2014

ISSN : 0974 - 7435 Volume 10 Issue 18

BioTechnology
An Indian Journal

FULL PAPER

BTAIJ, 10(18), 2014 [10405-10410]

Parallel implementation of position-based biologicalsoft tissue deformation

Li Hua-Ming*, Kang Bao-Sheng
College of Information Science and Technology, Northwest University,
Xi'an 710127, (CHINA)
E-mail : huaminglee@hotmail.com

ABSTRACT

With the development of the medical training systems and surgical simulation techniques, the improvement of deformable soft tissue simulation becomes highly demanded. Real-time performance of simulation algorithm is the most urgent bottleneck problem that needs to solve. This paper introduces a solution to implement deformable simulation of biological soft tissue in real time. This is accomplished through the use ofhierarchical position based dynamics, implemented usingCUDA parallel framework. The simulation results are directly sent to vertex buffer objectfor rendering to avoid the costly communication between CPU and GPU. The experimental results show significate improvements on performance in comparison to CPU algorithm.

KEYWORDS

Soft tissue; Deformable simulation; Position based dynamics; CUDA; Parallel algorithm.

INTRODUCTION

Surgical and medical trainingsimulation is ideally be used to provide a virtual environment of prototyping procedures as well as for research and development of novel procedures,the simulation of elastic materials characteristicof biological soft tissueis an important issue for creating visual effects, has attracted much research attention. Deformable body simulation was explored very early by Terzopoulos[1],since then, a lot of work has been published and related technologies are growing rapidly. Those researchmethods can be split into two main categories: mesh based methods and meshless methods.

One of the most popular mesh based methods to simulating deformable solids is the Finite Element Method (FEM). O'Brien[2] used FEM to simulate solidfracture with linear tetrahedral elements. Kaufmann[3] proposed to usingdiscontinuous Galerkin FEM support for arbitrary non-convex polyhedral elements allows for the efficient simulation of deformable object cutting. For thin deformable bodies, spring-mass technique is very popular with vessel and skinsimulation. Provot[4]was the first to use spring-mass network for simulating cloth in 1995. This technique is also used to convert any geometry into a soft body by using angular and linear spring[5]. Non-linear springs[6] can be used to capture cloth behavior more faithfully.
Contrasting, meshless methodis a fast technique to simulate deformable body. In 2006, Müller[7] introduced the position-based dynamics (PBD),the most typicalmeshless method, which omits the velocity and acceleration layer and immediately works on the positions, can be sufficient to create the desired deformable effects. This method can be used to simulate a variety of materials such as soft bodies,skin or even blood fluids by using different constraints. ThenMüller[8] presented a non-linear multigrid algorithm to speedup position based dynamics. In order to simulate various types of solid in same theory, amethod[9] has been presented that use oriented particles that store rotation and spin, along with the usual linear attributes, position and velocity.

No matter what kind of algorithm, deformation's computational complexity is always relatively large. How to build a simulator with the ability of real-time simulation is still an active area of research. A parallel computing approach is explored to satisfy real time constraints required by real time physics simulation. Since nowadays graphic processor unit (GPU) has evolved into an extremely powerful and flexible processor, while the compute unified device architecture (CUDA)[10] is specialized to compute intensive highly parallel computation. The goal of this paper is to designan algorithm for implementing ageneralposition based dynamics model[8] for deformable simulationon modern GPUs with the purpose of speeding the simulation up.

POSITION BASED DYNAMICS

Position-based dynamics became popular in the lastyears because they are fast, robust and controllable. It allows for imposing non-linear constraints of a geometric nature on a deformable surface, as in the case of volume preservation of the whole surface or of maintaining distance between two nodes of the mesh during deformation. This permits the modeling of the virtual structures without the use of internal or external forces, which simplifies the deformable model and produces unconditionally stable simulations, as a result of the elimination of the overshooting problem.

