THE REMOTE-SENSING IMAGE FUSION BASED ON GPU

Jun Lu*, Baoming Zhang, Zhihui Gong, Ersen Li, Hangye Liu

Zhengzhou Institute of Surveying and Mapping, 450052 Zhengzhou, China – lj2000hb45@126.com

Commission VII, WG VII/6

KEY WORDS: GPU, FBO, Rendertotexture, Fragment program, IHS transform, DWT

ABSTRACT:

Along with computing capability of the Graphic Processing Unit (GPU) getting more powerful, GPU is widely applied to the general purpose computing not just restrict to graph manipulation. Remote sensing image data can be parallel processed in many of the image fusion arithmetic. The fusion arithmetic in spatial domain is mapped to the SIMD computing way on GPU. This paper realized the product fusion, ratio fusion, high-pass filtering fusion and weighted fusion using GLSL in spatial domain, and accelerated the speed of fusion computing using RTT (Render to Texture) technology. This paper focused on the arithmetic in transformed domain, realized IHS transform fusion and DWT (Discrete Wavelet Transform) fusion. The IHS forward transform and inverse transform are mapped to two fragment shading processes, parallel computing and outputting of the 3 component in both transform processes is realized using MRT (Multiple Render Targets) technology. 2D DWT is divided into two steps of 1D DWT. An indirect address texture is created for every transform step and the transform of each level is based on the result stored on the texture of the last level. A FBO is set for every image to be fused to restore intermediate data and to do data exchange. The result shows that for the same fusion algorithm, the fusion images are identical using the two different methods, but the processing velocity in GPU implementation is obviously faster than the CPU implementation, and with the fusion algorithm getting more complicated, the fusion images getting bigger, the advantage of the velocity is more obvious in GPU implementation.

1. INTRODUCTION

The fast development of remote sensing technology makes the rapid increase of image data we get. The image fusion technology provides an approach to get the needed information from image data. Many algorithms of remote sensing image fusion process pixels of the image in the same way, but the programming model based on CPU is generally serial and can process only one datum at one time, which doesn't make use of data parallel. The modern graphics processing unit has powerful parallel computing capability, and provides common functionality for both vertex and pixel shaders. In GPU, remote sensing image can be processed parallel and the time spent on image fusion will be shortened.

2. GRAPHICS PROCESSING UNIT

2.1 The GPU Rendering Pipeline

The current GPU is called the "stream processor" because it has powerful parallel computing capability and extensive memory band width, all data in stream program model is called stream. Stream is an ordered set which has the same data type. Kernel operates all of the steams, takes one or more streams as the input data and generates one or more streams as the output data. There are two kinds of programmable processors in GPU: vertex processor and fragment processor. Vertex processor deals with the vertex streams which constitute the geometry models. The computer graph indicates a 3D object by triangulation network. As an illustration to the mechanism in GPU, we describe the rendering of a texture-mapped polygon. The user first define the 3D position of each vertex through the API in graphics library (OpenGL or DirectX).The texture coordinate associating with each vertex is also defined at the same time. These vertices are then passed to the vertex engine for transformation. For each of them, a vertex shader (user-defined program) is executed. The shader program must be SIMD in nature, i.e. the same set of operations has to be executed on different vertices. Next, the polygon is projected onto 2D and rasterized (discretized) to framebuffer. At this stage, the fragment engine takes place. For each rasterized pixel, a user-defined fragment shader is executed to process data associated with that pixel (fragment). Again, the fragment shader must also be SIMD in nature. In the fragment shader, the associated texture can be fetched for processing. To utilize the GPU for 2D array (image), we can simply store the 2D data on a texture map. Note that each data can be 32-bit floating-point. We then define a rectangle with this texture map mounted on it, and render this texture-mapped polygon to the framebuffer.

