Parallel Curvature Filter for High Performance Image Processing

Wei Pan, Yuanhao Gong, Guoping Qiu*

*College of Information Engineering&Guangdong Key Laboratory of Intelligent Information Processing, Shenzhen University, Nanhai Road, Shenzhen, China 518060

ABSTRACT

Recently, curvature filter (CF) has been developed to implicitly minimize curvature for image processing problems such as smoothing and denoising. In this paper, we propose a parallel curvature filter (PCF) that performs on GPU which is much faster than the original CF on CPU. Inspired by Convolution Neural Networks processed by GPU, the convolution operations in curvature filter computation can be similarly paralleled by GPU so that the PCF on a single GPU can process 33.2 Giga pixels per second. Such performance allows it to work in the real-time applications such as video processing and biomedical image processing, where high performance is required. Our experiments confirm the efficiency and effectiveness of the PCF.

Keywords: Filter, mean curvature, parallel computing, performance enhancing, video processing, real-time

1. INTRODUCTION

Image processing plays a fundamental role in various research fields, such as biomedical process modeling, material science, observing physical experiments, astrophysics, etc. Generally, filters with low computational cost are preferred. In some special tasks or scenarios, high performance of image processing is usually required to achieve real time processing, and as the resolution increases, to reduce the computational time is becoming difficult. Thanks to the modern hardware, such as GPU, many image processing tasks can be parallel and reach high performance.

Recently, curvature filter is developed to minimize curvature regularization in image processing models.[1,2] It is a discrete filtering approach enables efficient computing of reduced-energy images by successively minimizing an energy function:

$$E(U) = E_0(U, I) + \lambda E_1(U)\,. \tag{1}$$

where the data-fitting energy $E_0(U, I) \geq 0$ measures the fitness of resultant image U to the original image I, and the regularization energy $E_1(U) \geq 0$ formalizes prior knowledge about U. Filters for Gaussian curvature (GC),[3,4] mean curvature (MC),[5,6] and total-variation (TV)[7] regularizers are implemented with a local approximation operators. By iteratively applying these operators to the original image, the regularization energy will be reduced to get a piecewise developable (GC), minimal (MC), or piecewise-constant (TV) surface. Among them, Gaussian curvature is an intrinsic property of the signal regardless of its representation and embedding.

However, many of the images that need to be enhanced are in high resolutions, or need to be processed in real or near real-time. Although curvature filter is very fast, it takes minutes to process a big size image, e.g. with size 2048×2048. This fact motivates us to perform it on GPU.

In this work, we present a image processing technique called parallel curvature filter (PCF) implemented on GPU which can be applied to scenarios such as high definition and real-time image smoothing. The performance of PCF is further evaluated with several different parameter settings.

Further author information: (Send correspondence to Wei Pan or Yuanhao Gong)
Wei Pan: E-mail: vpan@foxmail.com, Telephone: 86 13538128621
Yuanhao Gong: E-mail: gongyuanhao@gmail.com

2. PRINCIPLE

2.1 Geometric Formular

Assuming $\vec{x} = (x, y) \in \Omega$ denotes the spatial coordinate, where Ω is the input 2D image domain. Let $I(i, j) : \vec{x} \mapsto \mathbb{R}^+$ denote the given discrete digital image with coordinates i and j. Let $U(\vec{x})$ denote the desired output image to be estimated. We interpret the signal as a geometric surface over the space of the data, i.e., $\Psi(\vec{x}) = (\vec{x}, U(\vec{x}))$. According to this definition, curvature can be computed by taking partial derivatives over x and y. For Gaussian curvature, we have:

$$K(U(\vec{x})) = \frac{U_{xx}U_{yy} - U_{xy}^2}{(1 + U_x^2 + U_y^2)^2}\,, \tag{2}$$

where U_x and U_{xx} denote the first and second partial derivative with respect to x. We can minimize the total absolute GC by this regularizer:[4]

$$E_1^{\mathrm{GC}}(U) = \int_\Omega |K(U)|\,\mathrm{d}\vec{x}\,, \tag{3}$$

Similarly, the regularization energy for mean curvature (MC) is

$$E_1^{\mathrm{MC}}(U) = \int_\Omega |H(U)|\,\mathrm{d}\vec{x}\,. \tag{4}$$

with the MC H computed from U as

$$H(U) = \frac{(1 + U_y^2)U_{xx} - 2U_xU_yU_{xy} + (1 + U_x^2)U_{yy}}{2(1 + U_x^2 + U_y^2)^{3/2}}\,. \tag{5}$$

The regularizers mentioned here all minimize the surface energy in a local 3×3 pixel neighborhood around each pixel, as shown in the next subsection.

