Image Spatial Diffusion on GPUs

Lihua Zhu, Charles Wang
Cooperate Research
Thomson Inc
Beijing, China
{lihua.zhu, charles.wang}@thomson.net

Guangfei Zhu, Bo Han, Heng Wang
Peking University,
Beijing, China

Peijie Huang, En-Hua Wu
Institute of Software
China Academy Science
Beijing China

***Abstract*—Image spatial diffusion targets on blurring small discontinuities while sharpening distinct edges. We propose two GPU-based methods to efficient generate content-aware spatial diffused images. In the first method, we apply an enhanced bilateral filter on the input image. Our filter adopts a spatial weight kernel in the form of an inverted Gaussian. This Inverted-Gaussian-Spatial Bilateral filter (IGSB) can remove small spots in large smooth areas more efficiently than Gaussian bilateral filter that are commonly used. In the second method, we present a fast approximation of mean-shift algorithm on GPUs. Considering the parallel nature of GPUs, the process of mean-shift kernel density estimation is modified and improved to achieve satisfied color image diffusion effects.**

I. INTRODUCTION

Image spatial diffusion aims at blurring small discontinuities while preserving edges. Although there is a large body of researches devoted to this topic, we only focus on its application to Non-Photorealist Rendering (NPR) area.

Many techniques that stylize images or videos, belonging to the NPR area, have been proposed in recent years. We observe that a good image spatial diffusion step is usually a crucial part of these methods [1-3]. DeCarlo et al. [1] primarily focused on making images easier or faster to understand, as well as catered for artistic effects. They adopted the mean-shift segmentation algorithm proposed by Comaniciu and Meer [4] to segment images, and then filled in the segmented regions with their average colors. Wang et al. [2] described a system for transforming an input video into a highly abstracted, spatiotemporally coherent cartoon animation. Similar to DeCarlo's method, they also adopted mean-shift segmentation, but they applied it to a three-dimensional spatiotemporal video volume.

Primarily, stylization technique was applied for artistic purposes. In movie industry, "Waking Life" (2001) and "a Scanner Darkly" (2006) provided a fresh and amazing visual style by converting natural video into cartoon animations. As far as we know, producing such movies costs huge amount of money and time. It would bring great significance if the stylization process could be automatically or semi-automatically generated by the computer. Besides movie industry, as stylized images comprise less details and larger areas of uniform pixels, it can also be used for efficient visual telecommunication with high compression ratio, because a stylized image comprises smaller high frequency portion than the corresponding natural image. So a decoded stylized image comprises less annoying artifacts than the natural image that are compressed using the same compression ratio.

Therefore, how to achieve efficient stylization results is a valuable direction in our research. To extended DeCarlo's work from image to video, Winnemöller et al. [3] proposed a real-time video stylization framework. For spatial diffusion, they used a separated approximation to bilateral filter, implemented on GPU for acceleration purpose.

Generally, among all spatial diffusion approaches, both bilateral filter and mean-shift algorithm are most frequently adopted in image/video stylization methods. However, they have their own strengths and limitations respectively. The main strength of bilateral filter lies in the comparatively low computational cost. Because its separated approximation (proposed by Pham et al. [6], adopted by Winnemöller et al. [3]) can additionally lower the computational cost. But the drawback of bilateral filter is that the aesthetic effects it can achieve are not as good as those achieved by mean-shift segmentation algorithm which on the other hand brings in heavier computational burden.

For their powerful parallel computing ability and fascinating performance growth rate, GPUs are more attractive as coprocessors for various general-purpose computation problems. To tackle the drawbacks of bilateral filter and mean-shift algorithm mentioned above, we propose two GPU-based methods that generate a spatial diffused image from an input natural image. In the first method, we apply on the input image with a bilateral filter by a spatial weight in the form of inverted Gaussian. This Inverted-Gaussian-Spatial Bilateral (IGSB) filter can remove small spots in large smooth areas more efficiently than the classical bilateral filter. In the second method, we propose a fast approximation to the traditional mean-shift algorithm. To better utilize GPU's computational power, for all pixels, mean-shift vectors are iteratively calculated for a fixed number of times. This modification makes it possible to generate good diffusion effects with much higher run-time speed.

