化

很明显，上述算法的执行时间直接依赖于N的大小，而从相关推导公式中看，N的值直接取决于T和σ_r的大小，T越大,N越大，σ_r越大，N越小。当T固定时，比如为255时，取不同的σ_r对应的N值如下表所示：

σ_r	5	10	20	30	40	60	80	1000
N	1053	263	66	29	16	7	4	3

上表中，可以发现，当σ_r小于20时，所需要的循环次数N极具增加，当N超过100时，可以认为这个算法已经不再具有优化的意义了，可能比原始的算法还慢。这个从原理上也很好解释。当σ_r比较小时，高斯函数的曲线在中心线附近急剧下降，从而需要更多的三角函数来逼近他。

因此，进一步的优化需要从T的取值以及N的方面予以考虑。

我们观察到，在对值域高斯核函数进行计算时，使用的是f(y)-f(x),而不是f(y),通常情况下，一副图像的f(y)中所有值的最大值可能为255，但是某个区域内的f(y)-f(x)在全图中的值却不一定为255，如果这个值小于255，那么在上述算法中使用的T值就可以缩小，从而减少N的大小，比如当σ_r为10时，T取255，由上表可知N=263，而如果T为210，则大约只需要
263*（210/255）^2 = 178 次循环，几乎减少了100次。

那么一副图像的实际的T值打开是什么个范畴呢，使用标准的Lena图做了个测试，在不同的取样半径下获得的结果如下表：

r	1	3	5	10	15	20	30
T	153	205	208	210	211	215	215

这说明在很多情况下T值确实小于255，即使取样半径比较大。

这里则涉及到一个平衡问题。计算实际的T值一般情况下会获得小于255的结果，这有利于减小N，从而降低程序的执行时间，但是这个计算的过程本身也需要时间，如果这个时间大于其带来的好处，则这个改进就是退步。幸好，在很久以前，关于指定半径内的最大值算法就已经有了O(1)的快速算法， 其执行时间一般要小于进行一次本例中这种循环的时间。所以这个改进是值得的。

那么我在看看小 σ_r时大N的问题，当σ_r比较小时，我们观察其分布曲线，如下图：

σ_r = [1 3 5 10 15]时的曲线

由上图可以看出小σ_r时，曲线在中心线附近迅速衰减，理论表明这个距

离为[-3σ_r,3σ_r],在此之外的值可以忽略不计，因此，那些对最终结果没有什么贡献的循环就完全可以舍弃，这部分的理论推导可以详见论文 Acceleration of the shiftable O(1) algorithm for bilateral filtering and non-local means 。

我们知道，Non-Local算法在很大程度是双边模糊的扩展，只是其值域的相似度函数更加复杂，不是简单的f(y)-(f(x)那么简单了，而是和f(y)和f(x)的领域有关，因此直接的Non-Local实现理论上比双边滤波还要耗时，上面介绍的这种优化方式在后面这篇论文里提到也是可以用于Non-Local的，有兴趣的朋友可以自己去研究下。

五：小结和展望

可以看到，本文的这种优化方式实际上是利用Cos函数去毕竟高斯函数，在代码层次上，需要(N+1)*4此高斯模糊，而由上面相关表格可以看到，N的数值一般都在10以上，因此，至少要执行44次高斯模糊，这还不包括获取需要高斯模糊的数据部分以及最后的滤波结果获取。即使高斯模糊在高效，比如对于600*400的彩色图，5ms足也，那么执行双边模糊保守估计也要400ms左右，因此这个算法说实在的效率还是不行。

这两篇文章分别是2011年及2012年发表的，应该是代表着目前比较先进的技术，我在网上经常看到有人说双边滤波可是实时，实在是不晓得那些高人用的是什么理论，抑或是什么超级机器。

同我之间的一篇博文中双指数边缘平滑滤波器用于磨皮算法的尝试 提到的Beeps边缘保留算法相比，这里的速度就要慢很多了，而两者的效果相比基本上差不多，所以实在很纠结。

希望有更好的方法用于该算法。