Applied Mathematics & Information Sciences
An International Journal

Parallel Catmull-Rom Spline Interpolation Algorithm for Image Zooming Based on CUDA

Tunhua Wu[1] *, Baogang Bai*[1] *and Ping Wang*[2]

[1] School of Information and Engineering, Wenzhou Medical College, Zhejiang 325035, China
[2] Department of Environmental Science, Wenzhou Medical College, Zhejiang 325035, China

Received: Apr. 22, 2012; Revised Jun. 25, 2012; Accepted Jul. 7, 2012
Published online: 1 Mar. 2013

Abstract: In order to scale video image real-timely, a GPU-aided parallel interpolation algorithm was proposed. Catmull-Rom Spline algorithm for image zooming was reformed into SIMD (Single instruction, multiple data) mode according to CUDA programming model. Re-sampling of each pixel was completed by a GPU thread. Hence, time-consuming re-sampling procedure of the whole zooming process were handled by parallel threads. The proposed algorithm runs hundreds times faster than traditional algorithm in experiments, and the speed is fast enough for scaling video frames real-timely. In addition, this algorithm can be extended to solve many other image processing related problems, such as image denosing and image segmentation.

Keywords: CUDA, GPU, SIMD, Catmull-Rom Spline, Image Zooming.

1. Introduction

With the development of high-definition flat-panel display technology as well as the popularity visual media (such as Internet TV and HDTV), the demands for high-definition images and videos grow rapidly. Usually speaking, to get high-definition media require more investment for purchasing new shooting equipment. But there exists a large number of old shooting equipment, and they are still usable. Hence, if the images or videos captured by old equipment could be enlarged to suitable resolution by software real-timely, we would greatly cut the expenses. In order to scale the video frames real-timely, a GPU-aided (Graphic Processing Unit-aided) parallel interpolation algorithm was proposed.

Spline interpolation is a mainstream method on image zooming. Non-linear algorithms, such as Neural Spline and S-Spline, have better zooming results, but they are generally time-consuming [1,2]. Hence, linear methods are widely used in the practical applications. Bi-Cubic Spline and Catmull-Rom Spline are the two most typical linear methods [3–5]. Through experiment, we found that the scaled images utilizing Catmull-Rom Spline interpolation have better resolution than those utilizing Bi-Cubic Spline interpolation (i.e. preserving more high-frequency information). However, Catmull-Rom Spline interpolation is still not a real-time method to handle the high-resolution images. So, the speed of interpolation should be improved. The best way to solve this problem is to use parallel computing.

CUDA (Compute Unified Device Architecture) is the GPGPU (General Computation on GPU) model developed by NVIDIA [6–9]. It can realize parallel computing on graphics cards. Usually, GPU is composed of dozens or even hundreds of SPs (stream processors). And there are thousands of parallel threads running in GPU. As everyone knows, if we want to scale image, gray value of each pixel on target image should be estimated by inverse mapping the pixel to source image. This process is called re-sampling. For every pixel of target image, the re-sampling method is identical. So, we may describe the re-sampling method as a kernel function which can be executed in SIMD mode under CUDA environment. Under this assumption, each GPU thread completes the re-sampling of corresponding pixel on target image. For the involvement of parallel computing, the image zooming speed may be increased dramatically.

CUDA parallel computing research has been carried out for about four years. In the research area of image

* Corresponding author: e-mail: fruitful@xmu.edu.cn

zooming, Ruijters and Gui combined Bi-Cubic Spline interpolation algorithm with CUDA technology respectively [10, 11]. Using their methods, the zooming speed can be improved 100-200 times. Xiao proposed a CUDA-aided Bi-Linear interpolation algorithm [12]. The algorithm results in a speed-up of 28 times on 2048*2048 images. At present, there is no report on the research of CUDA-based parallel Catmull-Rom Spline interpolation. So, this paper may fill the gap.

