Cloth Simulation through Position Based Dynamics using CUDA
CS 432: GPU Accelerated Computing
Project Research Paper

Muhammad Hammad Maqdoom

DSSE, Habib University, PK

Lama Imam

DSSE, Habib University, PK

Abstract The growing availability of parallel computing architectures has prompted research into the best algorithms and approaches for maximising parallelism. The GPU has evolved as a parallel processing device in addition to the parallelism offered in the CPU. The purpose of this research was to construct a hybrid parallel CPU/GPU cloth simulation application to investigate the joint utilisation of CPU and GPU parallelism. To test the merits of the hybrid approach, the application was first designed in sequential CPU form, then in parallel CPU form.The application uses a position-based dynamic approach, accumulating internal and external forces before computing accelerations using Newton's second law of motion. The velocities and, eventually, the locations of the object are updated using a time integration method. Only a few simulation approaches (most rigid body simulators) use impulse-based dynamics to manipulate velocities directly.

In this project we introduce an OpenGL based cloth simulation code base, in which we implemented a GPU version of the already existing CPU version of the Position Based Dynamic cloth simulation algorithm. We produced this simulation with high resolution on the GPU and have plotted the significant difference in runtimes for both approaches. We used the opencloth repository on github to access source code, where we made use of the libraries existing in the repository to implement a GPU version of the Position Based Dynamic cloth simulation approach.

The findings of this work could lead to future research into using GPUs to execute time integration methods connected with updating velocities in the position based dynamics algorithm in more sophisticated cloth simulation settings.

Introduction Cloth simulations can now be found in a wide range of settings. The existence of virtual clothing and cloth elements links a wide range of visual output, from Blockbuster films to garment prototype tools. Researchers have been working on algorithms and methods to display cloth elements convincingly using computers for over 20 years. From simple geometric approximations to complicated physical systems, these techniques are used[1].

The complexity of the cloth simulation, as well as the time constraints imposed on processing it, is dictated by whether it will be used as part of an interactive system or as part of a pre-rendered animation. Pre-rendered animations, such as those used in Films, are subjected to significantly fewer time constraints than interactive systems, like video games. The amount of time it takes to make a pre-rendered animation is immaterial to the animation's viewer, as they have no idea how long it took to develop the sequence. The length of time it takes to develop a sequence is only important to the animation's producer if it has an impact on the final animation's cost.The interactive scenario is quite unique. The animation's speed affects the user's experience. The ideal animation rate for an interactive system is 30 frames per second (frames per second). Animated objects move in a fragmented manner if the frame rate is too low, missing the fluidity of natural motion. In the pre-rendered world, minutes and even hours are acceptable times for the production of a single frame. This, however, is not acceptable when it comes to producing animations with multiple frames per second.

Interactive systems research is motivated by a desire to push simulation complexity to greater levels while retaining interactive frame rates. Such systems have extraordinarily high performance requirements, and implementing them remains a key issue for cloth simulation researchers. The approaches utilised in this study are more suited to pre-rendered animation.

Cloth simulation computing hardware There are three steps to cloth simulation. The cloth abstraction model is implemented in the **first stage**. This step is largely concerned with the model's data types and populating those data types with the simulation's initial conditions. The numerical solver is implemented in **stage two**, which will calculate the placements of the simulation's components by solving equations of motion within the application. Collision detection and reaction are implemented in **stage three**. The mechanism through which the simulation identifies whether more than one element of the simulation is

occupying the same point in space is called collision detection. The simulation's collision response is how it handles identified collisions. Parallelism can be used at each of these levels[1].

Problem Statement To keep improving cloth simulation performance, hardware architectures must make use of parallelism. Greater rates of parallelism will be responsible for future hardware performance advancements. Not only should cloth simulation algorithms be built to make use of today's parallelism, but they should also be able to scale to tomorrow's parallelism. Researchers will continue to investigate the extent to which cloth simulation techniques can accomplish this. The question that arises is, how can cloth simulation and generation performance be improved for future technological advancements. This study focuses on the implementation of the position based dynamic algorithm using a GPU for faster cloth generation, and proposes a potential solution that can be built upon by researchers to fine grain cloth simulation in a multitude of applications such as video games.

Purpose of our Project The purpose of this research is to see if a cloth simulation approach that uses both CPU and GPU parallelism may deliver higher performance than a similar algorithm that uses serial and parallel CPU implementations.

