Tensor Voting Accelerated by Graphics Processing Units (GPU)

Changki Min and Gérard Medioni
University of Southern California
Integrated Media Systems Center
Los Angeles, CA 90089, USA
{cmin, medioni}@usc.edu

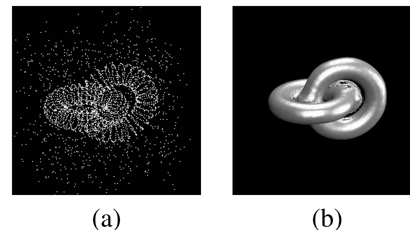
Abstract

This paper presents a new GPU-based tensor voting implementation which achieves significant performance improvement over the conventional CPU-based implementation. Although the tensor voting framework has been used for many vision problems, it is computationally very intensive when the number of input tokens is very large. However, the fact that each token independently collects votes allows us to take advantage of the parallel structure of GPUs. Also, the good computing power of modern GPUs contributes to the performance improvement as well. Our experiments show that the processing time of GPU-based implementation can be, for example, about 30 times faster than the CPU-based implementation at the voting scale factor $\sigma = 15$ *in 5D.*

1. Introduction

One of successful perceptual organization tools in the computer vision area is tensor voting. It was first introduced by Guy and Medioni [3], and has served for many vision problems. The main function of the framework is to extract geometrical features from given set of N-D points. In a 3D space, for instance, we can simultaneously extract junctions (or isolated points), curves, and surfaces from the 3D input points. Here, the points can be either unoriented or oriented, where oriented points are associated with surface normals or curve tangents. Figure 1 shows an example which extracts surfaces from the set of 3D input points. Although the input contains many noisy points, the tensor voting framework can successfully remove them and generate two smooth tori from the inlier points.

Using the geometrical function of the tensor voting framework, we can also solve many other vision problems. Since the framework finds smooth geometric features in N-D spaces, problems which satisfy the following conditions can be solved by using the tensor voting framework:



(a) (b)

Figure 1. Surface extraction using tensor voting. (a) Input 3D points, (b) Extracted surface

- The problem can be formulated as grouping points in an N-D space
- The resulting groups (i.e., curves, surfaces, etc.) are locally smooth.

For instance, Mordohai and Medioni [9] applied the tensor voting framework to the multiple view stereo problem, Tang et al. [13] solved the problem of epipolar geometry estimation in an 8D space using the framework, and two-frame motion analysis was studied in [10] with the 4D tensor voting framework. Also, other problems such as inpainting [4], image correction [5], and affine motion estimation [6] have been studied with the framework.

Although the tensor voting framework itself does not limit applications as long as they satisfy the above conditions, sometimes it is not practical to use the framework with a large number of input points because the voting process is computationally very intensive. To overcome this limitation, we present a new GPU-based voting implementation which achieves significant performance improvement over the conventional CPU-based implementation.

During past several years, the performance of GPUs has dramatically improved. For instance, some GPUs achieve the memory bandwidth of 35.2 GB/sec, and 63 GFLOPS which is about 4.3 times faster than a 3.7GHz Intel Pentium4 SSE CPU [2]. A more recent GPU, NVIDIA GeForce

The 18th International Conference on Pattern Recognition (ICPR'06)
0-7695-2521-0/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

7800GTX, is known to achieve up to 54.4 GB/sec memory bandwidth and 200 GFLOPS. Due to the high performance of GPUs, even non-graphics frameworks are being developed by many researchers based on GPUs. Such research work is known as GPGPU (General Purpose GPU) computing, and [11] discusses various recent developments in GPGPU computing.

Since GPUs have been developed and optimized especially for graphics-oriented tasks, it is important to note that the tasks which have 1) high independence between data elements, 2) high parallelism, 3) intense arithmetic computation, and 4) a large number of data, can take advantage of the power of GPUs. The tensor voting framework perfectly satisfies the above requirements. Especially, the parallel architecture of GPUs allows many tokens to collect (or cast) votes *simultaneously*, and it is the main source of the significant speed improvement when the tensor voting framework is implemented in GPUs. A large number of input tokens and the intense arithmetic computation for voting are also efficiently handled by GPUs.