2. CUDA architecture

CUDA is NVIDIA version GPGPU model. It is a library for parallel computing on graphics card. The program language is CUDA-C, a special edition of C language. And the instructions will be converted to GPU-oriented codes by graphics card driver. Generally, GPU contains dozens or even hundreds of stream processors. NVIDIA GeForce 9600GT, for example, contains 64 stream processors, and its computing capacity reaches 312 GFlops.

2.1. Multi-threading model

There are thousands of threads running in GPU. After the compute-intensive part of application had been transferred from CPU (Host) to GPU (Device), GPU will house a large number of threads to work together. All the threads read and process their own data set individually, but the procedures they perform are identical. The content executed on GPU can be generalized to a specific module called kernel function (Kernel). That is, kernel function should be invoked and controlled by Host, and it is executed in Device. A group of threads running the same kernel function form a block. And block is a basic unit of management. Fig.1 illustrates the relationship among thread, block and grid. A number of threads make up a block (512 threads at most), while some blocks can form a grid. The device may contain many kernel functions running simultaneously. Each grid corresponds to a particular Kernel. Thread within a block can be identified by threadID. Likewise, block within a grid can be identified by blockID. The size of block and grid should be assigned before the Kernel can be executed.

2.2. Storage area

The storage area of GPU can be divided into the following sections: registers, local memory, shared memory, global memory, constant memory and texture memory. Each thread has its own register and local memory. Every thread in the same block shares a common storage space (i.e. share memory). And all the threads in the same grid share a part of global memory, constant memory and texture memory. When a Kernel needs to be executed, CPU will deliver the required data from main memory to device memory (such as global memory and texture memory) at first. Then, CPU will invoke a great many of threads to execute the Kernel. Finally, the results will be sent back to main memory for further processing.

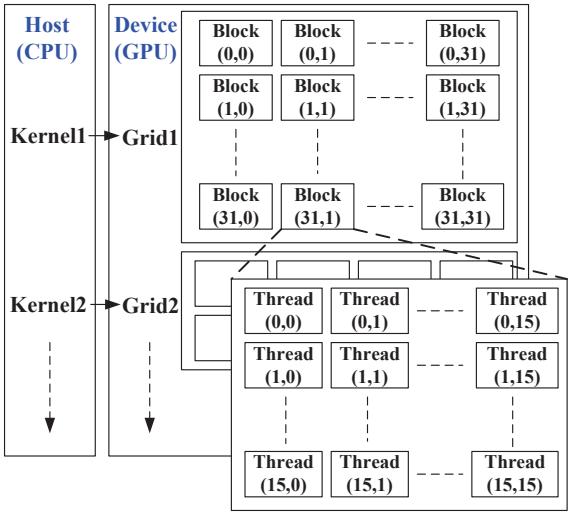

Figure 1: A multi-threading model of CUDA. 32*32 blocks make up Grid1, and 16*16 threads make up each block. Each thread in Grid1 executes the same instructions of Kernel1 (in SIMD mode).

3. Catmull-Rom spline interpolation

Catmull-Rom Splines are a family of cubic interpolating splines formulated such that the tangent at each point p_i is calculated using the previous and next point on the spline, $\tau(p_{i+1} - p_{i-1})$, $0 \leq \tau \leq 1$. Consider a single Catmull-Rom segment, $p(s)$. Suppose it is defined by four control points, p_{i-1}, p_i, p_{i+1} and p_{i+2}. And $p(s)$ is the interpolated curve between p_i and p_{i+1}.Then $p(s)$ can be expressed by:

$$p(s) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix} \quad (1)$$

Fig.2 illustrates the curve segment. Catmull-Rom Splines have C^1 continuity, local control, and interpolation. The parameter τ is known as $''tensionfactor''$ and it affects how sharply the interpolated curve bends at the control points. To increase the value of τ is to make the curve more sharply at the control points. It is often set to 1/2.