II. INVERTED-GAUSSIAN-SPATIAL BILATERAL FILTER

A. Algorithm

For an input image I and output image H, the general form of bilateral filter could be defined as follows:

978-1-4244-2342-2/08/$25.00 ©2008 IEEE.

$$H(c,\Omega)=\frac{\sum_{a\in\Omega} I(a)g(I(a)-I(c))f(P(a)-P(c))}{\sum_{a\in\Omega} g(I(a)-I(c))f(P(a)-P(c))} \quad (1)$$

In this formula, a is a neighboring pixel in window Ω around pixel c. $P(\cdot)$ returns the position of a pixel and $I(\cdot)$ returns the intensity value of a pixel. $f(\cdot)$ is spatial filter function and $g(\cdot)$ is intensity filter function. Both $f(\cdot)$ and $g(\cdot)$ are typically Gaussian function.

Tomasi et al. [5] gave bilateral filter a comprehensive analysis and figured out that the preservation of crisp edges is mainly due to the intensity filter, while the spatial filter takes a major role in the regions where the intensity distribution is flat. Based on this fact, we design an Inverted-Gaussian-Spatial Bilateral filter (IGSB) to remove small spots in large smooth area more effectively.

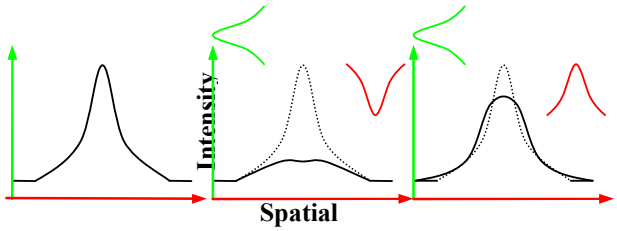

Figure1. Left: the original signal; Middle: signal smoothed by IGSB; Right: signal smoothed by GSB. The green curves indicate the intensity filter, and the red curves indicate the spatial filter. The comparison between Middle and Right shows that the spot is removed more efficiently by IGSB than by GSB.

Considering a small spot signal inside a large smooth area (Fig.1, Left), according to Tomasi's conclusion, if it is surrounded by outliers outside the intensity filter window, it will be recognized as part of a crisp edge, so it will be left almost unchanged. Otherwise, the spatial filter will take a dominant role for non-spot areas. Traditional Gaussian-Spatial Bilateral filter (GSB) assigns heavier weight to the nearer pixels. The signal, after smoothed by GSB, is illustrated in Fig.1, Right. However, we notice that for the goal of effectively removal of the spot, the pixels that are farther should be assigned heavier weights than the nearer ones. Based on this observation, we design IGSB, as shown in Fig 1, Middle.

We define IGSB as follows:

$$H(c,\sigma_r,\sigma_d)=\frac{\sum_{a\in\Omega} I(a)e^{-\frac{1}{2}(\frac{|I(a)-I(c)|}{\sigma_r})^2}(C-e^{-\frac{1}{2}(\frac{E(P(a),P(c))}{\sigma_d})^2})}{\sum_{a\in\Omega} e^{-\frac{1}{2}(\frac{|I(a)-I(c)|}{\sigma_r})^2}(C-e^{-\frac{1}{2}(\frac{E(P(a),P(c))}{\sigma_d})^2})} \quad (2)$$

In this formula, $C-e^{-\frac{1}{2}(\frac{E(P(a),P(c))}{\sigma_d})^2}$ is the spatial weight, which takes a curve of inverted Gaussian. C is a freely selectable constant. Good results were achieved with $C=2$. σ_d is related to the blur radius, determining the size of window Ω. Increasing σ_d generally results in more blurring1. Function $E(\cdot)$ returns the Euler distance between two pixels. $e^{-\frac{1}{2}(\frac{|I(a)-I(c)|}{\sigma_r})^2}$ is the intensity weight. σ_r is related to the extent of the intensity filtering.

B. GPU-Based Implementation

The graphics API is called to draw a render target which is in exactly the same size with '*I*'. '*I*' can be treated as a texture, so the pixels in '*I* 'can be processed in the pixel shader, which is a stage of graphics pipeline. In pixel shader, program can be executed in parallel with the pixel level, so it is generally called a "pixel program". Here, we write a pixel program to demonstrate the workflow of our IGSB filter algorithm.