This paper is organized as follows. The following section 2 briefly introduces the tensor voting framework, and section 3 explains the details of the tensor voting implementation in GPUs. The performance comparison between GPU and CPU is presented in section 4 followed by our conclusion and future work in section 5.

2. Brief overview of tensor voting

Due to limited space, we do not present all the details of the tensor voting framework here. Rather, we refer readers to [8][7] for the complete tensor voting theory.

The tensor voting framework has two elements: *tensor calculus* for data representation, and *tensor voting* for data communication. Each input point is initially encoded as a tensor which is a symmetric nonnegative definite matrix. The shape of the tensor defines the type of geometric feature (e.g., point, curve, surface, etc.), and the size defines its saliency, or confidence measure.

After the encoding step, each token (a point with its associated tensor) casts votes to its neighboring tokens based on predefined voting kernels. Each voting kernel is a tensor field, and it encapsulates all voting-related information such as the size and shape of the voting neighborhood, and the vote strength and orientation.

The basic idea of the voting kernel can be explained by the *fundamental 2D stick field*, and this is illustrated in Figure 2. Assume that we are computing the vote cast from the token O (i.e., voter) to P, and the normal $\overrightarrow{N}$ is known for the voter. To generate the vote, we must consider two things: the orientation and strength of the vote (Figure 2(a) and (b), respectively). The orientation (gray arrow starting from P) is given by drawing a big circle whose center

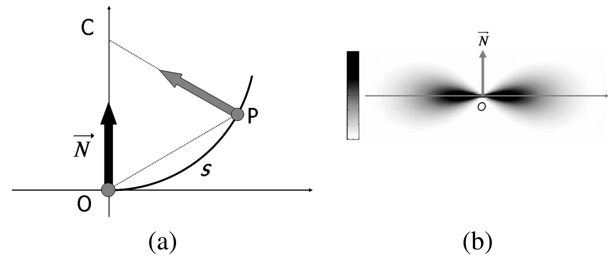


Figure 2. Fundamental 2D stick field. (a) orientation, (b) intensity-coded strength

is in the line of $\overrightarrow{N}$ (in this case, it is at C), and it passes both O and P while preserving the normal $\overrightarrow{N}$. This process ensures the smoothest connection between two points, O and P, with associated normals. The strength of the vote is computed by the following decay function:

$$DF(s, \kappa, \sigma) = e^{-\left(\frac{|s|^2 + c\kappa^2}{\sigma^2}\right)}.$$

Here, $|s|$ is the arc length, κ is the curvature, c controls the degree of decay, and σ is the scale of voting (neighborhood size). By rotating and integrating the fundamental 2D stick field, we generate all other voting fields such as ball fields, plate fields, and any higher dimensional voting fields.

During the voting process, each input token collects votes from its neighbors by tensor addition, and the final tensor at the token is analyzed to measure the saliency of each geometric feature.

3. Tensor voting implementation with GPUs

3.1 Voting mode

The parallel structure of the voting can be implemented in two different modes: (1) vote-collection mode, (2) vote-cast mode. In the first case, all tokens *simultaneously collect* votes from the token which casts votes (Figure 3(a)). In the second case, all tokens *simultaneously cast* votes to the token which collects votes (Figure 3(b)). We follow the first mode because it is more suitable to the current GPU architecture.

3.2 Implementation

The memory structure of GPUs is quite different from CPUs in that GPUs do not have read-and-write memory. Instead, they have separate read-only memory (texture memory) and write-only memory (frame buffer memory). For the tensor voting implementation, we load all input tokens

IEEE COMPUTER SOCIETY

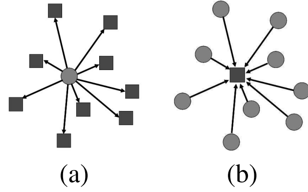


Figure 3. Implementation of the parallel structure of voting: (a) vote-collection mode, (b) vote-cast mode.

into the read-only texture memory, and write the cast votes to the write-only frame buffer memory (more specifically, we use offscreen buffers using FBO (Framebuffer Object), which can be found in [1]).