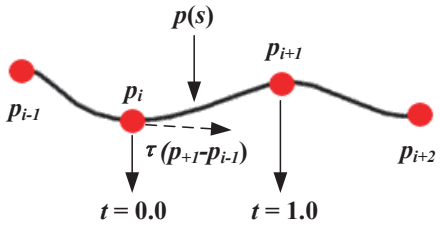

Figure 2: A curve segment $p(s)$ between p_i and p_{i+1} interpolated by Catmull-Rom Spline. The tangent at point p_i is $\tau(p_{i+1} - p_{i-1})$, τ = 1/2.

4. Image zooming based on parallel Catmull-Rom spline interpolation

Image zooming is a process of re-sampling. Suppose the scaling ratio in horizontal direction and vertical direction is M and N respectively. Then, any pixel in the target image (x', y') should be inverse mapped into source image. Suppose the corresponding point of (x', y') is (x, y), then $x = x'/M$, $y = y'/N$ Generally, the value of x and y are not integer, and the gray value at (x, y) should be estimated. Assume that $P(x, y)$ is the gray value of (x, y) on source image. Then, $P(x, y)$ can be estimated by bidirectional Catmull-Rom Spline interpolation in a 4*4 neighborhood of (x, y). Fig.3 is a schematic diagram of re-sampling. As shown on Fig.3, the bidirectional interpolation procedure can be divided into two steps: (**1**) To generate four curves in horizontal direction by Catmull-Rom Spline interpolation. Four points with the same horizontal coordinate with (x, y) on the curves should be recorded; (**2**) To generate a curve based on the recorded points by Catmull-Rom Spline interpolation. And the point with the same vertical coordinate with (x, y) on the curve is the target point. The method for re-sampling can also be described as follows:

Step 1. For any point on target image (x', y'), calculate the corresponding point (x, y) on source image by inverse mapping: $x = x'/M$, $y = y'/N$, M and N are scaling ratio. Let $u = floor(x)$, $v = floor(y)$, then the neighborhood of (x, y) is made of 4*4 pixels, denoted by $(u+i, v+j)$, i and $j \in [-1, 2]$. Let $t_x = x-u$, $t_y = y-v$, t_x and $t_y \in [0, 1]$.

Step 2. Perform Catmull-Rom Spline interpolation on $\{P(u-1, v-1), P(u, v-1), P(u+1, v-1), P(u+2, v-1)\}$, $\{P(u-1, v), P(u, v), P(u+1, v), P(u+2, v)\}$, $\{P(u-1, v+1), P(u, v+1), P(u+1, v+1), P(u+2, v+1)\}$ and $\{P(u-1, v+2), P(u, v+2), P(u+1, v+2), P(u+2, v+2)\}$ respectively. The interpolated curves are C_1, C_2, C_3 and C_4. Then, record the gray values of the curves when $t = t_x$. The gray values are denoted by: $Q_1 = C_1(t_x)$, $Q_2 = C_2(t_x)$, $Q_3 = C_3(t_x)$ and $Q_4 = C_4(t_x)$.

Step 3. Perform Catmull-Rom Spline interpolation on $\{Q_1, Q_2, Q_3, Q_4\}$, that will generate an interpolated curve C_5. The re-sampling result of (x', y') is $C_5(t_y)$.

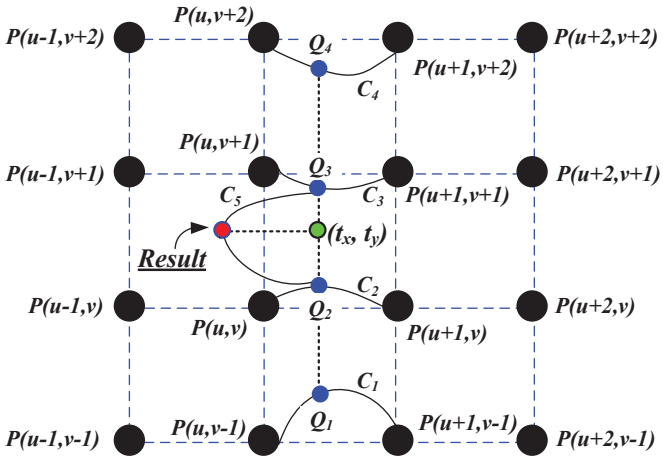

Figure 3: A schematic diagram of re-sampling. The gray value of (x, y) is the result. (t_x, t_y) is the relative coordinate of (x, y), $t_x = x - floor(x) = x - u$, $t_y = y - floor(y) = y - v$. C_1, C_2, C_3 and C_4 are the interpolated curves in horizontal direction, $Q_i = C_i(t_x)$. C_5 is the interpolated curves in vertical direction. The result is $C_5(t_y)$.

