University of Huddersfield Repository

Su, Yang, Xu, Zhijie and Jiang, Xiang

Stream-Based Data Filtering for Accelerating Metrological Data Characterization

Original Citation

Su, Yang, Xu, Zhijie and Jiang, Xiang (2008) Stream-Based Data Filtering for Accelerating Metrological Data Characterization. In: Proceedings of the 14th International Conference on Automation and Computing. ICAC, pp. 81-85. ISBN 978-0-9555294-2-0

This version is available at http://eprints.hud.ac.uk/id/eprint/4744/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/

Stream-Based Data Filtering for Accelerating Metrological Data Characterization

Yang Su[a,b] Zhijie Xu[a] and Xiangqian Jiang[a]
[a]School of Computing & Engineering, University of Huddersfield
Queensgate, Huddersfield HD1 3DH, UK
[b]School of Communication & Information Engineering, Xi'an University of Sci. & Tech., Xi An 710054, China
E-mail:y.su@hud.ac.uk

***Abstract*: The main task of engineering surface metrology is to characterize a surface by assessing components such as form, waviness and roughness that correspond to different wavelength segments in the frequency domain, which are often extracted by deploying filtering techniques. The effectiveness of a specific kind of filtering algorithms is jointly determined by their filtering accuracy and computational efficiency. In this paper, a data stream-based programming paradigm is introduced which takes advantage of the programmability and parallel computation capacity of modern graphics process unit (GPU) to execute and accelerate the Gaussian filtering process that is extensively used in surface metrological data processing. In contrast to the results obtained by running MATLAB simulation kit for similar processes, the software framework speeds up the filtering process substantially while yielding satisfying accuracy as that of the corresponding MATLAB program, which proved the practicability and validity of the proposed computation model.**

Key words: Surface metrology, Graphics process unit, Gaussian filtering, Parallel computation

1 Introduction

Form, waviness and roughness are three main factors used to assess quality of a surface. These elements can be characterized as various frequency components, for instance, the roughness corresponds to high frequency (or short wavelength) components, the waviness corresponds to medium frequencies, and the form to low frequencies [1]. As a result, filtering techniques, -- for instance, Gaussian filter (GF), Gaussian regression filter (GRF) and spline filter -- are widely used in surface metrology to extract various frequency components to assess an engineering surface. One of the key problems of metrology data processing by applying filtering techniques is its computational accuracy and efficiency which often poses as a dilemma that has to be addressed [2]. In some cases, efficiency has to be sacrificed for more accurate processing results, especially in the case when raw metrology data coming in of a huge size.

In recent years, with the development on programmability of graphics process units (GPUs), General-purpose computing on graphics process unit (GPGPU) has been becoming a trend that issues data intensive computing on GPU rather than to CPU [3]. Various GPU languages such as high level shading language (HLSL) from Microsoft Co., C for graphics (Cg) from Nvidia Ltd, and OpenGL Shading Language (GLSL) from Silicom Graphics that were originally developed for on-screen rendering are starting to support the applications of GPGPU. As examples, linear algebraic and partial differential computations can now be carried out on most of the off-the-shelf GPUs on today's market. The chief characteristic of GPGPU is its adoption of the data parallel computational paradigm that is achieved by the stream processing, which supplied the foundation for GPGPU's superior computational efficiency over CPU-based approach.

In this paper, a software framework and its corresponding prototype for implementing GPU-accelerated Gaussian filtering for 3D metrological data processing is presented. This prototype employs the technique of Off-Screen Rendering (OSR) to the primitive metrological data in the model space. With OSR activated, Framebuffer Object (FBO) introduced by OpenGL has been employed as the solution for the process of render-to-texture, which is more flexible and efficient than the conventional approach of GPGPU filtering by using Pixel Buffer (PBuffer) only for data storage. The results of this approach are compared with those obtained by MATLAB simulations to validate the accuracy and effectiveness of the new technique.