For N-D tensor voting, each token consists of $(2N + N^2)$ floating number elements: one N-D position vector, N eigenvalues, and N N-D eigenvectors. For instance, a token in 5D space has 35 elements, and we use 9 textures to store all the 5D tokens because each texture element can store up to 4 values, RGBA. This is illustrated in Figure 4. Note that the texture elements (t_x, t_y) in all 9 textures correspond to a single input token.

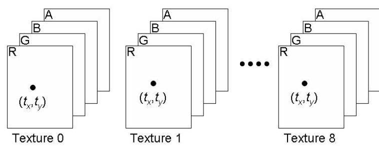


Figure 4. Texture memory setup for 5D

After storing all tokens into the texture memory, we setup a for-loop in the CPU code. This for-loop sets an input token as a voter (i.e., the token which casts votes to other tokens) one at a time until all tokens are processed. For each iteration, the CPU simply tells the GPU which token is the current voter. Then, in the GPU, *all* tokens in the neighborhood except the voter collect the information of the voter from the texture memory, and compute votes from it *simultaneously*. This parallel vote computing process is the main contribution of the GPU, and it dramatically reduces the overall voting processing time. In contrast, the CPU-based implementation allows only a single token at a time to compute a vote from the voter, which is the main bottleneck. Through the iteration, the computed votes at each token are accumulated in the offscreen frame buffer memory via *ping-pong buffering* technique [12]. When the iteration is completed, the frame buffer memory is copied to the CPU main memory for further processes. Figure 5 shows the overall structure of the GPU-based tensor voting implementation. The *parallel* voting process in the GPU is represented as a gray box.

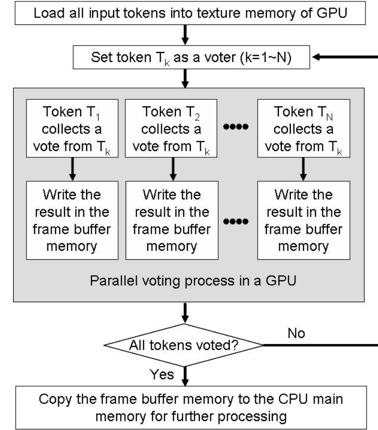


Figure 5. Structure of the GPU-based tensor voting implementation. N is the total number of input tokens.

The time complexity of the CPU-based implementation is $O(DN \log N)$, where D is the dimension of a space, N is the number of input tokens, and $\log N$ is for searching neighboring tokens. Usually, N is much larger than D so that the overall complexity is dominated by N. Assuming GPUs have N processing units, the time complexity becomes $O(N)$ because for each voter all N tokens compute votes simultaneously without searching neighbors of the voter. In practice, tokens which are far from the voter do not compute votes to save computation time because the votes are negligible.

4. Results

Our development system for the GPU-based tensor voting is summarized in Table 1. Although we have implemented 2D, 3D, 4D, and 5D tensor voting frameworks for GPUs, we present only the results of the 5D case (the most complicated one) due to limited space.

GPU	NVIDIA GeForce 7800GTX
GPU memory	256MB
Driver version	ForceWare 77.77
Shader	Cg 1.4
CPU	Intel Pentium4 3.2GHz
Main memory	2GB
Operating system	WindowsXP SP2

Table 1. Our development system

In order to compare the performance of GPU and CPU implementations, we tested 38,919 5D points. The points are encoded as three different tensor forms: 1) *ball* where

IEEE COMPUTER SOCIETY

the eigenvalues are set to (1,1,1,1,1), 2) *plate* where the eigenvalues are set to (1,1,0,0,0), 3) *arbitrary* where the eigenvalues are arbitrary numbers. The processing time (in seconds) of each encoding type for both GPU and CPU implementations and their ratios are shown in Table 2, and Figure 6, respectively. The *ball* tensor which requires the simplest vote computation is used in many applications, and we observe that the GPU-based code takes only 8 seconds to process all the 5D input points at $\sigma = 15$. This is huge improvement against the CPU-based code which takes 232 seconds (29 times faster). The *arbitrary* tensor requires the most complicated vote computation so that it takes more than a minute even for the GPU-based code. However, the GPU-based code still outperforms the CPU-based code.