The particle system composed of the set of N particles and the set of Mconstraints represents the finest level 0 of the hierarchy. A geometric constraint can be expressed as$C_j(p_1, p_2, \dots, p_n) \geq 0$. During the simulation, given the currentspatial configurationp of the set of particles, we wantto find a correctionΔpsuch that$C(p + \Delta p) = 0$. Inorder to be efficiently solved, the constraint equation isapproximated by

$$C(p + \Delta p) \approx C(p) + \nabla_p C(p) \cdot k\Delta p = 0 \qquad \textbf{(1)}$$

Where $k \in \{0, \dots, 1\}$is the stiffness parameter. The solutionof the set of the resulting nonlinear constraints governingthe dynamic system is computed through an iterativeGauss–Siedel solver. The problem of the system being under-determined is solved by restricting Δp to be in the direction of$\nabla_p C$ which conserves the linear and angular momenta. This means that only one scalar λ (a Lagrange multiplier) has to be found suchthat the correction

$$\Delta p = \lambda \nabla_p C(p) \qquad \textbf{(2)}$$

solves Equation (1). This yields the following formula forthe correction vector of a single particle$i$

$$\Delta p_i = -s w_i \nabla_{p_i} C(p) \qquad \textbf{(3)}$$

Where

$$s = \frac{C(p)}{\sum_j w_j \left|\nabla_{p_j} C(p)\right|^2} \text{and} w_i = {}^{1}\!/\!{}_{m_i} \qquad \textbf{(4)}$$

In this context, stiffness k can be considered as the speed with which the particle positions converge towards a legal spatial state, that is, a state in which all the constraints are satisfied. By tuning the value of k, It can be controlled how much a constraint is stringent during the evolution of the simulation. For example, a distance constraint between two particles withk = 0.5 behaves similar to a spring, whereas withk = 1.0 behaves nearly like a stiff solid.

PARALLELALGORITHM

During the simulation, deformation is computed by comparing the current deformed configuration of point samples with their reference configuration. Workflow of the entire algorithm can be described asfollows: The 3D model of the object to be simulated is firstdiscretized as a set of discrete particlepoint. When the object isaffected byforce, the deformation will be activated andthe system enters into thesimulation stage. After the simulation of each time step, those particle pointsare updated and directly sent to the GPU for rendering. To take advantage of CUDA, algorithm has been considering as many simple particle interactions in parallel, over more advanced algorithms that run serially. Implementationof the algorithm of each time stepcan be described asfollows:

[1] Compute per particle deformation gradient, apply forces and predict position
[2] Compute per particleneighbors, find and save contacts.
[3] Use one (or more) PBD solver steps on the current particle state. Solve all the constraints of this level using non-linear Gauss-Seidel.
[4] Update the position of each particle
[5] Read data calculated by CUDA from VBO
[6] Render the data in VBO directly on GPU.

For this paper, we haveconsidered the elastic forcewhile also accelerating the K nearest neighbor (KNN) calculations and incorporating OpenGL to render the results as they are calculated. The primary goalis to illustrate the potential acceleration that could be achieved by incorporating CUDA parallelization into the simulation.

CUDA IMPLEMENTATION

The algorithm we designed is inherently parallel, and easy to implement the speedups by performing operations on the Cuda. Most steps of the algorithm only involve computingmatrices based on local information, or information from neighbor particles, very little synchronization is required. Each iteration requires a matrix multiply and a few dot products. These operations don’t perform particularly well on a GPU, but easily outperform CPU implementations. Rough profiling of the sequential implementation revealed that these solves take the majority of the computation time, in some cases as much as 90%. Hence any speedup by performing these computations on the GPU will make a significant difference in overall runtime. Additionally, the matrices do not need to be stored explicitly; the matrix vector multiplication operation can be implemented as a loop over particles and their neighbors, resulting in predictable memory accesses and little synchronization.

The simulation on CUDA usesa uniform grid data structure presented by Simon Green[11]. The calculated results are written to a Vertex Buffer Object (VBO) and are directly rendered on the GPU, no copying is necessary between CPU and GPU except at the first frame of simulation. To solve the K Nearest Neighbors using the GPU we use a CUDA implementationpresented by Garcia[12] that provides speedup over sequential implementations.