2.2 Iterative Curvature Filter

Traditionally, solving curvature regularization terms listed aboves is difficult due to the complexity of the computation of surface curvature with conventional gradient decent method. The curvature filter proposed recently solved this problem[1] in an iterative way, as shown in Figure 1.

A surface can be locally apporximated by its tangent plane. Therefore, we can project $U^t(\vec{x})$ to $U^{t+1}(\vec{x})$ such that $U^t(\vec{x})$ is on the closest tangent plane of any neighboring pixel. In a 3×3 pixel neighborhood, 8 different projective distances can be enumerated, i.e., $\{d_i,\, i = 1, \ldots, 8\}$[1](Figure 1).

Each d_i is the projection distance to the corresponding geometric priors. The minimal one indicates that the current pixel has the highest probability to the desired geometric configuration as well as the closest fitting to the original image. Therefore it is selected to update each pixel in each iteration, as shown in equation 6 and 7.

$$U^{t+1}(\vec{x}) = U^t(\vec{x}) + d_m \tag{6}$$

$$|d_m| = \min\{|d_i|,\, i = 1, \ldots, 8\} \tag{7}$$

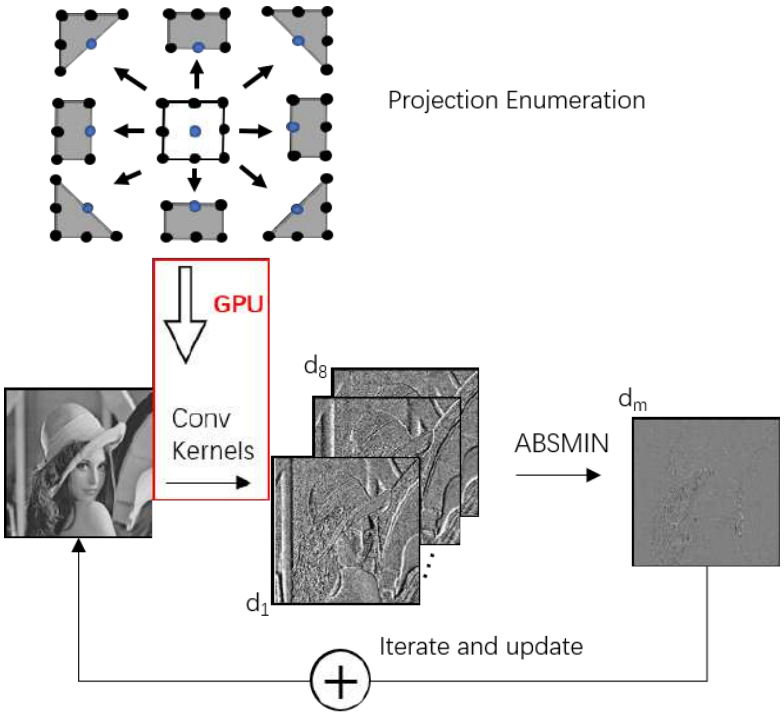

Figure 1. Illustration of the parallel curvature filter flow. Firstly, 8 projective distances are enumerated in a 3×3 neighborhood. Secondly, the 8 shifting distances of each pixel in an image surface is computed by convolution. Finally, the minimal one is selected to update each voxel in each iteration.

2.3 Parallel Curvature Filter

The curvature filter runs several convolution kernels on the same image to choose the possible geometric configuration for each pixel, as shown in equation 8.

$$f(x,y) \otimes h(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)h(x-m, y-n) \tag{8}$$

for $x = 0, 1, 2 \ldots M-1$ and $y = 0, 1, 2 \ldots N-1$. These convolution operations can be paralleled on GPU. The most computationally-expensive part in our algorithm is the 8 convolutions. It can be accelerated by the share memory on GPU. This yields our PCF as following.

