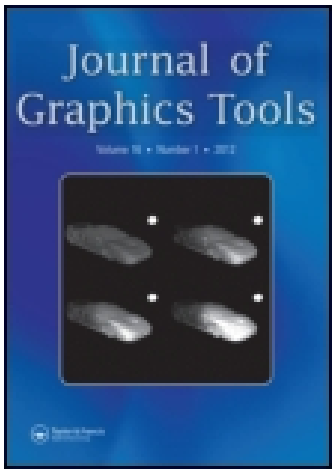


Journal of Graphics Tools
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ujgt21

A GPU-Based Implementation of Position Based Dynamics for Interactive Deformable Bodies

Marco Fratarcangeli[a] & Fabio Pellacini[b]
[a] Applied IT, Chalmers University of Technology, Gothenburg, Sweden
[b] Computer Science, Sapienza University of Rome, Rome, Italy
Published online: 09 Jul 2015.

To cite this article: 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

To link to this article: http://dx.doi.org/10.1080/2165347X.2015.1030525

Journal of Graphics Tools, 17:59–66, 2013

ISSN: 2165-347X print / 2165-3488 online
DOI: 10.1080/2165347X.2015.1030525

Routledge
Taylor & Francis Group

A GPU-BASED IMPLEMENTATION OF POSITION BASED DYNAMICS FOR INTERACTIVE DEFORMABLE BODIES

Marco Fratarcangeli[1] and Fabio Pellacini[2]

[1]Applied IT, Chalmers University of Technology, Gothenburg, Sweden
[2]Computer Science, Sapienza University of Rome, Rome, Italy

Position Based Dynamics (PBD) is a popular approach used for animating constrained particle systems representing soft bodies, rigid bodies, and fluids. In this article, we present a massively parallel implementation of PBD for fast, interactive animation of deformable bodies. We divide the set of constraints in independent partitions using a fast, greedy coloring graph algorithm. Then, during the animation, the constraints belonging to each partition are solved in parallel on the GPU. We employ an efficient simulation pipeline using a memory layout that favors both the memory access time for computation and batching for visualization. Our experiments show that the achieved performance speed-up is several orders of magnitude faster than its serial counterpart.

1. INTRODUCTION

Position Based Dynamics (PBD) [Müller et al. 07, Bender, Müller, et al. 14] has been employed in a broad range of applications, from knot simulation [Kubiak et al. 07], to face animation [Fratarcangeli 12], and automatic character skinning [Deul and Bender 13, Abu Rumman and Fratarcangeli, to appear]. Its original formulation considered just soft bodies, such as cloth and inflatable balloons, and it proved to be unconditionally robust, controllable, and fast. Recently, several techniques have been presented to extend it to both rigid bodies [Deul et al. 14] and fluids [Macklin and Müller 13]. Strain tensor constraints have been proposed [Müller et al. 14, Bender, Koschier, et al. 14], which allows animating complex physical phenomena, including lateral contraction, bending, plasticity, and anisotropy. An unified framework presented by [Macklin et al. 14] employs PBD as a building block to model in real time the animation of gasses, liquids, deformable solids, and rigid bodies, including interaction and collision with each other.

In the original PBD approach, described in [Müller et al. 07], the set of constraints are solved in a Gauss–Seidel fashion, one after the other, sequentially. As a constraint is solved, the particles influenced by it are immediately updated and then the following constraint is considered. This process is iterated several times for each animation frame; for each iteration, the difference between the current and the optimal solution of the system decreases. This serial approach is efficient when the number of constraints is relatively small and, thus, the number of iterations needed for reaching a good approximation of the global solution is low. However, when a high number of constraints is involved, both the computational cost of a single iteration and the number of iterations for each frame increases. In this case, the serial solution of the set of constraints quickly becomes unsuitable for interactive animation.

In this article, we present a technique for parallelizing the constraint satisfaction process, improving the speed of convergence of the

Submitted September 18, 2014; Accepted March 13, 2015.
Address correspondence to Marco Fratarcangeli, Applied IT, Chalmers University of Technology, Kuggen, Lindholmsplatsen 1, Gothenburg 41296, Sweden. E-mail: marcof@chalmers.se

solver and the time performance of the whole method, while retaining the robustness and the inherent simplicity of the original PBD.

