GPU Accelerated Simulation of Cardiac Activities

Rongdong Yu[1,2], Yin Zhang[1], and Sanyuan Zhang[1*]
[1] College of Computer Science, Zhejiang University, Hangzhou, China, 310027
[2] China Huadian Electric Power Research Institute, Hangzhou, China, 310030
Email: ylz@xs.hz.zj.cn, yinzh@zju.edu.cn, syzhang@zju.edu.cn

Patricia Chiang and Yiyu Cai
School of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore, 639798
Email: patricia.chiang@ieee.org, myycai@ntu.edu.sg

Jiangmin Zheng
School of Computer Engineering, Nanyang Technological University, Singapore, 639798
Email: asjmzheng@ntu.edu.sg

Koon-Hou Mak
Mak Heart Clinic, Gleneagles Medical Centre, Singapore, 258499
Email: makheart@gmail.com

***Abstract*—Much efforts have been made to develop realistic cardiac models for clinical and research purposes. However, to implement these models always needs to handle excessive computational loads due to the complex and dynamic natures of the heart given limited computational power of Central Processing Unit (CPU). In this paper, a real-time approach to cardiac modeling is proposed based on the Graphics Processing Unit (GPU). A hardware platform is first designed and tested with a simplified model to represent the cardiac activities. Motion of mesh-based heart is then approximated to simulate the movement of each vertex. Time functions are used within the GPU platform to describe the cardiac cycle. The parallel computing feature of GPU platform significantly speeds up the computation process in real-time with over 140,000 vertices motion based on the time functions. The program is developed on top of CUDA architecture proposed and developed by nVIDIA. Computational Experiments show visualization of the cardiac dynamics is significantly benefiting from this new solution. Further improvement of the GPU based cardiac simulation is discussed.**

***Index Terms*—GPGPU, CUDA, Cardiac Activities, Simulation**

I. INTRODUCTION

As a result of continued demand for programmability, modern graphics processing units (GPUs) such as nVIDIA's GeForce 8 Series or later generations are designed as programmable processors employing a large number of processor cores[1]. The fast increasing power of the GPU and its streaming architecture opens up a range of new possibilities for a variety of applications. With the enhanced programmability of GPUs, these chips are now being capable of performing not only the specific graphics computations they were originally designed for, but also algorithms for non-graphics applications, such as scientific computing[2], image processing[3], computational biology[4,5], Fast Fourier Transformation[6] and so on. The evolution of GPUs is mainly driven by the computer game market today. This leads to a relatively low unit price and to very rapid developments of next generation product. Most significantly, according to[7], the growth rate of the number of transistors embedded in GPUs is greater than for microprocessors and the peak performance of GPUs is approximately ten times faster than that of comparable CPUs.

However, there are still a number of challenges to be solved in order to enable wider acceptance of the GPU technologyby more users other than computer graphics specialists. Restrictions of the underlying streaming architecture have to be taken into account, e.g. random access type of writes-to-memory is not supported and no cross fragment data or persistent state is possible. Currently, all the internal registers are flushed before a new fragment is processed.

The two major GPU vendors, nVIDIA and AMD, just proposed their new developing platforms: the Compute Unified Device Architecture (CUDA) with nVIDIA[8] and Stream with AMD[9]. Unlike previous GPU programming models, these are proprietary approaches designed to allow a direct access to their specific graphics hardware. The vertex processor and fragment processor have already become programmable. Actually, both of them are designed as stream processors. This feature opens a new field in GPU application, called general-purpose

*Corresponding Author

doi:10.4304/jcp.5.11.1700-1705

computing on graphics processing units (GPGPU). CUDA is an extension of the C programming language, it treats the GPU as a data-parallel computing device without the need of mapping computations to the graphics pipeline; Stream is a virtual machine running proprietary assembler code and Brook+ programming language. Therefore, there is no compatibility concern between the two platforms. Presently, even with Open Computing Language (OpenCL), the so called cross-platform solution, its code has to be converted into one of the two platforms for compiling and running. However, both platforms have to overcome some major restrictions typically associated with previous GPGPU approaches, e.g., those set by the traditional graphics pipeline and the relative programming interfaces like OpenGL and Direct3D.