In theimplementation, we chose to map each particle to a CUDA thread, using a block size of 64 threads to allow 2 warps per block with plenty of registers for each thread to use. There are several points in the algorithm that require global synchronization. Figure 1 illustrates the control flow of the whole algorithm. Additionally, atomic addition is required to avoid race conditions when calculating the forces between all neighboring particles.

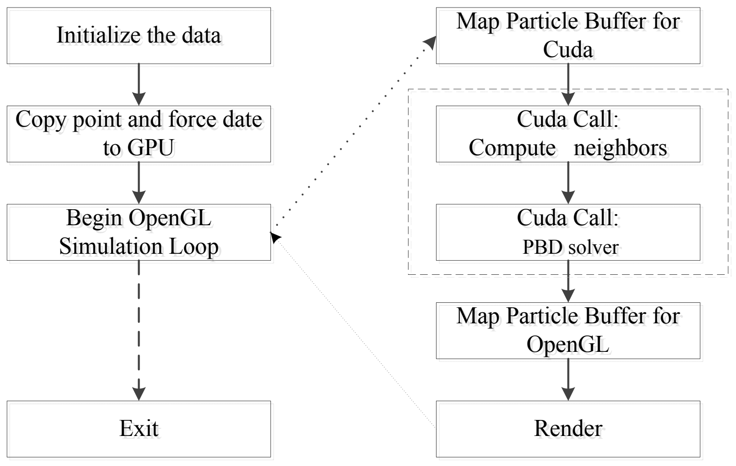
Initialize the data
Copy point and force date to GPU
Begin OpenGL Simulation Loop
Exit
Map Particle Buffer for Cuda
Cuda Call: Compute neighbors
Cuda Call: PBD solver
Map Particle Buffer for OpenGL
Render

Figure 1 : Overview of control flow between OpenGL and CUDA

A major advantage of using CUDA for this simulation is that results can be rendered efficiently as they are calculated, since the data is already stored on the GPU. We displayed the results by swapping a data buffer back and forth between OpenGL and CUDA. CUDA includes functions specifically for this type of interaction. The buffer is initially created as an OpenGL VBO, but during CUDA kernel calls, it is unmapped from OpenGL while its data is manipulated by the CUDA code. At the completion of the kernel function, the buffer is mapped back to OpenGL where it is rendered. This buffer contains the position data for the particles in the simulation as 4-tuple float values, corresponding to the X, Y, Z, andWhomogenous coordinates. This is represented using a float4 data type in CUDA.

Incorporating OpenGL into a CUDA program introduces a fair amount of additional overhead and has a significant impact on the overall structure and organization of the code. Once all the initialization has been taken care of for both CUDA and OpenGL, the program enters into the main animation loop for OpenGL. While in this loop, it uses several callback functions to handle input and output. We made the CUDA kernel calls from within the "display"callback function. This way, the display is updated as soon as new results are calculated. The downside of this organization is that as the kernel execution time increases for larger simulations, it causes the display function to take longer, which results in a sluggish program response to user input.

Currently, the calculation of nearest neighbors among particles is the most expensive operation in the simulation. So we designed that nearest neighbors are calculated for every 10th frame, and those results are reused until the next time they are updated. Even so, the KNN calculations tend to have the largest impact on overall execution time. For example, for 512 particles and 32 neighbors, a typical frame with no KNN calculations took around 5ms to update while those frames updating the neighborhoods tended to take over 100ms. For a larger simulation, with 32768 particles and 32 neighbors, KNN calculations took over 2 full seconds while frames between neighborhood updates required less than 25ms.

So we used a CUDA implementation for calculating K nearest neighbors. It basically uses CUDA to parallelize the brute force approach for calculating KNN, make use of texture memory for non-coalesced memory access during the calculations, and use an insertion sort to order distances. This implementation of KNN was definitely faster than the brute force method of comparing all points.

RESULTS

We have integrated the algorithm into a real time physics simulation system. It is tested our simulator through liver model occurs deformation influenced by external forces. Selected frames are shown in Figure 2.