A short introduction to the GPU architecture is presented. Interested readers may refer to books about GPU programming and the CUDA programming guide for further details.[9,10] A compiler generates executable code for CUDA device. The CPU regards a CUDA device as a multi-core co-processor. CUDA threads access data from multiple memory spaces during their execution. There are three levels of memory structure shown in Figure 2. The first level consists of private local memory and register which are owned by each thread. The second level is the shared memory which can be accessed by all threads of the block. The third level is the global memory, the constant memory and the texture memory which all threads can get access to.

3. RESULTS AND DISCUSSION

We implemented the CUDA code for our parallel curvature filter with the GPU block size ranging from 16 to 32. For specificity, the hardware used is given below:

CPU: Intel(R) Xeon(R) CPU E5-1620v4@3.50GHz

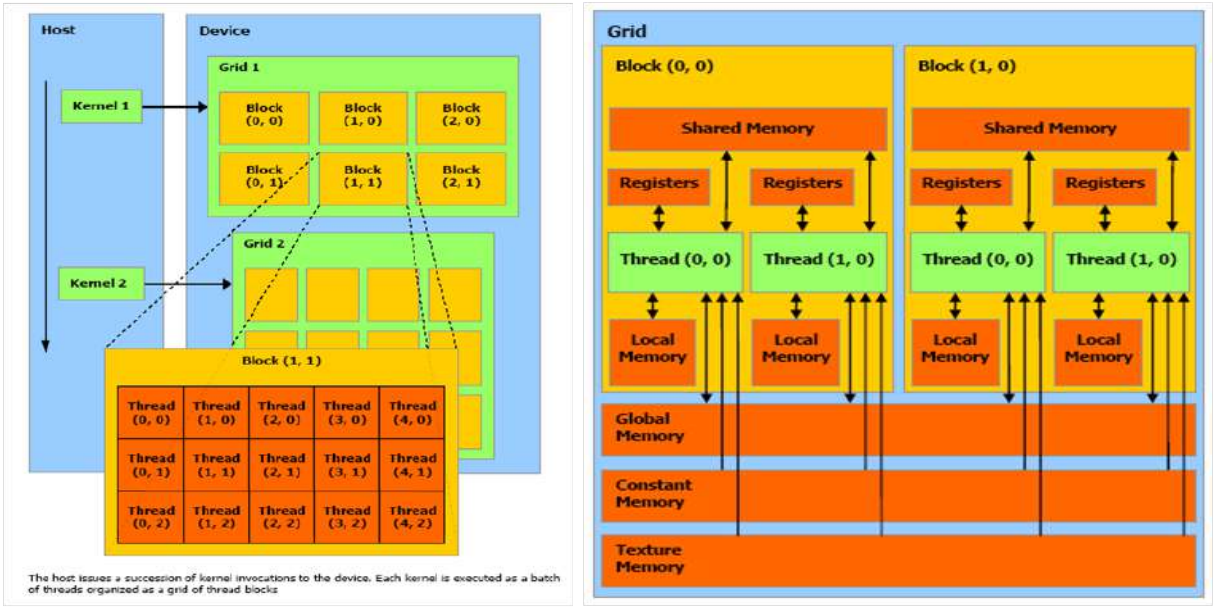

Figure 2. CUDA Memory Structure[11]

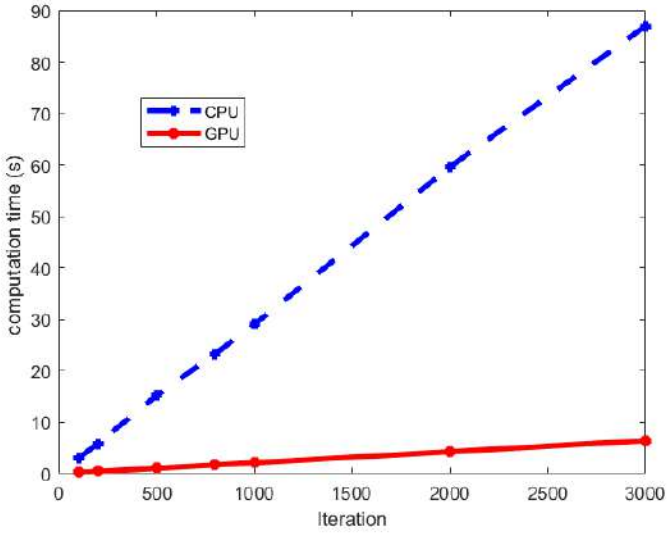

Figure 3. Comparison of computation time between CPU and GPU.