A. Prior works

Study on cardiac physiology shows that pacemaker cells in the heart generate action potentials at the beginning of each cardiac cycle, which are conducted to the whole heart through the conduction fibers spreading throughout the myocardium[10]. The myocardial contractile cells are then excited by the action potentials and contract according to the sequence of excitations, and the heart beats in a rhythmic motion. In order to properly model the cyclic dynamics of the heart, computational methods including many theoretical tools for the investigation of cardiac physiology and pathology have been developed. In the past two decades, many effective methods have been introduced aiming to improve the simulation accuracy of the electrical heart model. The cardiac electric wave propagation model (E model)[11], the electromechanical coupling model (EM model)[12], and the biomechanical model (BM model)[13] are among the most popular ones. Integration of these three models are also proposed to give a more complete macroscopic description of the physiological behavior of the heart; usually referred as the cardiac physiome model[14]. To simulate the heart cycle, the cardiac physiome model is used by experiments and validated through different criteria[12,15].

On the other hand, simplification strategy is often adopted especially in the cardiac physiome modeling where no closed form solution is given in order to identify what may be neglected while still obtaining acceptable results. For example, in the cardiac physiome model, the continuous space of the heart has to be discretized so that numerical methods can be applied to solve the formulations. The finite element methods (FEM) have been extensively used in computational cardiology. Using tetrahedral elements[15], meshing algorithm is relatively simple, yet the efficiency to assemble system matrices is considerable.

B. Our works

We have built a simulating system for cardiac intervention[16-18]. In our solution, boundary of heart lumens are detected by utilizing voxel, catheter deformation simulated by FEM method, is accelerated by GPU pipeline based algorithms, and achieves real-time performance. In this paper, algorithms based on GPU have been implemented using C++ and CUDA and tested on a tetrahedral mesh model consisting of vertices and tetrahedrons. Expereiments show the GPU is capable for high performance cardiac activities simulation.

II. METHODOLOGY

A. Anatomical Model

The mesh of the heart contains 148,516 vertices and 728,321 tetrahedrons[19]. The myocardium is represented as a tetrahedral volumetric mesh including anatomical information. The process to build such a model is detailed in[20]. The main anatomical information we used is the myocardium geometry, its division into different anatomical parts, including details of valves and valve gaps, see Fig. 1. Labeling of heart elements is done using boundary index in vertices (see Table 1).

B. Myocardium Mechanical Model

The myocardium constitutive law is complex and must include an active element for contraction, controlled by the trans-membrane potential computed in the previous section, and a passive element representing the mechanical elasticity. Several constitutive laws have been proposed in the literature[21- 26]. These laws are designed to precisely fit rheological tests made on *in vitro* cardiac muscle. Nash[12] has given a mechanical mathematic model of the ventricle activities. This model is based on the model generated from test of accurately anatomic cardiac muscle. In Nash's theory, the ventricular mechanics model divides the entire cardiac cycle into four phases, they are the diastolic filling, isovolumic contraction, ejection and isovolumic relaxation phases. During the cardiac cycle, the pressure of ventricle starts from the rest stress of the non-load static state and continuing increase in the cycle. Every state will be a quasi-static state generated from the pre-existing state as an initial state.

C. Simplified Mechanical Model

The ventricular mechanics model is efficient, but it has to use the Newton's method to minimize a system of nonlinear residuals with respect to the set of unknown solution variables. It is much complex, and no need for every vertex of the model.

TABLE 1.
THE CORRESPONDENCE BETWEEN THE BOUNDARY SIGN AND COMPONENTS.