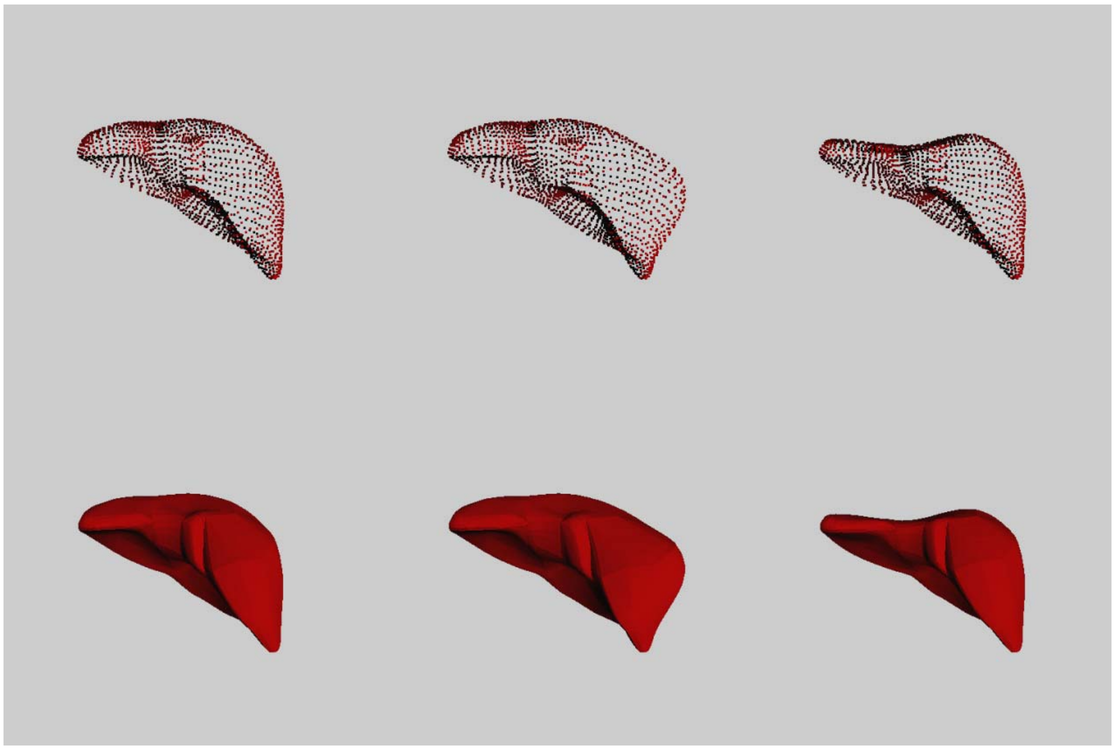

Figure 2 : Example simulation of liver model occurs deformation influenced by external forces

We compared our results to a CPU implementation in order to illustrate the speedup of our parallel algorithm. The testing is performed on a PC using Window 7 System with Intel Xeon X3320 CPU, 8.0GB system memory and NVIDIA GeForce GTX560Ti (1.0GB video memory and 384 core). Figure 3 and TABLE 1 show how performance of our method deals different soft tissue model with the number of particles, and the number of neighbors per particle.

TABLE 1 : Runtimes per timestep (ms)

Number of neighbors	Number of particles			
	512	4096	13824	37982
32 (CPU)	25	36	1374	4910
32 (GPU)	22	27	87	263
64 (GPU)	34	61	209	541