GPU: Nvidia GeForce GTX 1080 Ti, with 3584 CUDA cores and 11264 MBytes global memory size, 49152 bytes shared memory per block.

Compared with the original curvature filter proceeded solely on CPU, PCF is much faster, which is compatible for real-time applications. To evaluate the PCF, we did a simple comparison between them. Figure 3 shows the result of the same image processed by both CPU and GPU.

The results of GC PCF and MC PCF are shown in Figure 4 and Figure 5 respectively. In order to see the surface is regularized by iteration, we plotted the image surface intensity distribution along a line. Obviously the MC PCF is more efficient than the GC PCF.

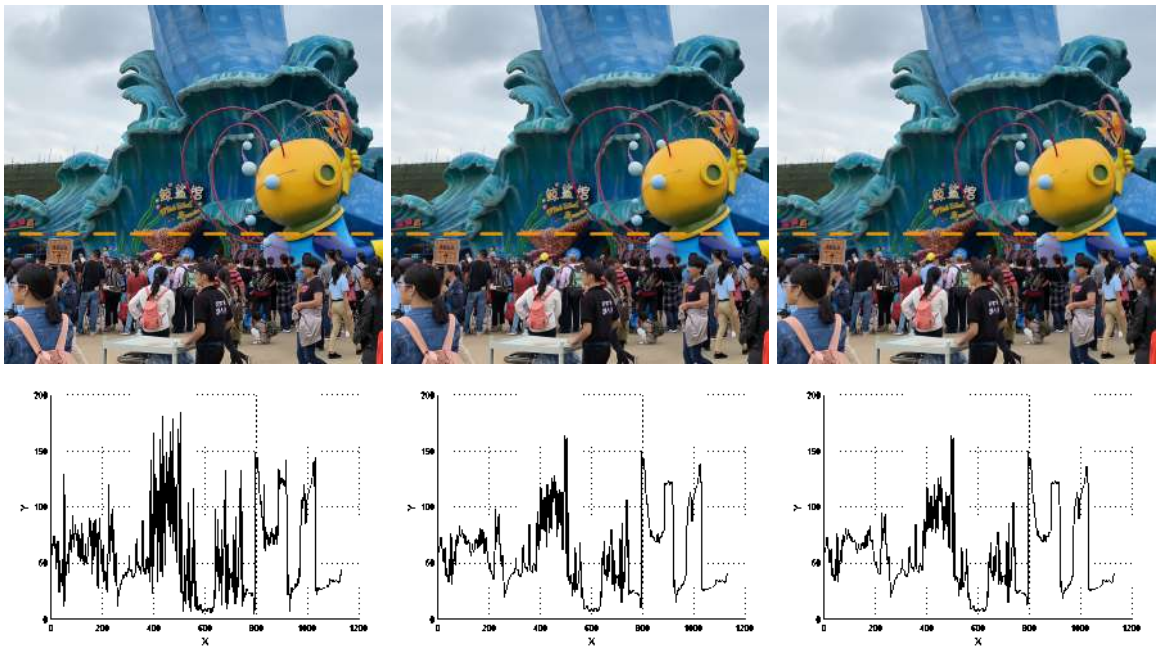

Figure 4. Gaussian Curvature PCF on a photo. From left to right are: origin photo, after 10, 30 iterations. The orange dash line profiles the intensity distribution of the image.