Components	**Color**	**Components**	**Color**
interior	N.A.	pulmonary valve	red
mitral valve	red	right ventricle	blue
mitral valve	yellow	left ventricle	green
aortic valve	blue	valve of foramen oval	green
aortic valve	purple	valve of foramen oval	red
aortic valve	green	aorta	blue
right atrium	red	tricuspid valve	green
right pulmonary v.	red	tricuspid valve	purple
pulmonary valve	orange	tricuspid valve	pink
pulmonary valve	blue	right pulmonary a.	dark red
inferior vena cava	orange	outer surface	pink
left atrium	orange		

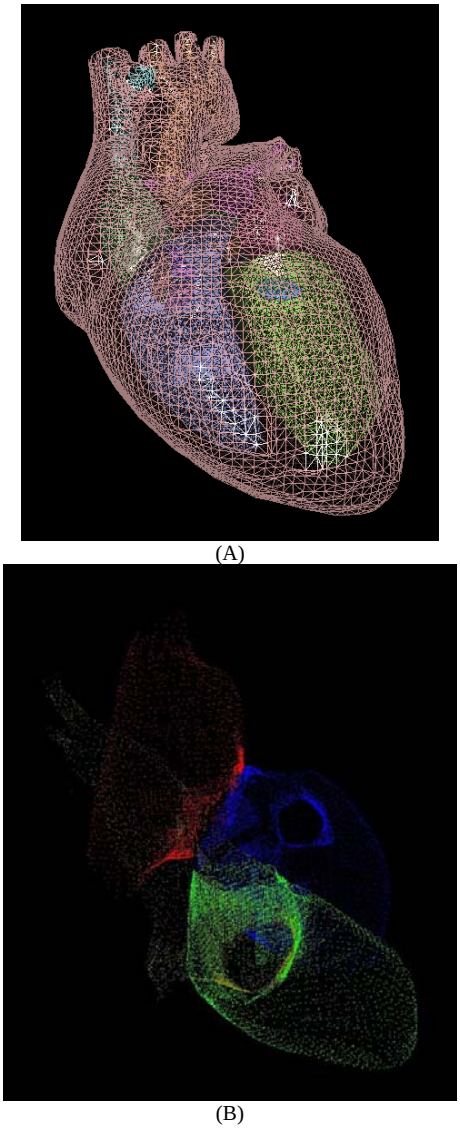

Figure 1. (A) The mesh of the heart model. (B) Points of the four lumens surfaces, marked with different colors for different parts.

First, we give some time functions to represent the relationship between location of the particles due to the state theory of the ventricular mechanics model.

The entire cycle of ventricle and atrium could be represented by a cosine equation,

$$S(t) = 1 + n \cdot \cos(kt + \phi) \quad (1)$$

with *S(t)* a zoom function by the time parameter *t*, *k* the cycle controller coefficient, ϕ the initial phase, and *n* the magnify coefficient.

Due to the normal heart cycle, for the motion extended in different speed through parts, so we give the particles of ventricle and atrium with different values of *n* and ϕ.

To simulate the motion of the cardiac muscle, we get the position limit h_i of one particle *i* in the cardiac muscle first, and particles within different lumens have their own h_i values. For one lumen, *c* represents the position of the center of mass. The position of particle i at the time of t can be expressed by

$$x_i = c + S(t)(h_i - c) \quad (2)$$

where xi is the location of the particle i at the time t.

According to the algorithm the computational complexity is $O(N)$, it seems much fast, but for more than 140,000 particles system, to compute so many vertices' coordinate is time-consuming. We find the solution could been easily implemented in a parallel way, naturally the best way to solve this problem is to use the Single Instruction Multiple Data (SIMD) or Multiple Instruction Multiple Data (MIMD) parallel machines with dozens of processors and compute the position of each particle apart.

All these approaches can be seen as accelerators - an approach satisfying the demand for a low cost solution to compute-intensive problems. The main advantage of GPUs compared to the architectures mentioned above is that they are commodity components.

D. CUDA Programming Model

In CUDA, the GPU is viewed as a compute device suitable for parallel data applications. It has its own device random access memory and may run a very high number of threads in parallel. Threads are grouped in *blocks* and many blocks may run in a *grid* of *blocks*. Such structured sets of threads may be launched on a *kernel* of code, possessing the data stored in the device memory. Threads of the same *block* share data through fast shared on-chip memory and can be synchronized through synchronization points. Theoretically, having more active threads per multiprocessor can help hide memory latency, and can also better fill the instruction pipeline so there are no idle processors. A multiprocessor is a set of stream processors. In CUDA architecture, a multiprocessor contains 8 individual stream processors. In the software level, one *block* runs on one multiprocessor. According to[27], the maximum number of threads that can run concurrently on a multiprocessor is 1024 in the CUDA architecture of version number later than 1.2. In practice, the number of threads is further limited by the shared on-chip memory and hence, the maximal number of active threads per multiprocessor is application-dependent.