Hypothesis In theory, serial implementation will give us the worst runtimes and give us the least efficient solution. Introducing parallelism will obviously increase our systems performance. When it comes to CPU parallelism vs GPU parallelism, we hypothesise that the GPU implementation will yield better results because of the availability of shared memory space and due to its parallel processing architecture, with which it can run various calculations over multiple data streams at the same time.

Literature Review

Müller, in his paper describes a hierarchical, position-based technique for cloth simulation by accelerating the solver's convergence speed[2]. Bender and Bayer used a red-black parallel Gauss–Seidel schema for animating inextensible cloth with a force-based approach in their research[3]. This approach is limited to meshes with a regular grid structure, despite its high performance. The mesh is separated into constraint strips. Strips with no common particles are independent of one another and can be solved simultaneously. Fratarcangeli [4] in their paper, describes a comparable strategy that relies on a Jacobi-like solver. All of these solvers work well with grid-like meshes (cloths), but they lack the generality required for simulation of elements varying structures.

Jacobi and Gauss–Seidel iterative solvers are commonly employed in interactive computer animation and simulation applications because they produce faster results than direct approaches, despite the fact that the resulting solutions have a higher residual error. For touch response, they've been used by multiple researchers such as Bridson[5] in their paper 'Robust Treatment of Collisions, Contact and Friction for Cloth Animation', Harada[6] in '"A Parallel Constraint Solver for a Rigid Body Simulation' and Tonge[7] in 'Mass Splitting for Jitter-free Parallel Rigid Body Simulation'. More recently, Macklin in his research proposed a unified architecture for rigid bodies, soft bodies, and fluids, in which animated entities are voxelized using constant sized spheres and constraints are addressed using a Jacobi-like solver [8].

To get an understanding of the math behind the position based dynamic algorithm as well as implement a GPU version of the pre-existing CPU solution using graph colouring approach, we referred to the study by Marco Fratarcangeli[9]

Methodology

Position Based Dynamics Position Based Dynamics (PBD) is a method for interactively animating deformable objects based on Verlet integration[10]. The objects are represented as a collection of n particles whose movement is controlled by m nonlinear constraints. By directly updating the particle placements, the system of equations is solved using Gauss–Seidel iterations. Internal forces are avoided in PBD, and the locations are updated in such a way that the angular and linear momentums are conserved implicitly. [9] As a result, the entire process is unaffected by the instabilities associated with interactive, physically based approaches. Nonlinear equality and inequality equations make up the set of constraints, as follows[9]:

$$C_i(p) \geq 0 | C_i(p) = 0 \text{ where i = 1, ... , m}$$

above, p is the vector of particle positions, n is the number of particles, and m is the number of constraints. The constraints are mostly nonlinear, and they're solved in order using Gauss–Seidel iterations. To get the correction Δp, each equation is linearized individually in the vicinity of C around the current configuration p. This is represented by the following equation[9]:

$$C_i(p + \Delta p) \approx C_i(p) + \nabla p C_i(p) \cdot (p) = 0$$

$\nabla p C_i(p)$ is a vector that contains the derivatives of the equation C_i with respect to the n components of p. p is imposed to be in the direction of $\nabla p C(p)$ such that:

$\Delta p = \lambda i \cdot \nabla p \cdot C_i(p)$

This condition conserves the linear and angular momenta implicitly while allowing the underdetermined system of constraints to be solved. Combining the above 2 equations and making λi the subject gives us [9]:

$$\lambda i = \frac{C_i \cdot p}{|\nabla p \cdot C_i \cdot p|^2}$$

Stretch Constraint One stretch constraint for the particles (p_x, p_y) at the end points of each edge of the input mesh, is defined as $C(p_x, p_y) = |(p_x, p_y)| - d = 0$ which includes the edges of the internal tetrahedrons. This is used to keep particles p_x and p_y at distance d, i.e. the length of the edge. Given the configuration (p_x, p_y) of two particles connected by a stretch constraint, the corrections to the positions to satisfy the constraint are:

$$\Delta p_x = -\frac{1}{2} * k_s (|p_x - p_y| - d)(p_x - p_y / |p_x - p_y|)$$

$$\Delta p_y = +\frac{1}{2} * k_s (|p_x - p_y| - d)(p_x - p_y |p_x - p_y|)$$

where $k_s \in (0, 1]$ is a stiffness scalar parameter, which slows the convergence of the constraint and provides a "springy" behaviour to the corresponding edge. [9]