1.1. Prior Work

A hierarchical, ad hoc, position-based approach for cloth by accelerating the convergence speed of the solver is described in [Müller 08]. A red-black parallel Gauss–Seidel schema for animating inextensible cloth using a force-based system was employed by [Bender and Bayer 08]. Although providing excellent performance, this method is restricted to meshes with a regular grid topology. The mesh is subdivided into strips of constraints. The strips that have no common particles are independent from each other and can be solved in parallel. A similar approach, relying on a Jacobi-like solver, has been presented in [Fratarcangeli 11]. All of these solvers are suitable for grid-like meshes (e.g., cloths), but lack the generality needed to simulate objects with arbitrary topology.

Jacobi and Gauss–Seidel iterative solvers are often used in the context of interactive computer animation because they usually provide results faster than direct methods, even though, in general, the obtained results have a larger residual error. They have been employed for contact response [Bridson et al. 02, Harada 11, Tonge et al. 12]. Recently, a unified framework for rigid bodies, soft bodies, and fluids was proposed [Macklin et al. 14], in which the animated objects are voxelized using fixed-size spheres, and the constraints are solved using a Jacobi-like solver.

1.2. Contribution

We present a technique based on a graph coloring algorithm to parallelize the solving process of the constraints connected in an arbitrary way. This allows animating interactively volumetric objects involving a larger number of constraints. As described in [Bender, Müller, et al. 14], the central idea is to build a graph in which each node is mapped to a constraint and two nodes are connected if the corresponding constraints share at least one particle. The graph is colored using a *distance-1* algorithm, such that neighbor nodes do not share the same color. By definition, this means that nodes with the same color do not share any particle. The constraints assigned to the same color can then be solved in parallel during the animation.

FIGURE 1. 100 nonconvex objects, consisting of more than 800 K constraints, fall on the ground. Each animation step is computed in 1.4 ms.

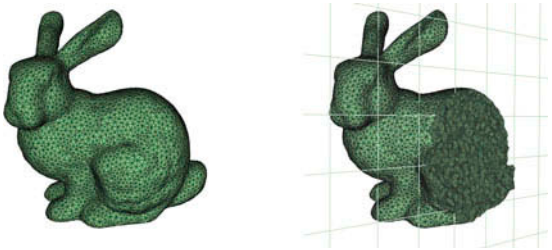


FIGURE 2. An input tetrahedral mesh.

We provide a parallel implementation targeting modern GPUs' architectures, able to interactively animate both flat and tetrahedral meshes, as in the example shown in Figure 2.

Differently from [Macklin et al. 14], we employ a Gauss–Seidel solver instead of a weighted Jabobi because, as shown in Section 5, it allows faster convergence speed without requiring the voxelization of the input mesh.

2. POSITION BASED DYNAMICS

Position Based Dynamics (PBD) [Müller et al. 07, Bender, Müller, et al. 14] is a method based on Verlet integration [Verlet 67] for interactively animating deformable objects. The objects are modeled as a set of n particles whose motion is governed by a set of m non linear constraints. The system of constraints is solved using Gauss–Seidel iterations by directly updating the particle positions. PBD avoids the use of internal forces, and the positions are updated such that the angular and the linear momenta are implicitly conserved. In this way, the whole process is not affected by

the typical instabilities of interactive, physically based methods.