The remainder of this paper is organized as follow: Section 2 gives a brief introduction to Gaussian filtering that is commonly used in surface metrological data processing, Section 3 presents the framework and prototype of the new GPU accelerated 2D Gaussian filtering technique that is based on OSR, Section 4 evaluates the performance of the proposed framework with Section 5 concludes and highlights the future works.

2 Review of Gaussian filtering in Metrology

The Gaussian filtered mean line is established as the reference line in 2D surface metrology by the ISO 11562-1996 standard and ASME B46-1995 standard [4,5].The impulse response of recommended Gaussian filter in spatial domain is given as

$$h(t) = \frac{1}{\alpha\lambda_c} e^{-\pi(t/\alpha\lambda_c)^2} \tag{1}$$

where t is the independent variable in the spatial domain, λ_c is the cut-off wavelength (in the units of t) , α is a constant and $\alpha = \sqrt{\ln 2/\pi} = 0.4697$.

For 3D surface assessment, a reference surface, also known as the mean surface, must be established. The reference surface can be obtained by employing a 2D Gaussian filter which corresponds to the following impulse response

$$h(x,y) = \frac{1}{\beta\lambda_{xc}\lambda_{yc}} \exp\{-\frac{\pi}{\beta}[(\frac{x}{\lambda_{xc}})^2 + (\frac{y}{\lambda_{yc}})^2]\} \tag{2}$$

where $\lambda_{xc}, \lambda_{yc}$ are the cut-off wavelengths along x- and y-directions respectively and $\beta = \alpha^2 = \ln 2/\pi$. From Eq.(2), we can see

$$\begin{aligned} h(x,y) &= \frac{1}{\beta\lambda_{xc}\lambda_{yc}} \exp\{-\frac{\pi}{\beta}[(\frac{x}{\lambda_{xc}})^2 + (\frac{y}{\lambda_{yc}})^2]\} \\ &= \frac{1}{\sqrt{\beta}\lambda_{xc}} \exp[-\pi(\frac{x}{\sqrt{\beta}\lambda_{xc}})^2] \cdot \frac{1}{\sqrt{\beta}\lambda_{yc}} \exp[-\pi(\frac{y}{\sqrt{\beta}\lambda_{yc}})^2] \\ &= h(x) \cdot h(y) \end{aligned} \tag{3}$$

According to this separable property, a 2D Gaussian filter can be implemented by using two independent 1D Gaussian filters, one in the x-direction, and the other in the y-direction. If $u(x,y)$ is used to stand for the primitive unfiltered profile and $w(x, y)$ for the filtered reference surface, then the filtered profile - $w(x, y)$ is equivalent to the convolution between $u(x,y)$, that is

$$\begin{aligned} w(x,y) &= u(x,y) * h(x,y) \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} u(\xi,\eta) h(x-\xi, y-\eta) d\xi d\eta \end{aligned} \tag{4}$$

For an actual digital signal processing system, the continuous input and output signal are discrete and finite. The discrete and finite form of Eq. (1) which is expressed as the value of k-th sample point

$$h_k = \frac{1}{\alpha\lambda_c} e^{-\pi(k\Delta\xi/\alpha\lambda_c)^2} \tag{5}$$

h_k actually stands for the value of k-th sample point when sampling the continuous impulse response according to a interval $\Delta\xi$. For an unfiltered surface profile $u(x,y)$, its discrete output of filtered surface by 2D Gaussian filtering is

$$\begin{aligned} w(x_i, y_s) &= \sum_{k=-m}^{m}\sum_{j=-n}^{n} u(x_{i-k}, y_{s-j}) \cdot h(x_k, y_j) \cdot \Delta\xi_x \cdot \Delta\xi_y \\ &= \sum_{k=-m}^{m} h(x_k)\Delta\xi_x \sum_{j=-n}^{n} u(x_{i-k}, y_{s-j}) \cdot h(y_j) \cdot \Delta\xi_y \\ &(i \in [-m,-m+1\ldots,m-1,m] \quad s \in [-n,-n+1\ldots,n-1,n]) \end{aligned} \tag{6}$$