2.2 Render To Texture

In traditional GPU rendering pipeline, the destination of the rendering computing is frame buffer which is a part of the video card memory, the image data stored in frame buffer will display on the screen in real time. The window size of the frame buffer should be the same as the texture size is we use the traditional GPU rendering pipeline to do the image processing, and the image size can be operated in one time is restricted to a certain range (the packed texture size should be smaller than the screen size), otherwise it may cause distortion because of the resampling on the image. We use the FBO to realize render to texture. The maximal size of the texture supported by consume level GPU is 4096×4096 (the size is bigger in new GPU) which has far exceeded the screen window size, and the off-screen rendering mode is accelerated by the hardware[1].

* Corresponding author.

3. PARALLEL DATA INPUTING OF IMAGE

The different arrangement of remote sensing image data in system memory affects the interior texture format and texture size, thereby affects the computing speed in GPU of the fusion programs. The remote sensing images to be fused are usually low spatial resolution multispectral images and hyperspectral images (or the high spatial resolution panchromatic image). One memory block usually stores one spectral band data (sometimes stores three spectral band data which calls false color composite image), or one hyperspectral image, or one high spatial resolution panchromatic image. This paper designed two packing ways of remote sensing image data to utilize the parallel processing characteristic in data-level of remote sensing image fusion arithmetic on GPU.

The first way is to rearrange the three multispectral images and the hyperspectral image (or the high spatial resolution panchromatic image) in memory. In packed image every four pixels form a cell to storage corresponding pixels in the four images to be fused. The height of the packed image is the same as the original, but the width of the packed image is 4 times as the original. Then load the packed image data to the texture memory and set the internal format of the texture to GL_FLOAT_RGBA32_NV, set the texture type to GL_TEXTURE_RECTANGLE_ARB, so each of the 4 components of every pixel occupies a 32 bit float space. In texture memory, the 4 channels of RGBA stores the 4 images to be fused respectively, figure 1 shows the packing process:

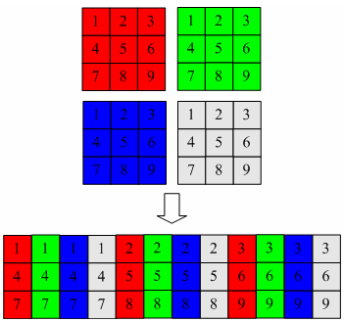

Figure 1. Load 4 images to 1 texture (4 channels of RGBA)

The second way is to load the each of the multispectral images and the hyperspectral image (or the high spatial resolution panchromatic image) to a texture separately; the internal format of the texture is set to GL_FLOAT_RGBA32_NV. There are four pixels in one texel, it means the image data is compressed in texture memory. By this way we can make full use of the four channels of every texel and the parallel computing capability of the fragment processor, greatly reduce the number of elements to be processed, and the remote image data needn't to be packed in system memory, figure 2 shows the packing process:

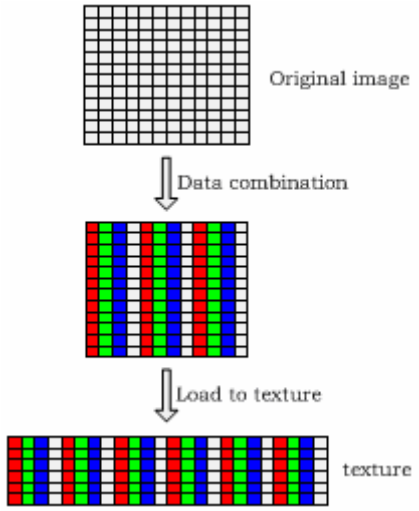

Figure 2. Load 1 image to 1 texture (4 channels of RGBA)

But if we adopt this 2D compressed storage mode, there may appear some blankness. So we have to try to reduce to blankness to assure the best texture size and reduce the texel number. We use the expressions as follows to get to texture size:
Height = floor(sqrt(N/4));

Width = ceil((double)(N/4)/(double)Height);

The first packing way reduce the times of texture fetching in fragment shaders, so the shading process is faster than that in the second way, but we should rearrange the image data in system memory, and will waste some CPU computing time. In the second packing way, we needn't to rearrange the image data in system memory but the texture fetching times in fragment shader is 4 times as in the first packing way. The data packing way should be chosen flexible in remote sensing image fusion and the effect of the design of the fragment programs by the data packing way. For those fusion arithmetic in spatial domain which is simple, the process of data rearrange in system should be avoid because the time cost in this step occupies a large proportion of the fusion time.