For every pixel of target image, the re-sampling method is identical. So, re-sampling method can be described as a kernel function which can be executed in SIMD mode under CUDA environment. In other words, if target images have N pixels, the kernel function should be executed in parallel by N GPU threads. As the communication latency between main memory and device memory is very high, it is inefficient to retrieve source image data in main memory during the process of resampling. In addition, thousands of threads run in parallel on GPU. When the threads frequently access source image data which were stored in main memory, the overall execution speed would be dropped greatly. The communication latency between GPU and device memory is very low. Therefore, source image data must be pre-stored in device memory. CUDA-based parallel re-sampling method can be described as follows:

Step 1. Load the image data (gray values) into main memory, and transfer the data into a device memory space of CUDA Array type by $cudaMemcpyToArray$ function in $cudaMemcpyHostToDevicemode$.

Step 2. Bind texture memory of GPU to the CUDA array by $cudaBindTextureToArray$ function. In this way, the source image data are put into texture memory. The source image data are taken as a texture lookup-table during the re-sampling process.

Step 3. Generalize the re-sampling procedure to a kernel function that will be executed in parallel by GPU. Each GPU thread executes a copy of the kernel function. In addition, when we want to define a kernel function, an identifier $''\_\_global\_\_''$ should be placed on the head of it.

Step 4. Determine the size of block and grid. According to the analysis of CUDA Occupancy calculator, a block of 16*16 threads is the best choice for our experimental system. As the number of pixels and threads is same, suppose the image resolution is $U * V$, then there are $U * V$ threads in a grid. So, the grid is composed of $(U/16) * (V/16)$ blocks.

Step 5. Invoke the parallel computing. Assume that name of the kernel function is $CatRom\_Resampling$, and a linear array d_output in device memory stores the resampling results. Then, execute the following code will get the results:

$CatRom\_Resampling$ <<<gridSize, blockSize>>> (d_output, *width*, *height*, *scale*);

Here, scale is the scaling ratio, width and height is the size of source image. In the parallel implementation of the kernel function, we must determine the corresponding thread index and block index of each pixel on target image. In this way, a unique thread corresponding to a specific pixel can be positioned. Then map the point of target image back to source image, and apply Catmull-Rom Spline interpolation algorithm to estimate the gray values of target points.

5. Experiments and analysis

The experimental system was developed by Microsoft Visual Studio 2008, NVIDIA CUDA Toolkit 3.2 and GPU Computing SDK 3.2. The last two tools can be downloaded from NVIDIA website freely, and the source code is open. The graphics card is NVIDIA 9600GT which has 512M device memory with 57.6GB/s bandwidth and 8 stream multiprocessors (i.e. 64 stream processors). Main supporting hardware include Intel Core2 E8400@4.0G (over-clocked), 4G DDR2-800 Memory and Intel P45 Motherboard. The testing platform is Windows 7. Driver version of the graphics card is 270.81.