For applications in surface metrology, the waviness and roughness are strictly limited within the micron scale, sometimes even to the nano-metre dimension [6]. To extract these miniature components from a measured surface, it requires that the kernel radius of a Gaussian filter, i.e., parameter m and n, must be large enough to guarantee the filtering accuracy. This essential step brings in a large amount of computation and results in the problem of computational efficiency. For example, when a primitive surface that was sampled at the size of 1024×1024 being filtered by a 2D Gaussian filter with kernel radius of 200, it took MATLAB program nearly 20 seconds to complete the calculation and rendering on a 2.6GHz PC with 2GB memory. For tackling this problem, Yuan et al. presented a new fast algorithm for constructing a series of Gaussian filters according to the approximation of Gaussian function through applying central limit theorem to reduce the multiplication operations in the procedure with certain degree of success.

In the last decade, graphics processing unit (GPU) has matured substantially and being viewed as a powerful embedded parallel processor with the ability of stream-based processing. Hardware acceleration is more accessible for speeding up scientific computation. Section 3 presents a GPGPU-based framework for 2D Gaussian filtering for improving processing efficiency of Gaussian filters for surface metrology.

3 GPU-accelerated 2D Gaussian filtering

3.1 Improving Data Efficiency using OSR

The display traversal in GPU's graphic pipeline can be divided into three tiers; namely geometry processing, rasterization, and fragment operation. These three tiers convert a set of planar polygon into a raster image in a virtual scene [7]. Geometry processing and fragment operations are issued in vertex pipeline and fragment pipeline respectively. Both vertex pipeline and fragment pipeline in early GPU issue a fixed sequence of processing stages. Modern GPUs are programmable and used as parallel processors; both the vertex processor and the fragment processor can be used for issuing algorithm/model devised by developers. For most of GPGPU applications, feedback loop is essential but pains-taking slow in which the intermediate results of calculation need to be stored on GPU but not visible. To solve this problem, off-screen rendering -- similar to rendering into on-screen window -- is highly recommended in GPGPU [8, 9].

In actual implementation practices, Pixel buffer (PBuffer) was first introduced by OpenGL for off-screen rendering which is defined with an enumerated list of available pixel formats. PBuffer can be bound to a texture directly for render-to-texture, which is achieved by instruction WGL_ARB_render_texture in OpenGL [10]. The biggest complaint of PBuffer is its requirement for unique GL contexts that include device context of graphics device interface (GDI) and rendering context. It means that the application must keep track of multiple sets of states, which is cumbersome and inconvenient. Additionally, context switching between different PBuffers is an expensive operation. To solve this problem Framebuffer Object (FBO) entered as a better render-to-texture solution. Except integrating directly with regular textures like the PBuffer, FBO requires no extra GL contexts and allows depth, stencil and color buffers to be shared among framebuffers which are impossible for PBuffer-based approach [11].

In this application, primitive surface metrological data originates from the model space with the proposed GPU-accelerated Gaussian filtering applies FBO as the solution for executing instructions.

3.2 Proposed software framework

The flow and framework of the proposed GPU-accelerated 2D Gaussian filtering system is depicted in Fig.1

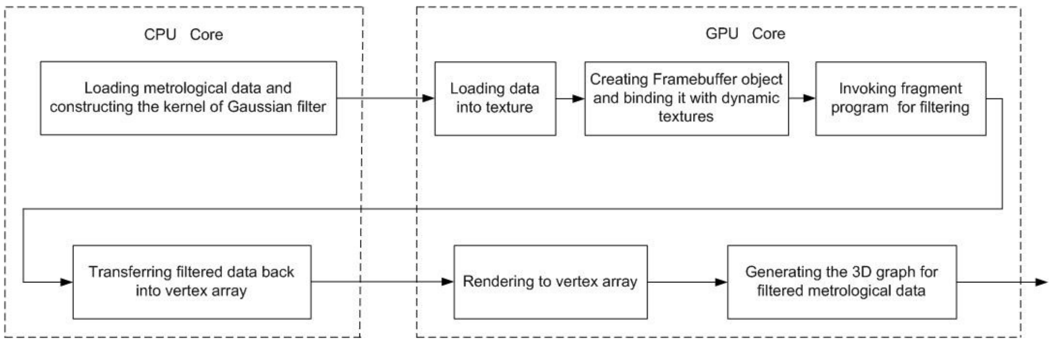
CPU Core
Loading metrological data and constructing the kernel of Gaussian filter
GPU Core
Loading data into texture
Creating Framebuffer object and binding it with dynamic textures
Invoking fragment program for filtering
Transferring filtered data back into vertex array
Rendering to vertex array
Generating the 3D graph for filtered metrological data