4. THE SPATIAL DOMAIN FUSION ON GPU

There are many arithmetic of multi-sensors remote sensing image fusion, which can be classified to spatial domain fusion and transformed domain fusion[2]. In spatial domain fusion we adopt some arithmetic to process the registered high resolution image and low resolution image in spatial domain to get the fusion image, there are mainly four kinds of fusion arithmetic in spatial domain: weighted fusion, product fusion, ratio fusion and high-pass filtering fusion.

The arithmetic of spatial domain fusion is relatively simple, the process of which in GPU is basically identical, the difference is just the fragment shader program. The process of spatial domain fusion based on GPU is as follows:

1) Pack the remote sensing image data. Generally the packing method is based on the image data format and we should try to reduce the work of the data rearrange.
2) Load the packed image data to texture and release the data in system memory.
3) Set FBO. Confirm the number and format of the texture bind to FBO according to the number and the format of the image we want to get through the fusion arithmetic.

4) Rendering by the fragment shader.
5) Download the texture which stores the rendering result to the system memory.

5. THE TRANSFORMED DOMAIN FUSION ON GPU

This paper mainly analyses IHS transform fusion and DWT fusion.

5.1 The IHS transform fusion on GPU

There are two kinds of color coordinate systems (or called color space) widely used in image processing: ① Color space comprised by Red, Green and Blue, which is called RGB space; ② IHS model, which is comprised by Intensity, Hue and Saturation. The transformation between RGB space and IHS space is called IHS transform[3]. The 3 components of IHS space which have their independency can be controlled separately and describe the color feature exactly. The process of IHS is as follow:

Normalize the original pixel value R_0、G_0、B_0:

$R=R_0/L$, $G=G_0/L$, $B=B_0/L$;

Where, L is the gray level of image.

In forward transformation, the values of I、H、S are calculated.
When $R=G=B$: $S=H=0$, $I=R$;
Otherwise:

$$\begin{aligned} I &= \frac{1}{3}(R+B+G) \\ S &= 1-\min(R,G,B)/I \\ h &= \arccos\left[\frac{R-G/2-B/2}{\sqrt{R^2+G^2+B^2-RG-GB-RB}}\right] \end{aligned} \tag{1}$$

When $G>=B$: $H=h$ and when G<B: $H=2\pi-h$.
Histogram matching, the grayscale of high resolution panchromatic image is stretched to make the average of its grayscale and summation of variance identical with the I component of IHS space.

In inverse transformation, the stretched high resolution image is regarded as new I component into calculation. The RGB values of original space are computed and the inverse transformation is as follow:

When $0 \le H < \frac{2}{3}\pi$:

$$\begin{aligned} R &= I(1+\frac{S\cdot\cos(H)}{\cos(\pi/3-H)}) \\ B &= I(1-S) \\ G &= 3I-R-B \end{aligned} \tag{2}$$

When $\frac{2}{3}\pi \le H < \frac{4}{3}\pi$:

$$\begin{aligned} G &= I(1+\frac{S\cdot\cos(H-2\pi/3)}{\cos(\pi-H)}) \\ R &= I(1-S) \\ B &= 3I-R-G \end{aligned} \tag{3}$$

When $\frac{4}{3}\pi \le H < 2\pi$:

$$\begin{aligned} B &= I(1+\frac{S\cdot\cos(H-4\pi/3)}{\cos(5\pi/3-H)}) \\ G &= I(1-S) \\ R &= 3I-B-G \end{aligned} \tag{4}$$

Firstly, according to packing method 2, RGB 3-bands multi-spectral image with low resolution is loaded into three textures, and then FBO is created and bound with 6 frame buffers, whose binding points are L_COLOR_ATTACHMENT0_EXT 、 GL_COLOR_ATTACHMENT1_EXT …… GL_COLOR_ATTACHMENT5_EXT. Each frame buffer is corresponding with an output texture.