Fig.2 gives the pseudo code of IGSB pixel program. '*uv*' are the current texture coordinates. '*inputImage*' is a texture sampler which has been bounded to an input image. '*deltad*' and '*deltar*' correspond to σ_d and σ_r respectively.

```
void bilateral_ps( in float2 uv, out float3 color,
                   uniform sampler inputImage)
{
    float3 cur = tex(inputImgae,uv)//get the filtering pixel
    float4 sum = 0, sum.rgb = cur,  sum.a = 1
    For each pixel  in the window
      //make a offset and then get a pixel from inputImage
       float3  p  = tex( inputImage,uv +offset)
      // spatial weight
       float w = 2-exp(dot(offset, offset)/(-2*deltad))
       //intensity weight
        w=w* exp(dot(cur-texel,cur-texel)/(-2*deltar)
        sum.rgb += p * a, sum.a += a
    EndFor
    color = sum.rgb/sum.a
}
```

Figure2. Pseudo Code of IGSB's Pixel Program

III. Mean-Shift Approximation

A. Algorithm

Given an image represented in CIE-Lab color space, for a center pixel $c_n=(x_c,y_c,L_c,a_c,b_c)_n$ with spatial components x and y and color components L, a and b, a mean-shift vector $m_{h,G}(c_n)$ is calculated as follows:

[1] Tomasi et al. [5] pointed out that one paradoxical effect happened when the value of σ_d was large enough.

$$m_{h,G}(c_n) = \frac{\sum_{s\in\Omega} s * g_h(s, c_n)}{\sum_{s\in\Omega} g_h(s, c_n)} - c_n \quad (3)$$

$$g_h(a, c_n) = g(\left\| \frac{c_n - a}{h} \right\|^2) \quad (4)$$

wherein $s = (x_s, y_s, L_s, a_s, b_s)$ is a pixel in a set of pixels Ω around center pixel c_n. $g_h(\cdot)$ is a profile of a kernel function $G(\cdot)$: $G(x) = \alpha * g_h(\| x \|^2)$ with a normalizing constant α. $g_h(\cdot)$ depends on parameter h .

Comaniciu and Meer [4] took the kernel function as:

$$k_{hs,hr} = \frac{C}{hs^2 hr^p} k(\left\| \frac{x^s}{hs} \right\|^2) k(\left\| \frac{x^r}{hr} \right\|^2) \quad (5)$$

wherein hs and hr are employed kernel bandwidths in spatial domain and range domain respectively. In our implementation, we use Gaussian function as the profile $k(\cdot)$.

After calculation of the mean-shift vector, the center pixel is shifted by the mean-shift vector spatially and in color dimensions. Calculation and shifting is repeated until the mean-shift vector equals to zero. In the final mean-shift filtered image, the color components of convergence $(L_{con}, a_{con}, b_{con})$ are assigned to a pixel p_{after} at the position (x_{c0}, y_{c0}) of the initial center point c_0 :

$$p_{after} = (x_{c0}, y_{c0}, L_{con}, a_{con}, b_{con}) \quad (6)$$

This mean-shift filtering is applied to all the pixels in the input image. From the process described above, we can see that mean-shift filter act as a global manner compared with bilateral filter, i.e., in bilateral filter, a pixel is only affected by the pixels in its vicinity, while in mean-shift filter, a pixel may be affected by pixels which are far away from it. Bilateral filter's locality makes it suitable for GPU implementation, for GPU's strength lies in processing data locally in parallel. But mean-shift doesn't have this locality. Specifically, we can not know which pixels in the input image will affect the current pixel until the whole process terminates.

To solve this problem, the number of iterations (denoted by r in the below text) for calculating mean-shift vectors for all pixels is fixed. Assuming that the potentially related pixels are localized in the neighborhood, the instructions executed on all pixels can be unified. So it can be suitable for the GPU pipeline. Since only the number of iterations has been fixed, and the other modules of mean-shift algorithms are still unchanged, so our GPU-based mean-shift algorithm almost approximates to the traditional mean-shift algorithm. From the computation angle, the GPU-based mean-shift would like more computation than Comaniciu's mean-shift algorithm. Our GPU-based algorithm terminates after a fix number of iterations which is independent of the mean-shift vector.