The set of constraints is composed by nonlinear equality and inequality equations such that:

$$C_i(\mathbf{p}) \succ 0, \quad i = 1, \ldots, m, \tag{1}$$

where the symbol $\succ$ stands for either $=$ or $\geq$, $\mathbf{p} = \left[\mathbf{p}_1^T, \ldots, \mathbf{p}_n^T\right]^T$ is the vector of particle positions, n is the number of particles, and m is the number of constraints. The constraints are generally nonlinear and they are solved sequentially through Gauss–Seidel iterations. Each equation is linearized individually in the neighborhood of C around the current configuration $\mathbf{p}$ to find the correction $\Delta\mathbf{p}$:

$$C_i(\mathbf{p} + \Delta\mathbf{p}) \approx C_i(\mathbf{p}) + \nabla_{\mathbf{p}} C_i(\mathbf{p}) \cdot \Delta\mathbf{p} = 0, \tag{2}$$

where $\nabla_{\mathbf{p}} C_i(\mathbf{p})$ is the vector containing the derivatives of the equation C_i w.r.t. the n components of $\mathbf{p}$.

The correction $\Delta\mathbf{p}$ is imposed to be in the direction of $\nabla_{\mathbf{p}} C(\mathbf{p})$:

$$\Delta\mathbf{p} = \lambda_i \nabla_{\mathbf{p}} C_i(\mathbf{p}). \tag{3}$$

This condition implicitly conserves the linear and angular momenta, while at the same time allowing solution of the under determined system of constraints. Combining Equations 2 and 3 yields:

$$\lambda_i = -\frac{C_i(\mathbf{p})}{\left|\nabla_{\mathbf{p}} C_i(\mathbf{p})\right|^2}. \tag{4}$$

2.1. Stretch Constraint

We define one *stretch* constraint for the particles $(\mathbf{p}_1, \mathbf{p}_2)$ at the end points of each edge of the input mesh, including the edges of the internal tetrahedrons:

$$C(\mathbf{p}_1, \mathbf{p}_2) = \left|(\mathbf{p}_1 - \mathbf{p}_2)\right| - d = 0, \tag{5}$$

is used to keep particles $\mathbf{p}_1$ and $\mathbf{p}_2$ at distance d, where d is the rest length of the edge.

Given the configuration $(\mathbf{p}_1, \mathbf{p}_2)$ of two particles connected by a stretch constraint, the corrections to the positions (Equations 3) in order to satisfy the constraint are:

$$\begin{aligned} \Delta\mathbf{p}_1 &= -\frac{1}{2} k_s \left(\left|\mathbf{p}_1 - \mathbf{p}_2\right| - d\right) \frac{\mathbf{p}_1 - \mathbf{p}_2}{\left|\mathbf{p}_1 - \mathbf{p}_2\right|}, \\ \Delta\mathbf{p}_2 &= +\frac{1}{2} k_s \left(\left|\mathbf{p}_1 - \mathbf{p}_2\right| - d\right) \frac{\mathbf{p}_1 - \mathbf{p}_2}{\left|\mathbf{p}_1 - \mathbf{p}_2\right|}, \end{aligned} \tag{6}$$

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

2.2. Tetrahedral Volume Constraint

We define one *volume* constraint for the particles $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4)$ at the corners of each tetrahedral of the mesh. The volume constraint maintains the rest volume of four particles forming a tetrahedron, enforcing the conservation of the total volume of the mesh:

$$C(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = \frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \cdot \mathbf{p}_{4,1} - V_0. \tag{7}$$

where $\mathbf{p}_{i,j}$ is the short notation for $\mathbf{p}_i - \mathbf{p}_j$ and V_0 is the rest volume of the tetrahedral.

The gradient with respect to each particle is:

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

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

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

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

The correction of each particle belonging to the tetrahedron is:

$$\begin{aligned}
\Delta\mathbf{p}_1 =& -\frac{1}{6}\cdot s\cdot k_v\cdot\left(\left(\mathbf{p}_{2,1}\times\mathbf{p}_{3,1}\right)+\left(\mathbf{p}_{3,1}\times\mathbf{p}_{4,1}\right)\right.\\
&\left.+\left(\mathbf{p}_{4,1}\times\mathbf{p}_{2,1}\right)\right),\\
\Delta\mathbf{p}_2 =& \frac{1}{6}\cdot s\cdot k_v\cdot\left(\mathbf{p}_{2,1}\times\mathbf{p}_{3,1}\right),\\
\Delta\mathbf{p}_3 =& \frac{1}{6}\cdot s\cdot k_v\cdot\left(\mathbf{p}_{3,1}\times\mathbf{p}_{4,1}\right),\\
\Delta\mathbf{p}_4 =& \frac{1}{6}\cdot s\cdot k_v\left(\mathbf{p}_{4,1}\times\mathbf{p}_{2,1}\right),
\end{aligned}\tag{12}$$