Tetrahedral Volume Constraint At the corners of each tetrahedral of the mesh, we define one volume constraint for the particles (p_1, p_2, p_3, p_4). The volume constraint enforces mesh volume conservation by maintaining the rest volume of four particles creating a tetrahedron[9]:

$$C_{p1,p2,p3,p4} = \frac{1}{6}((\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \cdot \mathbf{p}_{4,1} - V_0).$$

where $p_{i,j}$ represents $p_i p_j$ and V_0 denotes the rest volume of the tetrahedral. The gradient with respect to each particle is shown by:

$$\nabla_{\mathbf{p}_2} C(\mathbf{p}_{2,3}) = \frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}), \quad (8)$$

$$\nabla_{\mathbf{p}_3} C(\mathbf{p}_{3,4}) = \frac{1}{6}(\mathbf{p}_{3,1} \times \mathbf{p}_{4,1}), \quad (9)$$

$$\nabla_{\mathbf{p}_4} C(\mathbf{p}_{4,2}) = \frac{1}{6}(\mathbf{p}_{4,1} \times \mathbf{p}_{2,1}), \quad (10)$$

$$\begin{aligned}\nabla_{\mathbf{p}_1} C(\mathbf{p}_1) &= -(\nabla_{\mathbf{p}_2} C(\mathbf{p}_{2,3}) + \nabla_{\mathbf{p}_3} C(\mathbf{p}_{3,4}) \\ &\quad + \nabla_{\mathbf{p}_4} C(\mathbf{p}_{4,2})) \\ &= -\frac{1}{6}((\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) + (\mathbf{p}_{3,1} \times \mathbf{p}_{4,1}) \\ &\quad + (\mathbf{p}_{4,1} \times \mathbf{p}_{2,1})). \end{aligned} \quad (11)$$

The correction for each particle in the tetrahedron is:

$$\Delta \mathbf{p}_1 = -\frac{1}{6} \cdot s \cdot k_v \cdot (\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) + (\mathbf{p}_{3,1} \times \mathbf{p}_{4,1}) + (\mathbf{p}_{4,1} \times \mathbf{p}_{2,1})$$

$$\Delta \mathbf{p}_2 = \frac{1}{6} \cdot s \cdot k_v \cdot (\mathbf{p}_{2,1} \times \mathbf{p}_{3,1})$$

$$\Delta \mathbf{p}_3 = \frac{1}{6} \cdot s \cdot k_v \cdot (\mathbf{p}_{3,1} \times \mathbf{p}_{4,1})$$

$$\Delta \mathbf{p}_4 = \frac{1}{6} \cdot s \cdot k_v \cdot (\mathbf{p}_{4,1} \times \mathbf{p}_{2,1})$$

where k_v denotes the stiffness parameter and s denotes the scaling factor. [9]

```
(1) forall particles i
(2)     initialize xi = x 0 i ,vi = v 0 i ,wi = 1/mi
(3) endfor
(4) loop
(5)     forall particles i do vi ← vi +Δtwifext(xi)
(6)     forall particles i do pi ← xi +Δtvi
(7)     forall particles i do generateCollisionConstraints(xi → pi)
(8)     loop solverIterations times
(9)         projectConstraints(C1,...,CM+Mcoll ,p1,...,pN)
(10)    endloop
(11)    forall particles i
(12)        vi ← (pi −xi)/Δt
(13)        xi ← pi
(14)    endfor
(15) endloop
```

Algorithm

Algorithm sourced from [11].

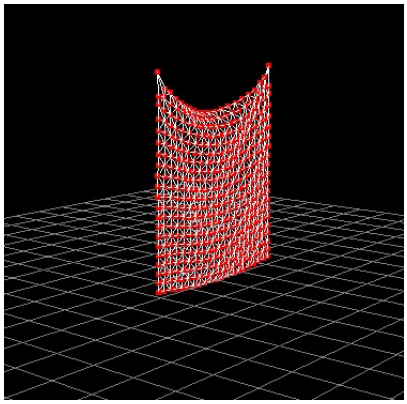
GPU Implementation It's vital to reduce the amount of data that travels through the main memory during the running of the algorithm by minimising CPU communication/use by running the algorithm in parallel on GPU. One of the key bottlenecks that severely penalises time performance is the time spent transferring data.