The original resolution of the source images is 128*128, and the data size is 16K pixels. Fig.5 illustrates the results of three image zooming algorithms. Catmull-Rom Spline interpolation generates more clear and sharp results. The source images were scaled by 2X to 14X (i.e. the data size very from 32K pixels to 224K pixels). Tab.1 shows the detail performance on CPU and GPU. The zooming procedure running on GPU was divided into two steps: (**1**) Send source image data to texture memory from main memory (called texture fetching); (**2**) Take the texture as a lookup table for interpolation. As the data size of source image is constant, execution time of texture fetching is constant too. In experiment, execution time of texture fetching on images of 128*128 pixels is 0.131ms. Time cost of the two steps mentioned above makes up the actual execution time on GPU. As shown in Fig.4, the zooming speed of GPU is significantly faster than CPU. And the relative speed grows with the increment of image data size. That is because when the image is small, the computation of re-sampling is so small that the time cost of texture fetching takes up a large proportion of total execution time. Namely, the advantage of parallel processing is unobvious when data size is small. With the increase of the image data size, time cost of re-sampling gradually becomes the dominant part of total execution time. In addition, the scaled images generated by GPU and CPU are almost same. As NVIDIA 9600GT has no support for native double-precision calculations, interpolation accuracy of GPU is not as good as CPU.

The proposed GPU-aided parallel image zooming algorithm can real-timely change resolution while maintaining acceptable quality. This algorithm can be directly applied to many real-time video processing systems, such as video-surveillance and IPTV (network TV). In addition, this algorithm can be extended to many other image processing-related issues, such as Levelset-based image segmentation method[13]: each GPU thread handles the iterative computing of an image pixel, which may dramatically improve the speed of curve evolution.

Table 1: Comparison of image zooming speed between CPU and GPU

Scale	Data Size (pixels)	CPU (ms)	GPU (ms)			Speed up (times)
			Texture Fetching	Resampling	Total	
2X	32K	31.2	0.131	0.206	0.337	92.6
3X	48K	62.4	0.131	0.316	0.447	139.6
4X	64K	93.6	0.131	0.465	0.596	157
5X	80K	140	0.131	0.661	0.792	176.8
6X	96K	203	0.131	0.889	1.02	199
7X	112K	280	0.131	1.185	1.316	212.8
8X	128K	363	0.131	1.517	1.648	220.3
9X	144K	436	0.131	1.804	1.935	225.3
10X	160K	546	0.131	1.992	2.123	257.2
11X	176K	639	0.131	2.18	2.311	276.5
12X	192K	780	0.131	2.369	2.5	312
13X	208K	889	0.131	2.561	2.692	330.2
14X	224K	1029	0.131	2.661	2.792	368.6

6. Conclusion

A GPU-accelerated image zooming algorithm based on Catmull-Rom interpolation was proposed. With the strong computing power of GPU and the parallel computing function of NVIDIA CUDA, the zooming procedure was completed at high speed in graphics card. The speed is fast enough for zooming video frames real-timely. In the meantime, its zooming effect remains favorable for practical applications. Experimental results show that the parallel computing in SIMD mode can greatly improve the efficiency of Catmull-Rom Spline interpolation. And the speed grows with the increase of image data size. In addition, the proposed algorithm can be extended to solve other image

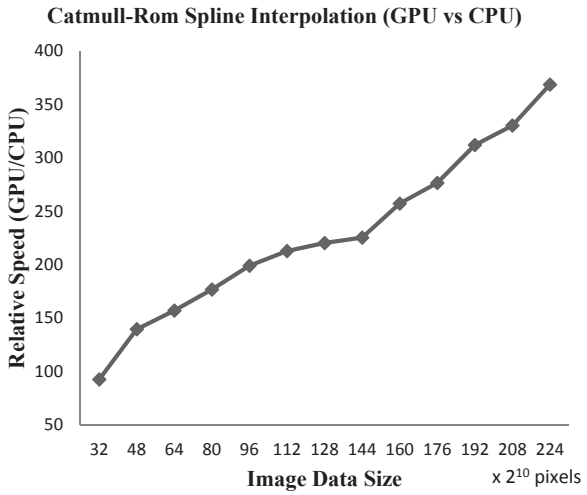

Figure 4: Relative speed of image zooming (GPU/CPU) increases with increasing image data size. The increment of image data size is 32K pixels.

processing-related problems, such as image denosing and image segmentation. To further enhance the performance of image zooming, non-linear image zooming algorithms should also be reformed to SIMD mode.