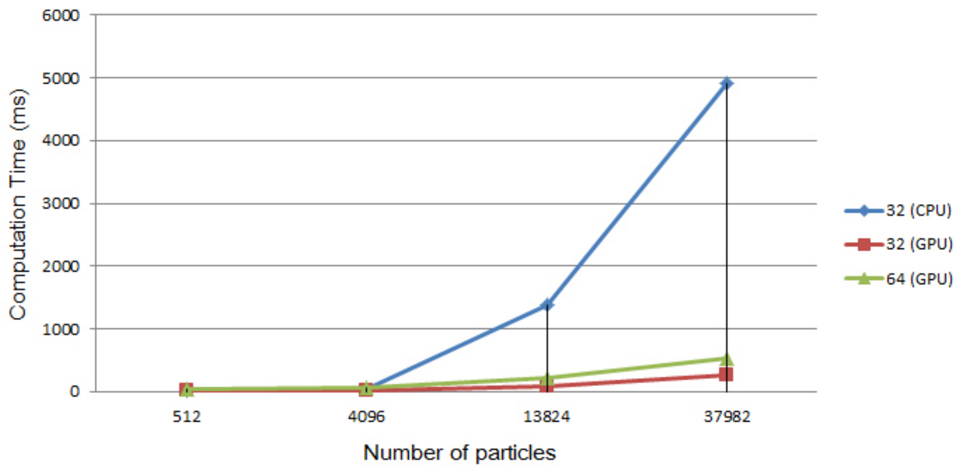

Figure 3 : The performance of testing with the different model (different particle number), and the different number of neighbors per particle

The performance for small numbers of particles is comparable between the sequential and parallel implementations, with only a slight speedup for the parallel version. With a large number of particles, the CUDA implementation is significantly faster than the sequential implementation. With 37,982 particles and 32 nearest neighbors, the CUDA implementation was 20 times faster than the sequential version. It also shows that the KNN queries are a significant bottleneck in the implementation.

CONCLUSIONS

This paperpresents a parallel CUDA algorithm for the implementation ofdeformable soft tissuesimulation based on general hierarchical position based dynamics. Then we discuss the wholecontrol flow between the CPU and GPU. Through soft tissuemodel discretized, particle interactions in parallel, the use of a uniform grid data structure, finally, algorithm has been achieved both include PBD solver andneighborrefactoring. As expected, the CUDA implementation provided significant speedup over the sequential implementation of the simulation, especially for large problem sizes. But the KNN algorithm used in parallel algorithm performed poorly. Investigating this further is a direction of future work. Through expanding PBD constraints, this algorithm can be quickly applied to other deformable areas, such as the simulation of skin, hair, and muscle.

REFERENCES

[1] Terzopoulos Demetri, Platt John, Barr Alan, et al; Elastically deformable models. Proceedings of the 14th annual conference on Computer graphics and interactive techniques, 205-214 **(1987)**.

[2] F.O'Brien James, K.Hodgins Jessica; Graphical modeling and animation of brittle fracture. Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 137-146 **(1999)**.

[3] Kaufmann Peter, Martin Sebastian, Botsch Mario, et al; Flexible simulation of deformable models using discontinuous Galerkin FEM. Graph Models, **71(4)**, 153-167 **(2009)**.

[4] Provot Xavier; Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. Graphics Interface, 147-154 **(1996)**.

[5] A.Joukhadar, C.Laugier; Fast dynamic simulation of rigid and deformable objects. Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on, 305-310 **(1995)**.

[6] Volino Pascal, Magnenat-Thalmann Nadia, Faure Francois; A simple approach to nonlinear tensile stiffness for accurate cloth simulation. ACM Transactions on Graphics, **28(4)**, 1-16 **(2009)**.

[7] Müller Matthias, Heidelberger Bruno, Hennix Marcus, et al; Position based dynamics. Journal of Visual Communication and Image Representation, **18(2)**, 109-118 **(2006)**.

[8] Müller Matthias; Hierarchical Position Based Dynamics. Proceedings of Virtual Reality Interactions and Physical Simulations, 13-24 **(2008)**.

[9] Müller Matthias, Chentanez Nuttapong; Solid simulation with oriented particles. ACM SIGGRAPH, 1-10 **(2011)**.

[10] Nickolls John, Buck Ian, Garland Michael, et al; Scalable Parallel Programming with CUDA. Queue, **6(2)**, 40-53 **(2008)**.

[11] Green Simon; Cuda particles. **(2008)**.

[12] V.Garcia, E.Debreuve, M.Barlaud; Fast k nearest neighbor search using GPU. Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on, 1-6, **(2008)**.