Secondly, forward transformation is executed by a shader in one circulation. In this paper, shaders are implemented by OpenGL Shading Language (GLSL), and RGB 3-bands multi-spectral image with low resolution is regarded as input texture. Set the number of color buffer area to be 3 by Draw Buffers Extension and the buffers are GL_COLOR_ATTACHMENT0_EXT, GL_COLOR_ATTACHMENT1_EXT, GL_COLOR_ATTACHMENT2_EXT, which are used to store I, H, S. In this paper, according to IHS transformation fusion model, two shaders are designed to execute the forward and inverse transformation.

In the first rendering, forward transformation shader is used and destination buffer will obtain values of 3 components I, H, S. It is not suitable for the algorithms which need to store the data when exporting statistics or measurements to be executed in GPU, so histogram matching is handled by CPU. Download the buffer data which store I compenent to system memory and execute histogram matching with high resolution panchromatic image. The data in I are normalized, firstly, we transform the data in I to original grayscale. After matching, the stretched high resolution image is normalized and regarded as new I component to load into texture. At the same time, the buffers that store H, S components are treated as source buffer (put the texture data and new I component data together to be input stream in the second rendering) and GL_COLOR_ATTACHMENT3_EXT 、 GL_COLOR_ATTACHMENT4_EXT 、 GL_COLOR_ATTACHMENT5_EXT are treated as destination buffers. In the second rendering, inverse transformation shader is used, and the data in destination buffer are downloaded into system to finish IHS fusion. The process of calculation is described by figure 3.

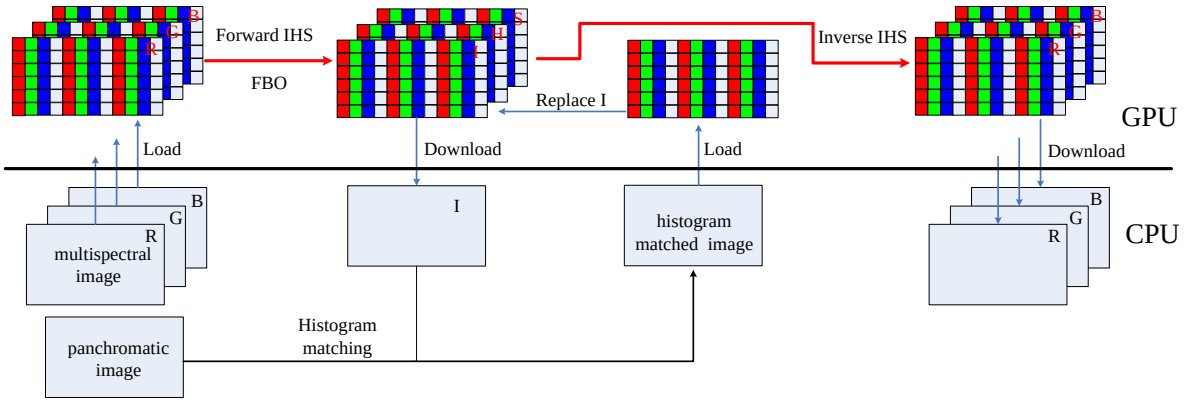

Figure 3. The process of IHS transform fusion

5.2 The DWT fusion on GPU

Wavelet transform has been applied in remote sensing image fusion for a long time, but the traditional way is slow because the arithmetic is very complex. Tien-Tsin Wong[4] proposed a method to realize the DWT on GPU for one image, this paper focused on the two images fusion.