Figure 1. Framework of GPU-accelerated 2D Gaussian filtering

In this structure, fragment shader is employed for calculation since the fragment pipeline normally provides more computational horsepower than vertex pipeline as there are more fragment streams. In addition, texture lookups are more efficient in fragment programs. The pseudo-code of a fragment program implemented in this research is shown in Fig.2.

```
float Filtering (uniform samplerRECT data :   TEXUNIT0,     // the metrology data
                 uniform samplerRECT kernel: TEXUNIT1,      // Gaussian filter's kernel
                 uniform int kernel_width,                  // kernel width in x- direction
                 uniform int kernel_height,                 // kernel height in y-direction
                 uniform float2 kernel_offset,              // kernel offset
                 float2 pos : TEXCOORD0                      // texel position in TEXUNIT0
                 ) : COLOR
{
        float c = 0;
        for(int y=0; y<kernel_height; y++) {
                for(int x=0; x<kernel_width; x++) {
                        float weight=texRECT(kernel, float2(x, y)).r;
                        c +=  texRECT( data, pos + float2(x, y)  + kernel_offset).r * weight;
                }
        }
        return c;
}
```

Figure 2. Pseudo-code of fragment program

For visualizing the filtered data, the computational results need to be transferred from GPU's memory back to CPU's memory at current stage in the form of vertex array. Although GPU has powerful parallel processing ability, this process creates a bottleneck for applications [12]. For example, it has been tested in our research that to transfer a 1124×1124 single precision floating-point block from Nvidia's 7900 GPU back to CPU took nearly 7 seconds.

To partially resolve this problem, the devised framework in this project employs the solution of data splitting that splits a set of data from its original size into several smaller blocks for speeding up data transfer from and to CPU. Potentially supported by Nvidia's Scalable Link interface (SLi) technology, data splitting works as follows: firstly, the primitive metrological data is split into n parts where n is constrained to square of an integer to guarantee the normal texture lookup in fragment program; then these n parts of data is convolved with the kernel of 2D Gaussian filter respectively and the filtered data is stored in n dynamic textures which are all bounded with a same Framebuffer object. At last, the data in n dynamic textures will be read back to vertex array in CPU. For the correctness and integrity of filtered data, the detailed process of splitting on the primitive data with the original size of W×H and storing mechanism for the filtered data in the form of n dynamic textures is shown in Fig.3.