In CUDA architecture as shown in Fig. 2, a multiprocessor has on-chip memory of four types:

(1) One set of registers per processor,
(2) A parallel data cache or shared memory,
(3) A read-only constant cache,
(4) A read-only texture cache.

These on-chip memories are used to implement fast I/O operations, especially, to speed up read and write access to the non-cached device memory. Thus, applications can take advantage of them by minimizing over-fetch and roundtrips to the low bandwidth device memory. Although the device memory has a low bandwidth, it is big in size and shared by all multiprocessors.

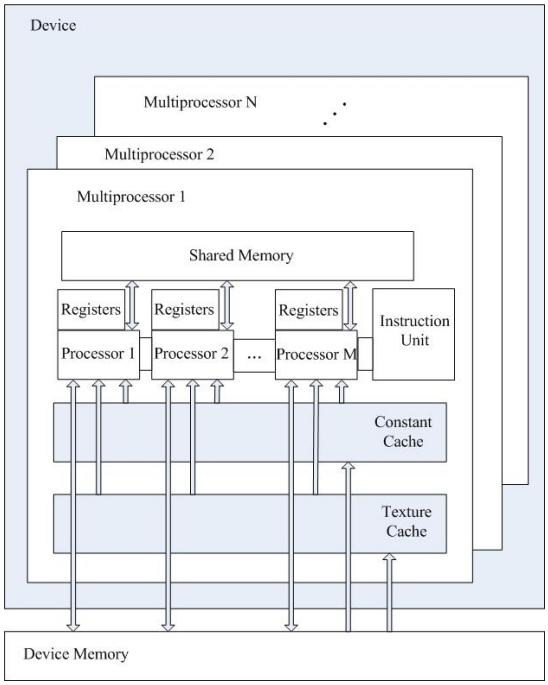

Figure 2. Hardware architecture of CUDA based processors.

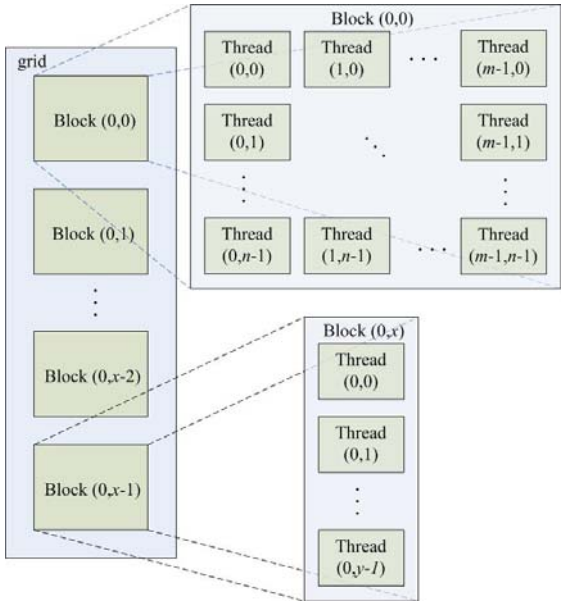

Fig.ure 3. Threads and block assignment. In one block of first type there are 512 threads, so for threads indices *m* and *n*, *m***n*=512, and blocks index *x*, *x*=[*N*/512]+1, *N* is the total number of threads. In the last block, *y*=*N*-(*x*-1)*512.