B. GPU-Based Implementation

The process of GPU-based mean-shift algorithm is illustrated in Fig.3. A texture in GPUs only has four RGBA channels, but, according to formula (3), five variables should be stored during the mean-shift process. We take Multi-Render Target (MRT) technique provided by modern GPUs to store five variables. Fig.3 shows that the spatial vector and color vector are separately stored into a spatial vector texture and color vector texture. RG channels are used in spatial vector texture, and RGB channels are used in color vector texture. The mean-shift vectors are updated in the pixel shader iteratively. Meanwhile, another two additional textures are utilized as alternative textures for swapping between the render target and data storage.

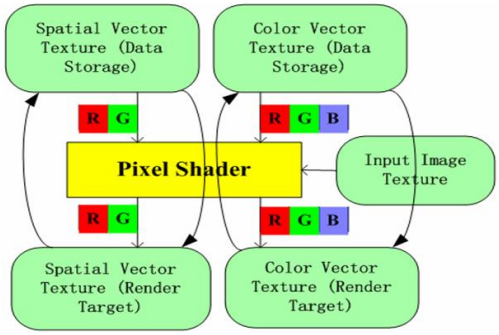

Figure3. Process of GPU-Based Mean-Shift Algorithm

IV. EXPERIMENT RESULTS

Our experiment environment is performed on an Intel Core 2 CPU 4400 2.0G with an Nvidia Geforce 8500 GT. Our programs are implemented with OpenGL2.0 and Cg 1.5.

Table I shows the time consuming between GPU and CPU. Our GPU based solutions for bilateral filter have ranged from 25 to 30 times performance improvement compared to the CPU solution with same configurations. As for the mean-shift algorithm when r is smaller than 140, our GPU based solutions make also eight times speed-up compared to the CPU based solution with same configurations.

Table I Time consuming between CPU and GPU

	Football (352x288)		Lake (512x512)	
	CPU	GPU	CPU	GPU
Bilateral	0.15s	0.006s	0.3s	0.01s
Mean-Shift	3.52s	0.46s	6.94s	0.8s

Fig.4 compares the filter results between IGSB and GSB on a football image with resolution 352x288. σ_r and σ_d are set to 4.5 and 2.5 respectively. The filtering is performed in CIE-Lab color space. Comparing Fig.4 (a) and Fig.4 (b), we can see that our IGSB can give a much cleaner result comparing to GSB. Fig.4 (c) and Fig.4 (d) give the detected edges from Fig.4 (a) and Fig.4 (b) respectively. The comparison between Fig.4 (c) and Fig.4 (d) manifests the difference between Fig.4 (a) and Fig.4 (b).

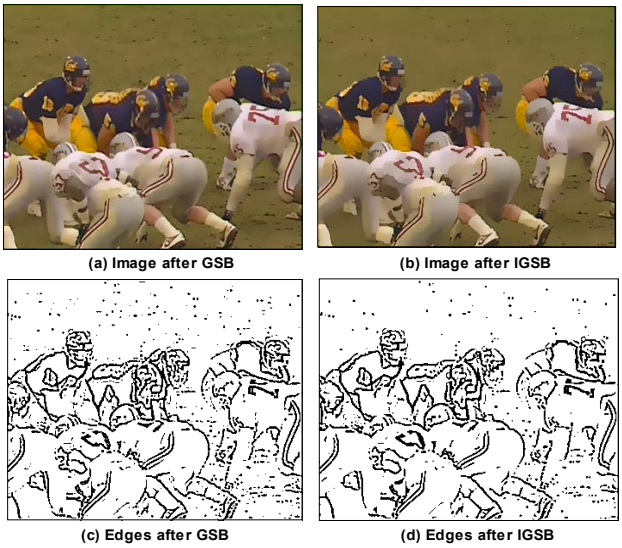

Figure4. GSB and IGSB applied to an frame in "football" sequence

Fig.5 (c) and Fig.5 (d) shows the filtered "lake" image (512X512) by our program. We choose $hs = 10$, $hr = 50$ for our algorithm and the window size is 20x20. The filtering is performed in CIE-Lab color space. Fig.5 (b) is the filtered result by Comaniciu and Meer's system "EDISON" [7].

Fig.5 (c) and Fig.5 (d) exhibit a stronger NPR style; meanwhile, they still well preserve the silhouettes of the objects. Fig.5 (d) has more large smooth areas than Fig.5 (c), because more iteration numbers have been taken to generate a better diffusion result in the algorithm of Fig.5 (d).