Acknowledgement

This work is supported by the Natural Science Foundation of China (No.11005081, 60873179), Natural Science Foundation of Zhejiang Province (No.Y1110322), Scientific Research Project of Zhejiang Education Department (No.Y201016244) , Scientific Research Project of Wenzhou (No.G20110004) and the Scientific Research Project of Wenzhou Medical College (No.QTJ09009).

References

[1] I.G. Tsoulos, I.E. Lagrais and A. Likas, Journal Neural, Parallel & Scientific Computations **13**, 161 (2005).
[2] Q. Zhang, S.H. Zhou and X.P. Wei, Applied Mathematics & Information Sciences **5**, 445 (2011).
[3] R. Keys, IEEE Transactions on Acoustics, Speech, and Signal Processing **29**, 1153 (1981).
[4] D. Ruijters, B.H. Romeny and P. Suetens, Journal of Graphics Tools **13**, 61 (2009).
[5] C. Yuksel, S. Schaefer and J. Keyser, Computer-Aided Design **43**, 747 (2011).
[6] Y. Liu, B. Schmidt, W. Liu and D.L. Maskell, Pattern Recognition Letters **31**, 2170 (2010).
[7] S. Singh, Communications of the ACM **54**, 46 (2011).
[8] H. Park and P.A. Fishwick, ACM Transactions on Modeling and Computer Simulation **21**, 18 (2011).

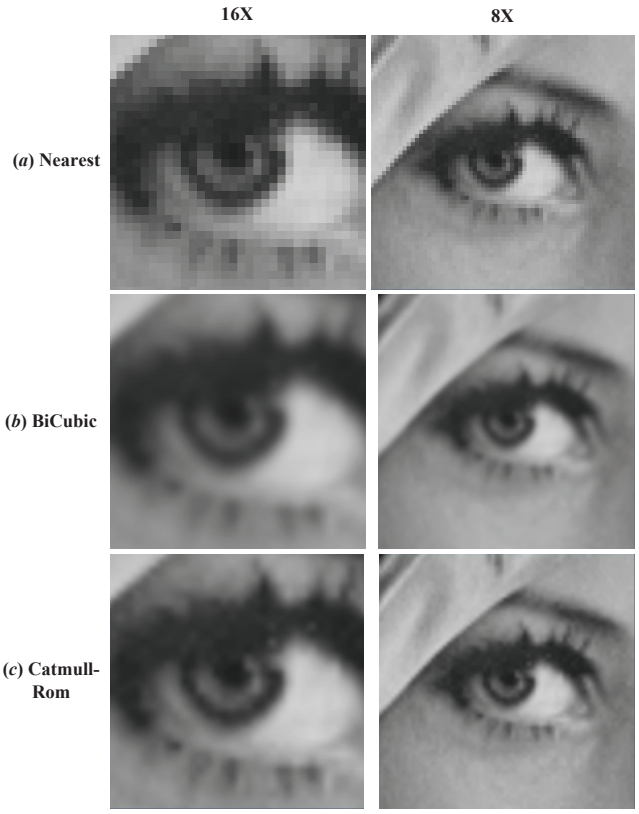

Figure 5: Comparison among three image zooming algorithms under CUDA environment. Catmull-Rom Spline interpolation generates more clear and sharp results.

[9] X.B. Gan, Z.Y. Wang, L. Shen and Q. Zhu, Applied Mathematics & Information Sciences **5**, 129S (2011).
[10] D. Ruijters and P. Thevenaz, The Computer Journal **55**, 15 (2012).
[11] Y.C. Gui, Q.J. Feng, L. Liu and W.F. Chen, Chinese Journal of Computer Engineering and Applications **45**, 183 (2009).
[12] H. Xiao, Chinese Journal of Computer Systems **32**, 2241 (2010).
[13] T.F. Chan and L.A. Vese, IEEE Transactions on Image Processing **10**, 266 (2001).

Dr. Tunhua Wu received his Ph.D from Xiamen University in 2008. His research interests lie in the areas of digital image analysis, pattern recognition and computer graphics. Email: wth@zjut.edu.cn.