The DWT of the digital images can be looked as a 2D DWT. The 2D DWT can be divided into two steps of 1D DWT first horizontally and then vertically. Take the horizontal 1D DWT as example. Let $\lambda_j(n)$ $\{\lambda_j(n)\}$ be the input signal at level j, $\{\lambda_{j-1}(n)\}$ and $\{\gamma_{j-1}(n)\}$ are the high-frequency (detail) coefficients and low-frequency (coarse) coefficients after filtering and downsampling:

$$\lambda_{j-1}(n) = \sum_k h(k)\lambda_j(2n-k) \tag{5}$$

$$\gamma_{j-1}(n) = \sum_k g(k)\lambda_j(2n+1-k) \tag{6}$$

where the parameter $h(k)$ is the low-pass filter and $g(k)$ is high-pass filter. For efficient SIMF implementation on GPU we rewrite (5) and (6):

$$z_{j-1}(n) = \sum_k f_{d,j-1}(n,k)\, f_{\lambda,j}(n,k) \tag{7}$$

where $f_{d,j-1}(n,k)$ is a position-dependent filter that selects the proper coefficient from $h(k)$ and $g(k)$ at decomposition level j-1, $f_{\lambda,j}(n,k)$ is a function that returns the corresponding data in the level j. These can be implemented by the indirect addressing technique.
Express the 1D DWT to the form of signal input and output:

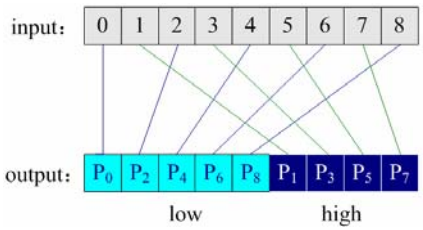

Figure 4. Mapping to the base position in 1D DWT

Assume that the length of input data sequence is P (P=9 in figure 4.), we should first make sure that the signal at $n(n \in [0, P-1])$ after 1D DWT is a low-pass signal or a high-pass signal, we define a filter selector variable a:

$$a = \begin{cases} 1 & (high \quad pass), \quad n > P/2 \\ 0 & (low \quad pass), \quad n \le P/2 \end{cases} \tag{8}$$

With a, we can get the position-dependent filter $f_{d,j-1}(n,k)$. Then we should determine the filtering center of the input signal corresponding to the output position n, we define the filtering center b which can be computed by the following equation.

$$b = 2(n - \alpha\left\lceil \frac{P}{2} \right\rceil) + \alpha + 0.5 \tag{9}$$

0.5 is added to address the pixel center in texture fetching. We can get all the elements in input data for filtering if b is determined.
If the fetching of neighbours goes beyond the image boundary of the current level, we need to extend the boundary extension. Common extension schemes include periodic padding, symmetric padding, and zero padding, etc. Figure shows the horizontal boundary extension (symmetric padding and the width of the filter kennel is 5):

Figure 5. Boundary extension

The output values are computed by performing the inner products of the accessed pixel and the filter kernel coefficients (*h* and *g*). An indirect address table is designed to store the address of the input signals at different level. The width of table for a data sequence is $w=l+K_0-1$, where *l* is the maximum length of the data sequence, K_0 is the width of the filter kernel. Figure 6 shows the indirect address table (l=16，K_0=5).

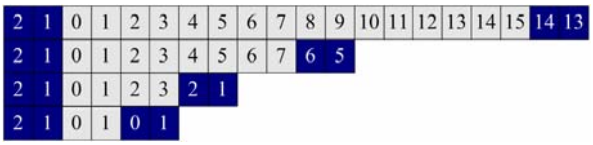

Figure 6. The indirect address table

The texture is organized with each row holding boundary extension, *a* and *b* values for one particular level of DWT. Inside each texel, channel R stores the indirect address of pixel with boundary extended. Channels G and B store *a* and *b* respectively.