E. CUDA Based Simulation Algorithm Implementation

For the parallel applications many algorithms have been proposed and implemented, because of too many application-dependent and architecture-dependent characters to consider. In this simulation algorithm, computing of one particle position is implemented by one thread, and we just put the particles to be processed in one array, so the account of threads is equal to that of particles *N*. These threads are distributed to the multiprocessors, each processor is assigned a subset of *N/P* (*P* is the number of processors) particles. For threads assignment, we set number of threads per *block* to 512. And there are [*N*/512]+1 *blocks* run on the device, where N is not the multiple of 512. There are two types of blocks. Blocks whose indices are from 0 to *x*-1, that means the blocks from the first to the last but one, are of the same type. They are 2D blocks, threads in the block are in 2D grid, see in the Fig. 3. The last block is a special one, because the total number of threads is not the multiple of 512, which is the number of threads in the first type of block. So the rests are put into the special 1D block (0,*x*-1), threads here are assigned in an 1D queue, see in the Fig. 3.

We simplified the threads and blocks assignment, let *m* equal to 1, and *n* equal to 512. That means the first type of blocks is simplified to 1D array, too. So the management of threads and memory can be much easier, but it has no improvement to the efficiency of the algorithm, as shown in Fig. 4.

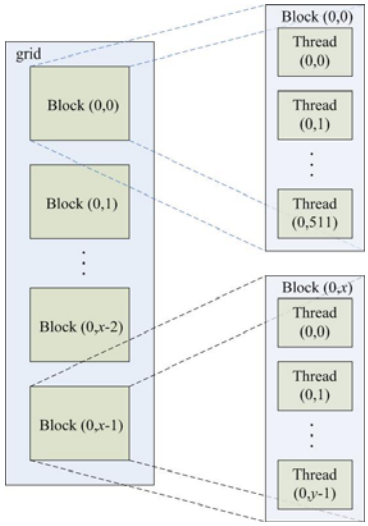

Figure 4. Simplified threads and blocks assignment. All blocks are treated as 1D array. And for blocks with index below *x*-1, all contain 512 threads. For block with index *x*-1, the threads number in it is *y*.

Data that shared by threads or keep using through steps should be treated carefully, such as the data describe the center of mass of four lumens and the position limit of particles. The former should be shared by threads; the latter should be used one at every time step and kept no change. Both of the two kinds of data are global existing, and shared by thread from different time step at last, so it is not wise to put them in the registers, though they are the fastest kind memory of lest amount. A large number of particles lead these data to be put into the global or the texture memory only. The share memory can only share the data between processors in the same multiprocessor, so it is not suitable, either, and global memory without any cache leads threads to wait in long access queue is also unsuitable. Finally, we choose the texture memory to save the latter and the constant

memory to save the former.

OpenGL is applied to visualize the result, and there are functions in CUDA to map the particles into the vertex array of GPU in the graphic memory that can also be accessed and modified by Vertex Buffer Object (VBO) technology, then GPU will draw them directly. And different parts of the heart have been marked by different colors as in Table 1.

III. Performance Evaluation

We have tested the solution by using CUDA Toolkit 2.0[8] and evaluated it on a workstation, having 1.866GHz Intel E6300 processor, 2GB system memory and one nVIDIA GeForce GTX 280 graphic card with 30 multiprocessors and 1GB graphic memory.

TABLE 2.
COMPARISON OF FPS (FRAME PER SECOND) AND SPEEDUP OF RUNNING PERFORMANCE ON A GPU-ACCELERATED COMPUTING AND GPU PIPELINE RENDER VERSION TO A CPU BASED COMPUTING AND SOFTWARE RENDER VERSION BASED ON TRADITIONAL OPENGL ARCHITECTURE.

Time	1	2	3
GPU-GPU	66.0	67.0	66.0
CPU-CPU	2.0	1.9	2.1
Acceleration	33.0x	35.3x	31.4x

TABLE 3.
COMPARISON OF FPS (FRAME PER SECOND) AND SPEEDUP OF RUNNING PERFORMANCE ON A GPU-ACCELERATED COMPUTING AND GPU PIPELINE RENDER VERSION TO A CPU BASED COMPUTING AND GPU PIPELINE RENDER VERSION.