where k_v is the stiffness parameter and s is the scaling factor:

$$s=\frac{\frac{1}{6}\left(\mathbf{p}_{2,1}\times\mathbf{p}_{3,1}\right)\cdot\mathbf{p}_{4,1}-V_0}{\sum_{i=1}^{4}\left|\nabla_{\mathbf{p}i}C\left(\mathbf{p}_i\right)\right|^2}.\tag{13}$$

3. ALGORITHM AND IMPLEMENTATION

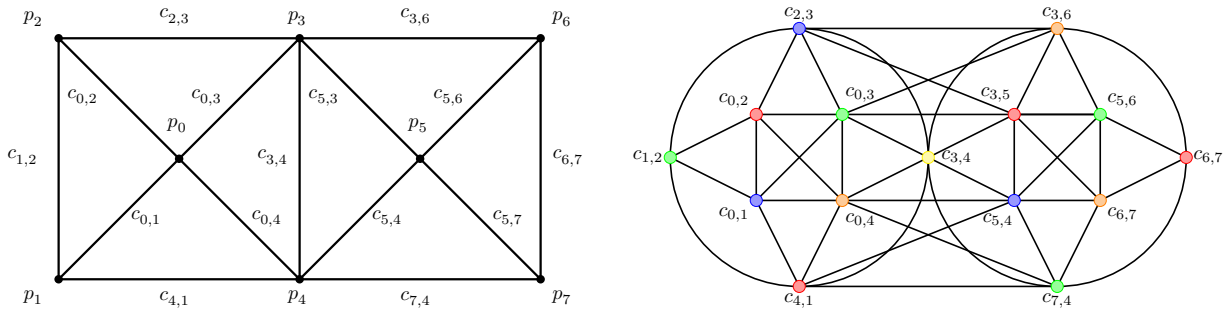
3.1. Graph Coloring

We implemented a Gauss–Seidel solver in a parallel fashion to speed-up the constraints' solving process. We define a graph with a node for every constraint in the system. Two nodes of the graph are connected if the corresponding constraints share at least one particle. Each color corresponds to a partition of constraints. We solve all the constraints belonging to a partition in parallel: we instantiate a thread for each constraint within the same partition. This way, all the constraints assigned to the same color are solved with a single kernel call. The number of kernel calls to compute one iteration of the parallel Gauss–Seidel solver is thus equal to the number of colors instead of the number of constraints, as in the serial approach. Figure 3 depicts this mechanism for a simple mesh composed by stretch constraints.

The graph coloring problem, in its simplest form, involves the assignment of colors to each node in the graph, such that two connected nodes do not share the same color. In other words, given a graph $G(V, E)$ and a set S of colors, a proper coloring is a map $c : V \to S$ s.t. $\forall \langle u, v\rangle \in E, c(u) \neq c(v)$.

Finding the minimal number of colors for coloring a generic graph G (the *chromatic number*) is known to be NP-hard [Garey and Johnson 79].

A widely known approach is the following: let $v_1, v_2, \ldots, v_n$ be an ordering of the nodes of the graph $G = (V, E)$; for $k = 1, 2, \ldots, n$, the sequential algorithm, assign v_k to the smallest possible color, as depicted in Algorithm 1, which is able to provide a good solution and can be computed during the initialization phase in $\mathcal{O}(n)$.



(a) A set of 8 particles connected by 15 stretch constraints. Here $c_{i,j}$ is the short notation for $C(p_i, p_j)$.

(b) The corresponding dual graph.

FIGURE 3. A graph coloring algorithm is applied to a simple particle system to parallelize the computation of the constraints. In this simple case, the constraints are solved in five steps, the number of colors, instead of 15, the number of constraints. Table 1 provides more detailed figures about the number of steps required to solve the constraints.