The 2D inverse DWT can be achieved by applying 1D inverse DWT horizontally and then vertically. Let $\{\lambda'_{j-1}(n)\}$ and $\{\gamma'_{j-1}(n)\}$ be the low-pass and high-pass signal at level *j*-1. The reconstruction of $\{\lambda_j(n)\}$ is given by

$$\lambda_j(n)=\sum_k h'(k)\lambda'_{j-1}(n-k)+\sum_k g'(k)\gamma'_{j-1}(n-k) \qquad (10)$$

where $h'(k)$ and $g'(k)$ are low-pass and high-pass reconstruction filters respectively. Similar to the forward DWT, (10) can be rewritten as

$$\lambda_j(n)=\sum_k f_{r,j-1}(n,k)\,f_{z,j-1}(n,k) \qquad (11)$$

where $f_{z,j-1}(n,k)$ returns the corresponding data in the up-sampled boundary-extended signal at level *j*-1. Express the 1D inverse DWT to the form of signal input and output:

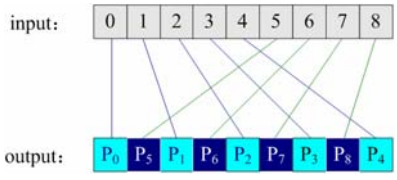

Figure 7. Mapping to the base position in 1D invert DWT

1) Halve the input signal as in figure 8.

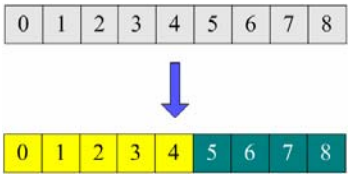

Figure 8. Grouping

2) Upsampling and boundary extension as in figure 9.

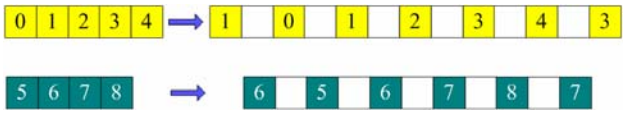

Figure 9. Upsampling and boundary extension

3) Interleaving two groups of signal as in figure 10.

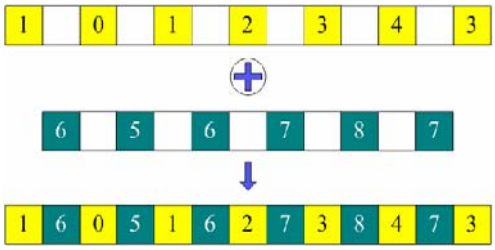

Figure 10. Interleaving

Once the indirect address table is ready, values in the next level can be reconstructed by convolution. Note that low-frequency elements must be multiplied to the low-pass reconstruction filter, while high-frequency elements must be multiplied to high-pass reconstruction.

Now we get the method to realize the DWT on one image, but remote sensing image fusion has to deal with two images (the case that the number of images is more than two is based on the case of two images, so this paper analyses two images fusion), the storage and management of the intermediate data is the key point.

This paper set a FBO for each of the image to be fused. Two texture objects are bind to the FBO of the first image: m_fbort1[0] and m_fbort1[1], the binding points are set to GL_COLOR_ATTACHMENT0_EXT and GL_COLOR_ATTACHMENT1_EXT. When DWT applied to the first image at level 1, the destination buffer is first set to GL_COLOR_ATTACHMENT1_EXT, and load the image data to texture m_fbort1[1] which is bind to GL_COLOR_ATTACHMENT1_EXT; then set the destination buffer to GL_COLOR_ATTACHMENT0_EXT, perform the horizontal 1D DWT fragment shader and store the result to texture m_fbort1[0]. Then set the destination buffer to GL_COLOR_ATTACHMENT1_EXT, perform the vertical 1D DWT fragment shader and store the result to texture m_fbort1[1]. At level 2, the source data is the data in texture m_fbort1[1], the destination buffer of horizontal 1D DWT is set to GL_COLOR_ATTACHMENT0_EXT and then the vertical is set to GL_COLOR_ATTACHMENT1_EXT. So the result of the 2D DWT is stored in m_fbort1[1] too. Because the 2D DWT is achieved by exchanging the buffers twice at each level,

the result will store in the texture m_fbort1[1] no matter at which level.