Time	1	2	3
GPU-GPU	66.0	67.0	66.0
CPU-GPU	40.1	41.0	42.0
Acceleration	1.6x	1.6x	1.6x

From Table 2 we can see, our GPU implementation achieves speedups of almost 33 compared to the CPU based computing and software render version, and from Table 3 we can see, our GPU implementation achieves speedups of almost 1.6 compared to the CPU based computing and GPU render version. Matrix convert and other specific graphics computation cost too much time, and it is clear that throwing it to GPU can release much of CPU's loading. Because of the number of processors, the parallel processing performance of GPU is much higher than it of CPU. And with the CPU based visualization, it is a potential bottleneck to visualize particle positions since this requires a transfer of vertex attributes each frame. Visualization of the GPU pipeline based solution has the advantage that the surface-mesh is cached in video memory.

IV. Conclusion and Future Work

The results of this work show that the new CUDA compatible graphic cards are now advanced enough to be considered as efficient hardware accelerators for the cardiac activities simulation algorithm. All key components of our algorithms have been mapped onto the GPU for execution. The CUDA architecture can not only run as a high performance GPGPU platform but also cooperate with other graphic libraries to accelerate the animation displaying and computing physical effects.

In this work we just using a simplified physiologic model to simulate the cardiac muscle activities, with the GPU computing capability this simulation can be real-time implemented. In the future, our work will including the extension of the simulation algorithm, using more complex model to improve the simulation accuracy and validity.

Acknowledgment

The authors wish to thank Mr Chen Wenyu and Ms C Indhumathi for their helping in the advices to the implementation of the algorithm. This work was supported by the National 973 program of China (No. 2009CB320804) and Zhejiang Provincial Natural Science Foundation of China (Y1090597) and China Scholarship Council.

References

[1] J. Owens. March 2005. Streaming architectures and technology trends. GPU Gems 2, 457-470.
[2] Krüger, J., and Westermann, R. 2003. Linear algebra operators for gpu implementation of numerical algorithms, ACM Trans. Graph, 22, 908–916.
[3] Xu, F., and Müller, K. 2004. Ultra-fast 3d filtered backprojection on commodity graphics hardware. In, IEEE International Symposium on Biomedical Imaging 2004.
[4] Liu, W., Schmidt, B., Voss, G., and Müller-Wittig, W. 2007. Streaming algorithms for biological sequence alignment on gpus, IEEE Transactions on Parallel and Distributed Systems 18(9), 1270–1281.
[5] Horn, D., Houston, M., Hanrahan, P. 2005. Clawhmmer: a streaming hmmer-search implementation, In: Proceedings of Supercomputing 2005
[6] Moreland, K., and Angel, E. 2003. The fft on a gpu, In: Proceedings SIGGRAPH/Eurographics Workshop on Graphics Hardware, 112–119.
[7] Owens, J., Luebke,D., Govindaraju, N.,Harris, M., Kruger, J., Lefohn,A., and Purcell, T. 2005. A survey of general-purpose computation on graphics hardware, In: Eurographics 2005, 21–51.
[8] nVIDIA CUDA. [http://developer.NVidia.com/object/cuda.html]
[9] AMD Stream SDK. [http://ati.amd.com/technology/streamcomputing/sdkdwnld.html]
[10] Germann, W.J., and Stanfield, C.L. 2005. Principles of Human Physiology, Pearson Benjamin Cummings, London.
[11] Knudsen, Z., Holden, A., and Brindley, J. 1997. Qualitative modelling of mechano-electrical feedback in a ventricular cell. Bulletin of Mathematical Biology, 59, 6, 115–181.
[12] Nash, M. 1998. Mechanics and Material Properties of the Heart using an Anatomically Accurate Mathematical Model. PhD thesis, University of Auckland.
[13] Glass, L., Hunter, P., and McCulloch, A. (eds.). 1991. Theory of Heart: Biomechanics, Biophysics, and