σ	Ball		Plate		Arbitrary	
	GPU	CPU	GPU	CPU	GPU	CPU
5.0	7	66	15	104	53	282
7.5	8	106	16	199	67	636
10.0	8	154	17	309	79	1015
12.5	8	213	20	454	92	1619
15.0	8	232	21	625	105	2262

Table 2. Processing time comparison between GPU and CPU codes (in seconds)

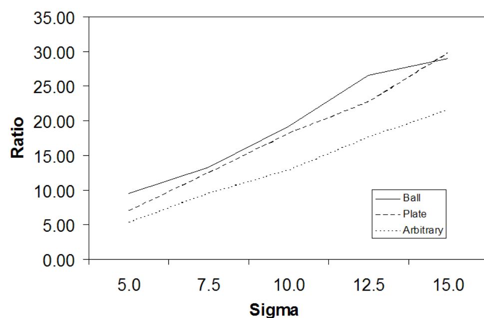


Figure 6. Processing time ratio of Table 2

5. Conclusions and future work

We have presented the new GPU-based tensor voting implementation, and the experimental results demonstrate its huge performance improvement. Thus, it allows the tensor voting framework to be used broader range of applications in which processing time might be crucial.

The current implementation, however, has some limitations. First, the maximum tensor voting dimension is limited to 5D because the number of offscreen frame buffers is limited with the current driver. Also, relatively small amount of GPU texture memory (current system has 256MB) restricts the number of input points. In fact, these issues originate from the current hardware and driver limitations. Thus, we will continue to update our implementation with their future releases to make the system faster and more flexible.

Acknowledgment

The research has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152, and U.S. National Science Foundation grant IIS 03 29247. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] http://www.opengl.org.
[2] I. Buck. Gpgpu: General-purpose computation on graphics hardware - gpu computation strategies & tricks. *ACM SIGGRAPH Course Notes*, August 2004.
[3] G. Guy and G. Medioni. Inferring global perceptual contours from local features. In *CVPR*, pages 786–787, 1993.
[4] J. Jia and C. Tang. Image repairing: robust image synthesis by adaptive nd tensor voting. In *CVPR*, pages I: 643–650, 2003.
[5] J. Jia and C. Tang. Tensor voting for image correction by global and local intensity alignment. *PAMI*, 27(1):36–50, January 2005.
[6] E. Kang, I. Cohen, and G. Medioni. Robust affine motion estimation in joint image space using tensor voting. In *ICPR*, pages IV: 256–259, 2002.
[7] G. Medioni and S. Kang. *Emerging Topics in Computer Vision*. Prentice Hall, 1st edition, 2004.
[8] G. Medioni, M. Lee, and C. Tang. *A Computational Framework for Segmentation and Grouping*. Elsevier, 1st edition, 2000.
[9] P. Mordohai and G. Medioni. Perceptual grouping for multiple view stereo using tensor voting. In *ICPR*, pages III: 639–644, 2002.
[10] M. Nicolescu and G. Medioni. Layered 4d representation and voting for grouping from motion. *PAMI*, 25(4):492–501, April 2003.
[11] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics State of the Art Reports*, pages 21–51, August 2005.
[12] M. Pharr. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, 2005.
[13] C. Tang, G. Medioni, and M. Lee. N-dimensional tensor voting and application to epipolar geometry estimation. *PAMI*, 23(8):829–844, August 2001.

0-7695-2521-0/06 $20.00 © 2006 IEEE