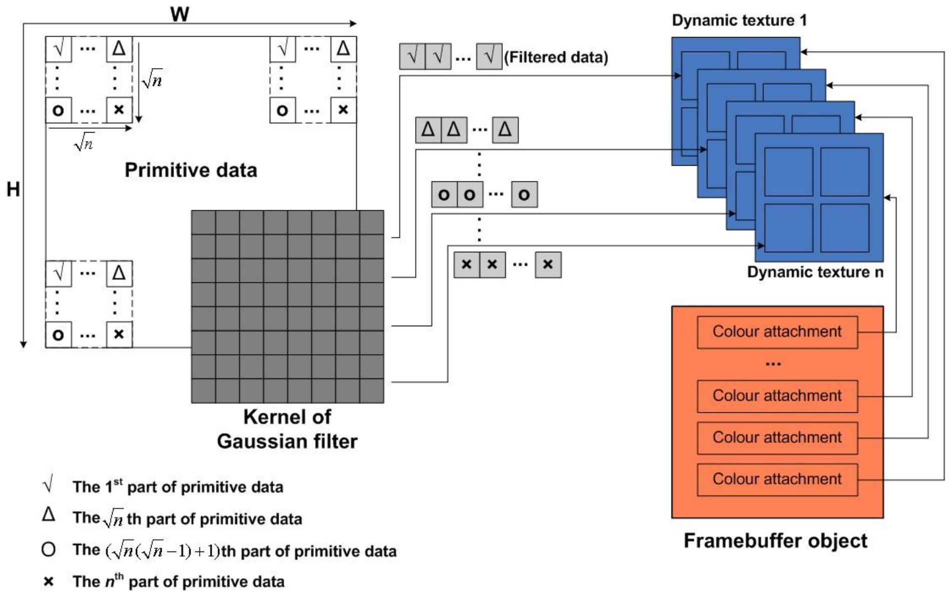
W
H
$\sqrt{n}$
$\sqrt{n}$
Primitive data
Kernel of Gaussian filter
(Filtered data)
Dynamic texture 1
Dynamic texture n
Colour attachment
Colour attachment
Colour attachment
Colour attachment
Framebuffer object
√ The 1st part of primitive data
Δ The $\sqrt{n}$ th part of primitive data
O The $(\sqrt{n}(\sqrt{n}-1)+1)$th part of primitive data
× The nth part of primitive data

Figure 3. Data splitting and storage in fragment program

4 Performance evaluation

The primitive surface used in our test is shown in Fig.4, which is sampled at the size of 1024×1024. The amplitude of surface terrain varies within micron level. The cut-off wavelength of 2D Gaussian filter, along x- and y-directions, are both equivalent to 0.8. The sampling interval within the cut-off wavelength, as donated in Equation (6), were both set to 0.016, thus the kernel radius of 2D Gaussian filter along x- and y-directions are both 50.

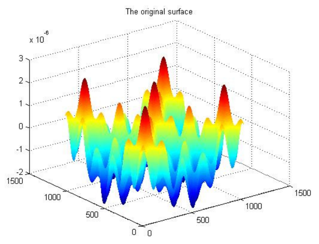

Figure 4. A primitive surface profile

Fig.5 and Fig.6 show the results obtained by MATLAB simulation kit and the developed GPGPU programming respectively, from which it can be seen that GPU-accelerated Gaussian filtering obtains the same accurate results as those obtained from MATLAB simulations.

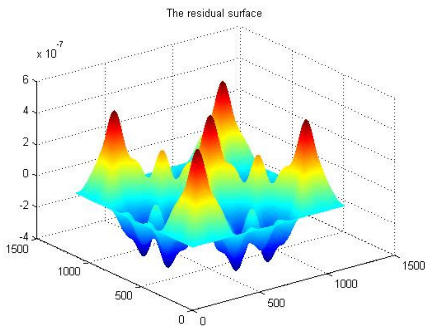

Figure 5. Result of Gaussian filtering issued by MATLAB simulations

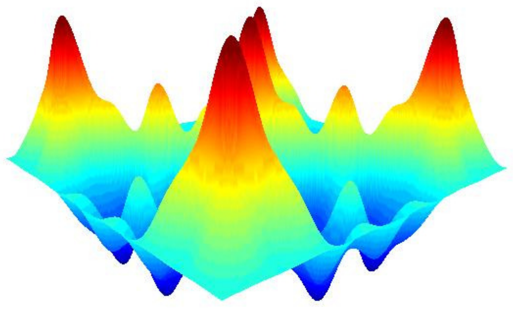

Figure 6. Result of GPGPU-based Gaussian filtering

To verify computational efficiency of the developed GPGPU programming framework, the run time of each stage of the model and the run time of MATLAB simulation programs are listed in Table 1. If only consider the time utilized in issuing filtering algorithm (convolving with Gaussian filter), the GPGPU programming framework has a speed up factor that is of ×10 scale. Although transferring the computational results form GPU back to CPU is still a bottleneck, the GPGPU programming framework has gained a better performance than CPU-driven MATLAB calculation.