Algorithm 1. (Greedy Heuristic for Coloring a Graph $G(V, E)$)

1: let $v_1, v_2, \ldots, v_n$ be an ordering of V
2: **for** $i = 1$ to n **do**
3: determine forbidden colors to v_i
4: assign v_i the smallest permissible color
5: **end for**

We used the *smallest-last* ordering defined in [Matula and Beck 83, Coleman and More 83]. Assume that the vertices $v_{k+1}, \ldots, v_n$ have been selected.

Choose v_k so that the degree of v_k in the subgraph induced by

$$V - \{v_{k+1}, \ldots, v_n\}$$

is minimal. This choice guarantees that Algorithm 1 produces a coloring with at most

$$max\{1 + \delta(G_0) : G_0 \text{ is a subgraph of G}\} \tag{14}$$

colors, where $\delta(G_0)$ is the smallest degree of the vertices in G_0.

4. GPU IMPLEMENTATION

During the design of an algorithm for the GPU, it is critical to minimize the communication overhead between the CPU and the GPU, reducing the amount of data that travels on the main memory bus. The time spent for transmitting data on the bus is actually one of the primary bottlenecks that strongly penalize the time performance [nvB 13].

We designed our system in order to minimize the amount of data that travels on the PCI bus and keep the data on the GPU as much as possible. In the initialization phase, we load all the data required for the animation on the video memory. Then, during the animation phase, we update the data structures directly on the GPU. In this way, the CPU is not involved in the animation process (besides being responsible for calling the GPU kernels), and any data exchange using the bus is avoided.

We use a structure-of-arrays approach to store all the properties of the particles and the constraints. Each array stores one of the properties for all the objects involved in the animation. The state of the particles consists of the following properties:

- *current position* `float P[3 * n];`
- *past position* `float P`$^{-1}$ `[3 * n];`
- *mass* `float mass[n];`

where n is the total number of particles represented in the scene. The properties describing each type of constraints are:

- *rest value* `float d[m];`
- *stiffness* `float s[m];`
- *list of affected particles* `int id`$_p$ `[m * c];`

where m is the number of constraints and c is the number of particles influenced by the type of constraints. For example, for stretch constraints $c = 2$ and for tetrahedral constraints $c = 4$.

Using structure-of-arrays allows us (1) to perform coalesced accesses to the video memory and (2) to load only the required data for each kernel call. For example, whereas the Verlet integration step requires all the properties for the particles, the constraint solving step requires just the current position's array. The outcome of the computation, stored in the array P, is mapped to a Vertex Buffer Object and rendered with a single OpenGL drawing call.

5. RESULTS

We have implemented our technique in CUDA and timings for various scenes were measured on an NVIDIA GTX 780 ti GPU (Table 1). We also compare with both the standard PBD implemented in c++ and running on a single core, and with our technique implemented in c++ using Intel Thread Building Blocks technology [Reinders 07], running on an Intel i7-5930K processor with 12 cores.

These times do not include rendering and collision handling. Figure 1 shows a set of 100 tori falling on the ground and colliding with each other. Figure 6 shows a flat cloth

TABLE 1. Quantitative results of our experiments on three different scenes (s0, s1, s2) depicted, respectively, in Figures 6, 7, and 1. For each scene, the number and type of constraints (s: stretch, b: bend, t: tetrahedral) are reported. For each type of constraint, the number of colors is reported, which corresponds to the number of phases required to compute one Gauss–Seidel iteration. We run the animations on a single core CPU (cpu st), a multi core CPU (cpu mt), and a GPU and report the average frames-per-second over 500 frames

		Constraints / colors				avg fps		
	Particles	s	b	t	its	cpu st	cpu mt	gpu
s 0	10K	29K/8	29K/18	–	8	25	73	1105
s 1	16K	80K/64	42K/20	50K/124	4	15	45	326
s 2	91K	493K/17	–	313K/26	8	3	5	700

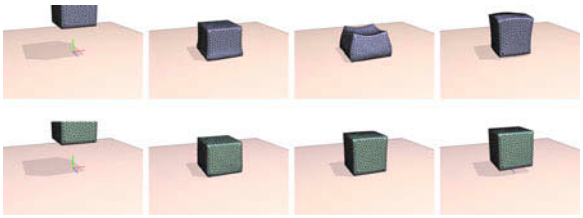


FIGURE 4. Falling cube composed of 25 K constraints; the time budget for each animation frame is 5 ms. The cube animated using the weighted Jacobi solver (*upper row*) is softer than that obtained with Gauss–Seidel (*lower row*) because its convergence rate is slower.

