Real-time physically cloth simulation with CUDA

Li Huaming*, Kang Baosheng

College of Information Science and Technology, Northwest University, Xi'an 710127, China

Received 1 September 2014, www.cmnt.lv

Abstract

With the development of the simulation technique, deformable cloth simulation has become highly desired. It can be widely used in many fields such as game, animation, virtual surgery, etc. Real-time algorithm is the most urgent bottleneck problem that needs to be solved. This paper introduces a solution to implement deformable simulation of cloth in real time, accomplished through using a meshless simulation technique, which is known as position-based dynamics, implemented using CUDA parallel framework. The simulation results are directly sent to vertex buffer object for rendering to avoid the costly communication between CPU and GPU. The experimental results show significate improvements on performance in comparison to CPU algorithm.

Keywords: Position-based Dynamics, Cloth Simulation, CUDA, Parallel Algorithm, Uniform Grid

1 Introduction

As the graphics industry continues to grow, cloth simulation continues to be an important issue for creating visual effects. It can be applied in many fields such as visual simulation, games, virtual surgery, safety training, etc. Deformable body simulation was introduced to graphics very early by Terzopoulos [1], since then, a lot of work has been published and related technologies are growing rapidly. Physical-based methods are the main direction of the majority research in this field. Those methods 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 solid fracture with linear tetrahedral elements. Kaufmann [3] proposed to using discontinuous 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 cloth simulation. 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 behaviour more faithfully.

Contrasting, the most popular meshless method is position-based dynamics (PBD) introduced by Müller [7] in 2007, 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, cloth or even fluids by using different constraints. Then Müller [8] presented a non-linear multigrid algorithm to speedup position based dynamics. In order to add wrinkles to simulated cloth, a method [9] has been presented that attach a higher resolution wrinkle mesh to the coarse base mesh allowing the wrinkle vertices to deviate from their attachment positions within a limited range. More recently, Kim [10] proposed an approach that applies unilateral distance constraint between particles to distant attachment point, preventing elastic over deformation of cloth.

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) [11] is specialized to compute intensive highly parallel computation. The goal of this paper is to design an algorithm for implementing a general position based dynamics model [7] for cloth simulation on modern GPUs with the purpose of speeding the simulation up.

2 Methodology

Position-based dynamics became popular in the last years 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 modelling 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.

2.1 PHYSICAL MODEL OF CLOTH

The cloth to be simulated is represented by a set of N particles and the set of M constraints. Those particles are

**Corresponding author* e-mail: huaminglee@hotmail.com

large collection of masses connected by spring constraints. So the mass-spring model is converted into a system of constraints and this model can be solved sequentially and iteratively, by directly moving nodes to satisfy each constraint, until sufficiently stiff cloth is obtained. Figure 1 shows three different types of constraints of our cloth physical model.

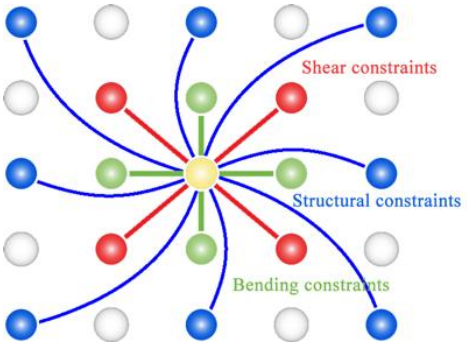

FIGURE 1 Different types of constraints of physical model.

2.2 THE SOLVER

The solver function projects the positions of vertices in the direction of the constraints to satisfy the constraints. The algorithm uses the idea of a Gauss-Seidel-type iteration in that it solves each constraint independently. The solver is repeatedly called through all the constraints to project the particles to valid locations with respect to the given constraint alone.

2.3 CONSTRAINTS PROJECTION

A constraint can be expressed as $C_j(p_1,p_2,\ldots,p_n) \geq 0$. During the simulation, given the current spatial configuration p of the set of particles, we want to find a correction Δp such that $C(p + \Delta p) = 0$. The constraint equation is approximated by

$$C(p + \Delta p) \approx C(p) + \nabla_p C(p) \bullet k\Delta p = 0, \quad (1)$$

where $k \in \{0,\ldots,1\}$ is the stiffness parameter. 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 such that the correction

$$\Delta p = \lambda \nabla_p C(p) \quad (2)$$

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

$$\Delta p_i = -sw_i \nabla_{p_i} C(p), \quad (3)$$

where

$$s = \frac{C(p)}{\sum_j w_j \left|\nabla_{p_j} C(p)\right|^2} \quad (4)$$

and

$$w_i = 1/m_i. \quad (5)$$

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 with $k = 0.5$ behaves similar to a spring, whereas with $k = 1.0$ behaves nearly like a stiff solid.