Table 1. Processing time of GPGPU program and MATLAB simulation

	GPGPU	MATLAB
Data from CPU to GPU	0.62s	Not required
Convolving with Gaussian filter	0.41 s	4.94s
Data from GPU to CPU	1s	Not required

5 Conclusions and future work

So far in this project, a software framework for GPU-accelerated Gaussian filtering has been designed and developed. Testing results have revealed its superiority over a pure CPU based implementation. However, there are still problems to be solved before wider adoption. One of the shortcomings of Gaussian filter used in surface metrological data processing is that it brings in edge effects. To overcome this problem, other kinds of filtering techniques, such as Gaussian regressive filter, R_k filter, spline filter, and wavelet-based filter on GPGPU need to be investigated. These filters are believed to possess higher processing accuracy than conventional Gaussian filter but unavoidably costing more computational time. Proposing a benchmark and guidelines for designing "hardware"-oriented solutions for these filters is one of the primary tasks for the next stage of the project.

GPGPU is a relatively new research area with limited number of published solutions. It is believed that approaches in increasing case studies will encourage GPU manufactures and shader language developers to work closer and "shaping" the next generation of GPUs. Some new GPU hardware structures, platforms and tools -- such as DX10-Confirming graphics cards and CUDA language have already promised new potentials for future applications.

Acknowledgment

We would like to thank Dr Shaojun Xiao and Dr Feng Xie in the Centre of Precision Technology in Huddersfield University for obtaining us the MATLAB simulation program to evaluate our model's performance by comparing with the results of running the MATLAB program.

[1] J. Raja, B. Muralikrishnan, Shengyu Fu. "Recent advances in separation of roughness, waviness and form", Journal of the International Societies for Precision Engineering and Nanotechnology, Vol. 26, 2002, pp.222-235.
[2] M. Numada, T. Nonura, K. Kamiya, et al. "Filter with variable transmission characteristics for determination of three-dimensional roughness", Precision Engineering, Vol. 30, 2006, pp.431-442.
[3] J. D. Owens, D. Luebke, N. Govindaraju, et al. "A Survey of General-Purpose Computation on Graphics Hardware", Computer Graphics Forum, Vol. 26, Issue 1, 2007, pp.80 -113.
[4] ISO 11562:Geometrical product specification(GPS)—surface texture: profile method—metrological characteristics of phase correct filters. Geneva: International Organization for Standardization,1996.
[5] ASME B46.1:Surface Texture: Surface Roughness, Waviness, and Lay. New York: American Society of Mechanical Engineers,1995.
[6] L. Blunt, X. Q. Jiang. "Advanced techniques for assessment surface topography: Development of a basis for 3D surface texture standards 'SURFSTAND'", Kogan Page Science, London, 2003.
[7] K. Engel, M. Hadwiger, J. M. Kniss, et al. "Real-Time Volume Graphics", Course Note 28 in SIGGRAPH2004, 2004, pp. 14~18.
[8] S. Guthe, W. Strasser. "Advanced techniques for high-quality multi-resolution volume rendering", Computers & Graphics, Vol. 28, Issue 1, 2004, pp.51-58.
[9] Juekuan Yang, Yujuan Wang, Yunfei Chen. "GPU accelerated molecular dynamics simulation of thermal conductivities", Journal of Computational Physics, Vol. 221, Issue 2, 2007, pp. 799-804.
[10] Christopher Oat. "Rendering to an off-screen buffer with WGL_ARB_pbuffer", Technology paper of ATI

Inc. pp.1-13. Available on: http://ati.amd.com/developer/ATIpbuffer.pdf.

[11] Emil Persson. "Framebuffer Objects", Technology paper of ATI Inc. pp.1-12. Available on: http://ati.amd.com/developer/SDK/AMD_SDK_Samples_May2007/Documentations/FramebufferObjects.pdf.

[12] I. Geys, L. V. Gool. "View synthesis by the parallel use of GPU and CPU", Image and Vision Computing", Vol. 25, Issue 7, 2007, pp.1154-1164.