Be the same, the FBO of the second image binds two textures too: m_fbort2[0] and m_fbort2[1]. The result of the 2D DWT will be stored in the texture m_fbort2[1].

The next step is to fuse the data in texture m_fbort1[1] and m_fbort2[1]. The texture m_fbort1[1] and m_fbort2[1] are bind and passed to weighted fusion fragment program by two uniform variables the type of which is samplerRECT, and the destination buffer is set GL_COLOR_ATTACHMENT0_EXT. Then pass the data in this buffer to GL_COLOR_ATTACHMENT1_EXT. The invert DWT is a reconstruction process from the highest level. The result data of the reconstruction at level *j* is the source data of level *j*-1. As the case of forward 2D DWT, the result of 2D inverse DWT is stored in buffer GL_COLOR_ATTACHMENT1_EXT. At last download the data in texture m_fbort1[1] bind to GL_COLOR_ATTACHMENT1_EXT to system memory to get the fusion result.

6. EXPERIMENT RESULT

Experiment condition: DELL PC, 3GHz Pentium(R) 4, NVIDIA GeForce 8800 GTS with 320M video memory;
Experiment data: Landsat TM images. We choose 4 groups of registered images and the size (pixel) of each group is: 256×256，512×512，1024×1024，2048×2048.
Table 1 shows the time cost of the fusion arithmetic both on CPU and GPU.

Image size		256×256	512×512	1024×1024	2048×2048
product fusion	GPU	8.0	16.8	51.0	184.4
	CPU	16.5	65.4	263.9	1098.3
ratio fusion	GPU	8.1	17.0	50.9	184.1
	CPU	8.1	33.1	136.9	533.5
high-pass filtering	GPU	10.3	17.1	45.0	150.0
	CPU	3.6	14.5	59.4	297.9
weighted fusion	GPU	7.6	16.4	50.4	181.8
	CPU	6.6	26.8	106.9	454.4
IHS fusion	GPU	60.3	89.4	195.5	636.8
	CPU	42.8	172.1	650.2	2621.4
DWT fusion	GPU	647.5	770.7	1043.1	2135.5
	CPU	419.3	1627.8	5808.1	23910.5

Table 1. Time cost comparision of GPU and CPU

From the result we can see that the image fusion based on GPU do not has the advantage as based on CPU while the data size is small, but with the image size getting bigger, the image fusion speed based on GPU is much quicker than that based on CPU, and the computing time on GPU does not increase proportionally with the data size, and the advantage increases with the complexity of the fusion arithmetic.

REFERENCES

[1] Randi J.Rost,2006. OpenGL Shading Language. Posts & Telecom Press, Beijing, pp.83-96

[2] Jia Yonghong,2005. Multi-sensors Remote Sensing Image Data Fusion. Surveying and Mapping Press, Beijing, pp.31-38.

[3] TANG Guo-liang, PU Jie-xln , HUANG Xin-han,2006. Color Image Fusion Algorithm Based on IHS and Wavelet Transformation, APPLICATION RESEARCH OF COMPUTERS, 23(10), PP. 174-176.

[4] Tien-Tsin Wong., Chi-Sing Leung, Pheng-Ann Heng, Jianqing Wang,2004. Discrete Wavelet Transform on Consumer-Level Graphics Hardware. ACM Workshop on General-Purpose Computing on Graphics Processors (GP2).