3 Parallel Algorithm

During the simulation, deformation is computed by comparing the current deformed configuration of point samples with their reference configuration. Algorithm consists of two parts: the simulation and rendering. Workflow of the entire algorithm can be described as follows: The 3D model of the object to be simulated is first discretized as a set of discrete particle point. When the object is affected by force, the deformation will be activated and the system enters into the simulation stage. After the simulation of each time step, those particle points are 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. Implementation of the algorithm of each time step can be described as follows:

1) Compute per particle deformation gradient, apply forces and predict position
2) Compute per particle, update and save the uniform grid structure.
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 have only considered the elastic force and the simplest possible spatial subdivision structure. The primary goal is to illustrate the potential acceleration that could be achieved by incorporating CUDA parallelization into the simulation.

4 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 computing matrices 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 do not 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 uses a uniform grid data structure as an acceleration structure in collision detection presented which distance values of particles are computed for each grid point. The major advantages of the uniform grid are the simplicity to implement them and the constant time necessary for distance queries at any of the grid points. However, uniform grids have drawbacks, resulting from their constant sampling rate, the grid data structure is generated from scratch each time step. We detect collisions between particles using the discrete element method (DEM) method, and collision model consists of several forces, including a spring force, which forces the particles apart. The method for update the uniform grid is that using GPU atomic operations, which allow multiple threads to update the same value in global memory simultaneously without conflicts. 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.

In the implementation, we chose to map each particle to a CUDA thread, using a block size of 32 threads. Each particle calculates which grid cell it is in. It then loops over the neighboring 27 grid cells (3x3x3 cells) and checks for collisions with each of the particles in these cells. If there is a collision the particle's velocity is modified. There are several points in the algorithm that require global synchronization. Figure 2 illustrates the control flow of the whole algorithm. Additionally, atomic addition is required to avoid race conditions when calculating the forces between all particles.

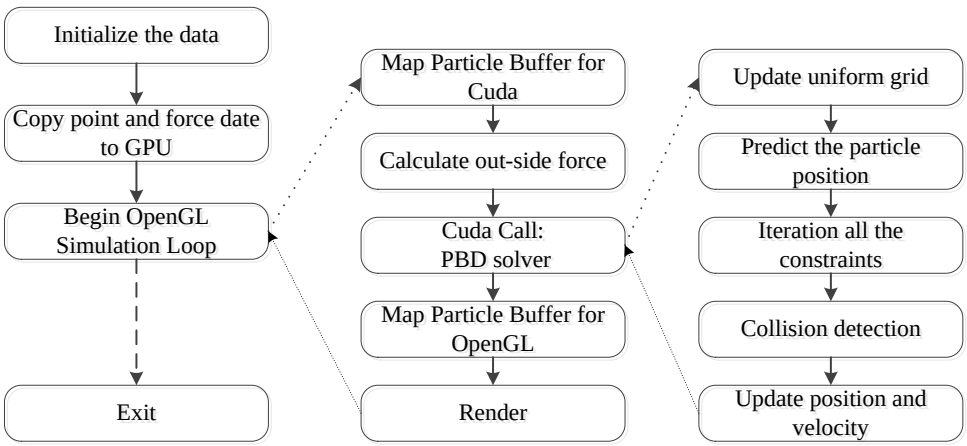

FIGURE 2 Overview of control flow between OpenGL and CUDA.

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, and W homogenous 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.

5 Results

We have integrated the algorithm into a real time physics simulation system. It is tested our simulator by dropping a piece of cloth fell on a sphere. Selected frames are shown in Figure 3.

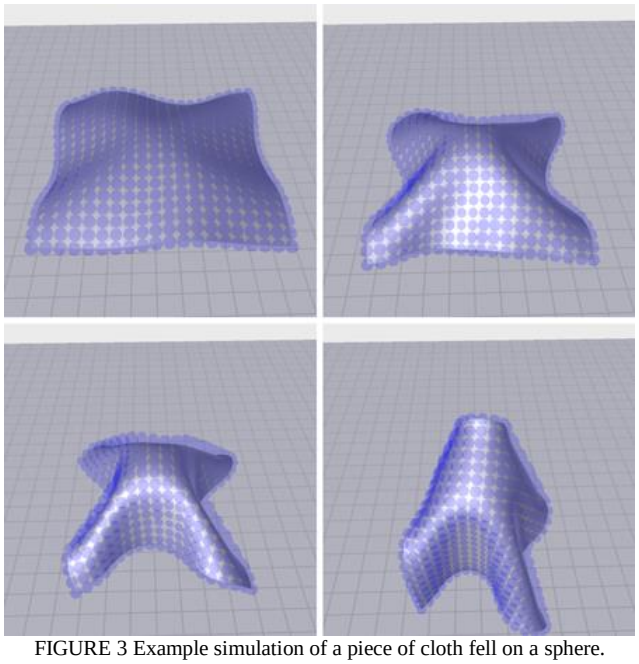

FIGURE 3 Example simulation of a piece of cloth fell on a sphere.