Nonlinear Dynamics of Cardiac Function. Springer, Heidelberg.
[14] McCulloch, A., Bassingthwaighte, J., Hunter, P., and Noble, D. 1998. Computational biology of the heart: From structure to function. Progress in Biophysics and Molecular Biology 69, 153–155 .
[15] Sermesant, M., Delingette, H., and Ayache, N. 2006. An electromechanical model of the heart for image analysis and simulation. IEEE Transactions on Medical Imaging 25, 5, 612–625.
[16] Rongdong Yu, Patricia Chiang, Jianmin Zheng, Sanyuan Zhang, Yiyu Cai, Real-Time and Realistic Simulation for Cardiac Intervention with GPU , accepted by the 2nd International Conference on Computer modeling and simulation (ICCMS 2010).
[17] Rongdong Yu, Yiyu Cai, Jianmin Zheng, Wenyu Chen, Xiuzi Ye, Sanyuan Zhang, Yin Zhang, Patricia Chiang, Mon Hnin Tun, An Architecture of A VR Simulation System for Cardiac Intervention, the 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI 2008)
[18] Rongdong Yu, Patricia Chiang, Wenyu Chen, Jianmin Zheng, Yiyu Cai, Xiuzi Ye, Sanyuan Zhang, Yin Zhang and Koon-Hou Mak, A Framework for GPU-accelerated Virtual Cardiac Intervention, The International Journal of Virtual Reality, 2009, 8(1):37-41.
[19] http://users.ices.utexas.edu/~jessica/medical_data/heart/Heart_Valve_new.htm
[20] Y. Zhang, and C. Bajaj. 2004. Finite Element Meshing for Cardiac Analysis. ICES Technical Report 04-26, the Univ. of Texas at Austin.
[21] P. Hunter and B. Smaill. 1988. The analysis of cardiac function: a continuum approach, Progr. Biophys. Mol. Biol., 52, 101–164.
[22] J. Humphrey, R. Strumpf, and F. Yin. 1990. Determination of a constitutive relation for passive myocardium: I. A new functional form, ASME J. Biomechan. Eng., 112, 333–339.
[23] J. Guccione and A. McCulloch. 1991. Finite element modeling of ventricular mechanics, in Theory of Heart: Biomechanics, Biophysics, and Nonlinear Dynamics of Cardiac Function. Berlin, Germany: Springer-Verlag, 121–144.
[24] P. Hunter,M. Nash, and G. Sands. 1997. Computational electromechanics of the heart, in Computational Biology of the Heart. New York:Wiley, ch. 12, 345–407.
[25] J. Häfner, F. Sachse, C. Sansour, G. Seemann, and O. Dössel. 2002. Hyperelastic description of elastomechanic properties of the heart: a new material law and its application, Biomedizinische Technik, 47-1/2, 770–773.
[26] D. Caillerie, A. Mourad, and A. Raoult. 2002. Toward a fiber-based constitutive law for the myocardium, in Proc. Modeling & Simulation for Computer-Aided Medicine and Surgery (MS4CMS'02), 12, ESAIM Proceedings, 25–30.
[27] nVIDIA: nVIDIA CUDA Compute Unified Device Architecture-Programming Guide, (2008) http://developer.download.nvidia.com/compute/cuda

Rongdong Yu is a PhD student with Zhejiang University, China. Sponsored by the China Scholar Council, he is doing a collaborative research with Nanyang Technological University under an attachment project. His research interests include Bio-medical Science and Computer Sciences.

Dr. Yin Zhang, Associate Professor, she is with the College of Computer Science, Zhejiang University. Her research interests are mainly in artificial intelligence, geometric modeling and image processing.

Dr. Sanyuan Zhang is Professor with the College of Computer Science, Zhejiang University. His research interests are mainly in computer aided geometric design, and image processing.

Patricia Chiang receives her BEng degree from National University of Singapore and her MSc degree from Nanyang Technological University. Her research interests are in Signal Processing for Biomedical Engineering applications.

Dr. Yiyu Cai is associate professor with the School of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore. His research interests include VR, Computational Biomedical Sciences, and Interactive & Digital Media.

Dr. Jianmin Zheng is with the School of Computer Engineering, Nanyang Technological University. His research interest includes computer aided geometric design, CAD/CAM, computer graphics, animation, digital imaging and visualization.

Dr. Koon-Hou MAK is an interventional cardiologist and a visiting associate professor with the School of Mechanial and Aerospace Engineering, Nanyang Technological University, Singapore. His research interest is the development of medical devices, interventional cardiology and acute coronary syndromes.