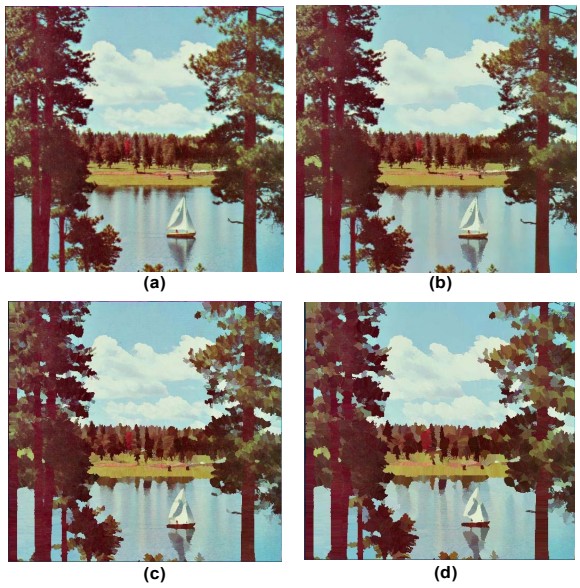

Figure5. (a) Original Lake Image; (b) Mean-shift result by Comaniciu and Meer (hs=7, hr =6.5) ; (c) Mean-shift Result by our GPU-Based alogrithm, r=4;(d) Mean-shift Result by our GPU-Based alogrithm, r=20.

The performance of our program is independent of the content but closely relates with the resolution of the image. Fig.6 gives the performance data of images with different resolution (256X256, 512X512, 1024X768) and different iteration value *r* (40, 100, 140, 150, 160, 200).

From Fig.6 we can see that when r is smaller than 140, our program can finish mean-shift filtering in less than one second (EDISON spent 6.94s to process lake image). When r is larger than 140, the time's ascending rate is getting sharper and sharper, which indicates that GPU is now the performance bottleneck. Notice that window size is another factor that will affect the performance. In the experiment, it is fixed as 20X20.

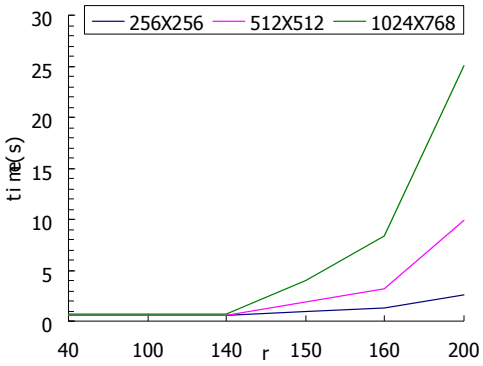

Figure6. Performance of GPU-version Mean-Shift Filtering

V. CONCLUSION

In this paper, we present two GPU-based algorithms for image spatial diffusion, which have been widely used in efficient image/video stylization in NPR area. We have respectively designed an IGSB filter for obtaining better diffusion effect and a fast approximation of mean-shift filter algorithm which can be fit on integrating into GPUs and then can bring the better computation efficiency. To improve the image diffusion visual effect, we take IGSB to remove small spots in large area more effectively than commonly used bilateral filter. While the GPU-based mean-shift filtering algorithm runs much faster than traditional mean-shift methods, at the same time, it can make the same good diffusion effects as the traditional mean-shift algorithm.

REFERENCES

[1] D. Decarlo and A. Santella, Stylization and abstraction of photographs. In Proceedings of ACM SIGGRAPH 2002, 769-776.

[2] J. Wang, Y. Xu, H-Y. Shum and M. Cohen, Video toning. ACM Trans. Graph., 23(3): 574-583, 2006.

[3] H. Winnemöller, S.C.Olsen and B. Gooch, Real-time video abstraction. ACM Trans. Graph., 25(3): 1221-1226, 2006.

[4] D. Comaniciu and P. Meer,: Mean shift: a robust approach toward feature space analysis.IEEE Trans. on PAMI (2002) 603-619.

[5] C. Tomasi and R. Manduchi, Bilateral filtering for gray and color images. In Proceedings of ICCV, 839-846. IEEE, 1998.

[6] T.O.Pham and L.J.V.Vliet, Separable bilateral filtering for fast video preprocessing. In IEEE Internat. Conf. on Multimedia & Expo., 2005, CD1-4.