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, and NVIDIA GeForce GTX560Ti (1.0GB video memory and 384 core). Figure 4 and Table 1 show how our method deals with the number of particles.

TABLE 1 Runtimes per timestep (ms)

Number of particles	512	4096	16384	32768
CPU	25	36	1374	4910
GPU	34	61	209	541
Update Grid Average Time (GPU)	14	21	72	242

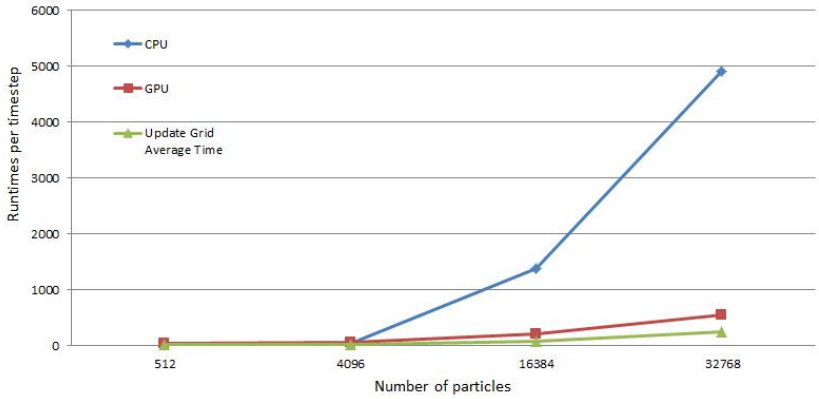

FIGURE 4 The performance of testing with different particle number

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 32,768 particles, the CUDA implementation was 20 times faster than the sequential version. It also shows that the update of uniform grid is a significant bottleneck in our implementation. The atomic-based algorithm is generally slower and we can use sorting method to improve this phenomenon.

6 Conclusions

This paper presents a parallel CUDA algorithm for the implementation of cloth simulation based on general position based dynamics. Then we discuss the whole control flow between the CPU and GPU. Through solid model discretized, particle interactions in parallel, the use of a uniform grid data structure, finally, algorithm has been achieved both include PBD solver and DEM. As expected, the CUDA implementation provided significant speedup over the sequential implementation of the simulation, especially for large problem sizes. But the uniform grid update 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 biological soft tissue, elastic-plastic solid, and fluids.

References

[1] Terzopoulos D, Platt J, Barr A 1987 *the 14th annual conference on Computer graphics and interactive techniques* ACM Press: New York 205-14

[2] O'Brien J F, Hodgins J K 1999 *the 26th annual conference on Computer graphics and interactive techniques* ACM Press: New York 137-46

[3] *Deleted by CMNT Editor*

[4] Provot X 1996 Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior *Graphics Interface* 147-54

[5] Joukhadar A and Laugier C 1995 *the IEEE/RSJ International Conference on Intelligent Robots and Systems* IEEE Press: Pittsburgh 305-10

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

[7] Müller M, Heidelberger B and Hennix M 2006 Position based dynamics *Journal of Visual Communication and Image Representation* **18**(2) 109-18

[8] Müller M 2008 Hierarchical Position Based Dynamics *Virtual Reality Interactions and Physical Simulations* Eurographics Association: Grenoble 1-10

[9] *Deleted by CMNT Editor*

[10] *Deleted by CMNT Editor*

[11] Nickolls J, Buck I, Garland M 2008 Scalable Parallel Programming with CUDA *Queue* **6**(2) 40-53

Authors

Li Huaming, 1977.06, Xi'an County, Shaanxi Province, P.R. China

Current position, grades: PhD student, College of Information Science and Technology, Northwest University, China
University studies: B.Sc. in Computer Science from Northwest University in China. He received his M.Sc. from Northwest University in China.
Scientific interest: His research interest fields include Computer Graphics, Virtual Reality, GPU Parallel Computing
Publications: 5 papers published in various journals.
Experience: He has completed one scientific research projects.

Kang Baosheng, 1961.10, Xi'an County, Shaanxi Province, P.R. China

Current position, grades: Professor of College of Information Science and Technology, Northwest University, China.
University studies: received his B.Sc. in Mathematics from Northwest University in China. He received his M.Sc. from Northwest University in China.
Scientific interest: His research interest fields include Computer Graphics, Computer Aided Geometric Design, Multimedia Technology
Publications: more than 50 papers published in various journals.
Experience: He has teaching experience of 29 years, has completed more than ten scientific research projects.