We designed our system to keep the data on the GPU as much as possible. In the initialization phase, we load all the data required for the OpenGL Rendering on the video memory. Then, during the object, constraints and calculations phase, we update the data structures directly on the GPU. In this way, the CPU is not involved in the process (besides being responsible for calling the GPU kernels), thus using this technique we were able to optimize the work opencloth in CUDA. Then we rendered the results using OpenGL

Results and Analysis

We have implemented our technique using CUDA and FPS for the scene was

measured and compared. We also compare with both the standard PBD implemented in C++ and running on a single core on an AMD Ryzen 7 PRO 1700 with Eight Cores, and with our technique implemented in CUDA on the NVIDIA GTX 1650 SUPER GPU .



We test the iteration count independent nature of our algorithm by simulating a hanging cloth subject to gravity. The cloth model consists of a grid of particles connected by a graph of distance constraints.

Runtimes and Comparison Here are the results we obtained by varying our iterations:

- **Number of solver iterations per step.**
  - **Test case 1**
    PBD = 2 ,
    CPU FPS = 116 ,
    GPU FPS = 2293
  - **Test case 2:**
    PBD = 20 ,
    CPU FPS = 41 ,
    GPU FPS = 810.5
  - **Test case 3:**
    PBD = 100 ,
    CPU FPS = 11 ,
    GPU FPS = 217.5

Conclusions We have presented a simple rework of position-based dynamics. Our method requires computing the PBD model using an optimised GPU version. We accomplished this by parallelising the serial functions in the already existing CPU version of the opencloth PBD algorithm. We were able to establish faster frames per second by varying iterations amongst the CPU and GPU version as listed above, with GPU running faster than CPU for each test case. This proves the usefulness of our project. This, we feel, broadens the scope of PBD to include applications that require higher speed. We expect industry and practitioners to quickly adopt our method because it only takes minor changes to an existing PBD solver.

Acknowledgments Without the support of Dr. Muhammad Mobeen Moviana this project would not have been possible. His opencloth repository really provided us with a challenging project and left us with some extremely valuable insights in the world of GPU and its various applications.

References

[1] Digitalcommons.lsu.edu. (2022). Retrieved 6 May 2022, from https://digitalcommons.lsu.edu/cgi/viewcontent.cgi? article=4066&context=gradschool_dissertations.
[2] Matthias Müller. “Hierarchical Position Based Dynamics.” In Virtual Reality Interactions and Physical Simulations (VRIPhys2008). Eurographics Association, 2008.
[3] Jan Bender and Daniel Bayer. “Parallel Simulation of Inextensible Cloth.” In Virtual Reality Interactions and Physical Simulations (VRIPhys). Eurographics Association, 2008.
[4] Marco Fratarcangeli. “Comparing GLSL, OpenCL and CUDA: Cloth Simulation on the GPU.” In Game Engine Gems 2 First edition, edited by Eric Lengyel, pp. 365–379. A K Peters/CRC Press, 2011. Available at http://www.gameenginegems.net/.
[5] Robert Bridson, Ronald Fedkiw, and John Anderson. “Robust Treatment of Collisions, Contact and Friction for Cloth Animation.” ACM Transactions on Graphics 21:3 (2002), 594–603. Available at http://doi.acm.org/10.1145/566654. 566623.
[6] Takahiro Harada. “A Parallel Constraint Solver for a Rigid Body Simulation.” In SIGGRAPH Asia 2011 Sketches, SA ’11, pp. 22:1–22:2. New York, NY, USA: ACM, 2011. Available athttp://doi.acm.org/10. 1145/2077378.2077406.
[7] Richard Tonge, Feodor Benevolenski, and Andrey Voroshilov. “Mass Splitting for Jitter-free Parallel Rigid Body Simulation.” ACM Transactions on Graphics 31:4 (2012), 105:1–105:8. Available at http://doi.acm. org/10.1145/2185520.2185601.
[8] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. “Unified Particle Physics for Real-Time Applications.” ACM Transactions on Graphics (SIGGRAPH 2014) 33:4 (2014).
[9] Marco Fratarcangeli & Fabio Pellacini (2013) A GPU-Based Implementation of Position Based Dynamics for Interactive Deformable Bodies, Journal of Graphics Tools, 17:3, 59-66, DOI: 10.1080/2165347X.2015.1030525
[10] L. Verlet. “Computer Experiments on Classical Fluids. Ii. Equilibrium Correlation Functions.” Physical Review 165 (1967), 201–204.