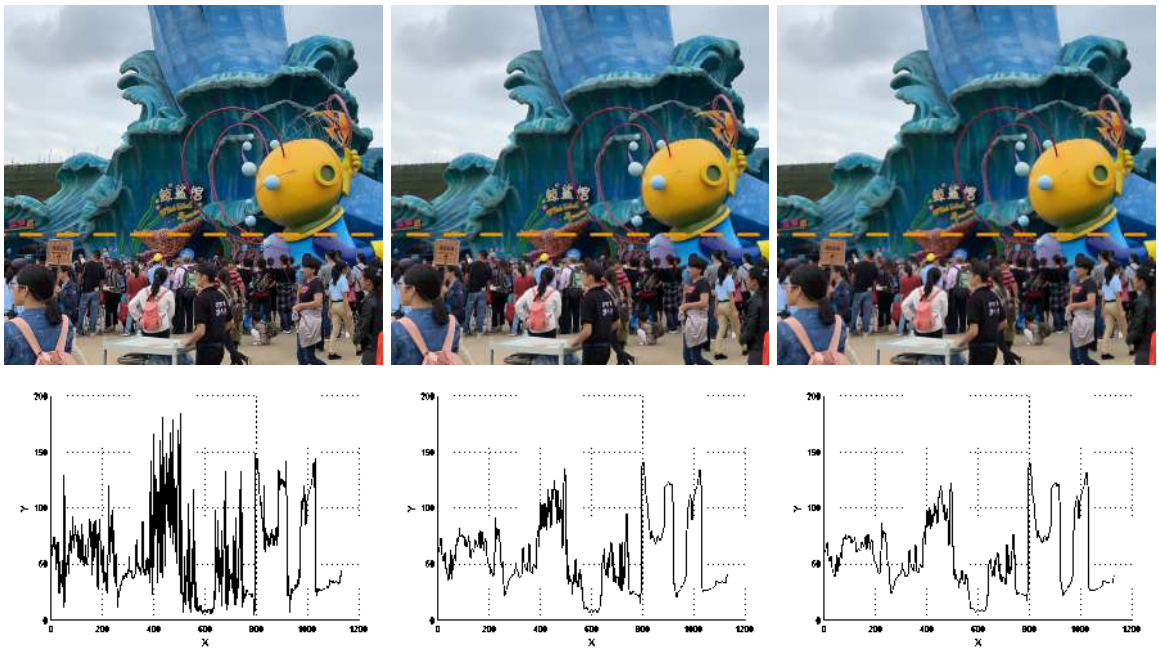

Figure 5. Mean Curvature PCF on a photo. From left to right are: origin photo, after 10, 30 iterations. The orange dash line profiles the intensity distribution of the image.

REFERENCES

1. Y. Gong and I. F. Sbalzarini, "Curvature filters efficiently reduce certain variational energies," *IEEE Transactions on Image Processing* **26**, pp. 1786–1798, April 2017.
2. Y. Gong, "Curvaturefilter." https://github.com/YuanhaoGong/CurvatureFilter.
3. M. Ibrahim, K. Chen, and C. Brito-Loeza, "A novel variational model for image registration using gaussian curvature," *arXiv preprint arXiv:1504.07643* , 2015.
4. S.-H. Lee and J. K. Seo, "Noise removal with gauss curvature-driven diffusion," *IEEE Transactions on Image Processing* **14**(7), pp. 904–909, 2005.
5. H. Zhu, H. Shu, J. Zhou, X. Bao, and L. Luo, "Bayesian algorithms for pet image reconstruction with mean curvature and gauss curvature diffusion regularizations," *Computers in Biology and Medicine* **37**(6), pp. 793–804, 2007.
6. A. I. El-Fallah and G. E. Ford, "Mean curvature evolution and surface area scaling in image filtering," *IEEE Transactions on Image Processing* **6**(5), pp. 750–753, 1997.
7. L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: nonlinear phenomena* **60**(1-4), pp. 259–268, 1992.
8. Y. Gong and I. F. Sbalzarini, "Local weighted gaussian curvature for image processing," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pp. 534–538, IEEE, 2013.
9. S. Cook, *CUDA programming: a developer's guide to parallel computing with GPUs*, Newnes, 2012.
10. N. Matloff, "Programming on parallel machines," *University of California, Davis* , 2011.
11. C. Ding, "Cuda tutorial." http://geco.mines.edu/tesla/cuda_tutorial_mio.