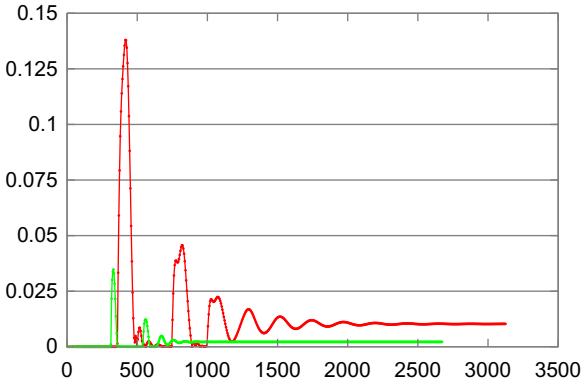


FIGURE 5. Convergence graph of the falling cube animation (Figure 4): residual error on constraints versus time (in ms). Color key: Green, Gauss–Seidel; Red, averaged Jacobi.

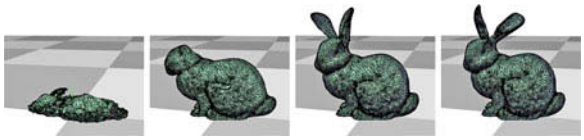


FIGURE 6. A flattened Stanford Bunny restores its original volume. The volumetric mesh consists of 170 K constraints and each animation step is computed in 3 ms.

FIGURE 7. Cloth consisting of 60 K constraints falls and collides with a tube. Each animation step is computed in 0.1 ms.

falling on a tube, and Figure 7 shows a volumetric mesh initially flattened to the ground, which restores its original volume. The tetrahedral meshes used in Figure 1 and Figure 7 are obtained using [Boissonnat et al. 02]. The corresponding animations are shown in the accompanying video.

5.1. Discussion and Limitation

The performance of the GPU solver is bounded by the number of times the GPU kernels are called, rather than the number of particles and constraints involved into the animation. This is depicted in the case of *s0* and *s2* in Table 1, where the performance speed-up of the GPU is higher than *s1* because, despite the larger number of particles, the number of colors is smaller than in the other cases.

The main issue with using graph coloring for parallelizing the solving process with Gauss–Seidel is that the number of constraints for each color varies significantly. This leads to an asymmetric amount of work in each kernel call and, therefore, the parallel performance is not optimal. This problem does not exist in the weighted Jacobi approach, used in [Macklin et al. 14]: in a first parallel pass each constraint is satisfied separately and then, in a second pass, all the displacements influencing each particle are summed and averaged. This process is clearly simple to implement on a parallel processor such as the GPU, and it is faster to execute; however, the convergence speed to reach a solution is slow compared

with the other methods for solving linear systems. We compared the convergence speed of the weighted Jacobi solver with the Gauss–Seidel approach, using a test scene with a falling cube composed of 25 K constraints (Figure 4). We imposed a time budget of 5 ms for each animation frame. The iterations for the Jacobi solver were fast to compute, and we accommodated 24 iterations within the time budget for each animation frame. Using the Gauss–Seidel approach, each iteration requires a number of kernel calls equal to the number of colors, and so it needs, in general, more time than a Jacobi iteration. For this reason, only eight iterations could be accommodated in the given time budget. Nonetheless, the convergence speed of the Gauss–Seidel solver is higher than the Jacobi, and overall, the solution of the system is reached faster, as shown in Figure 5 and in the accompanying video.

ORCID

Marco Fratarcangeli http://orcid.org/0000-0002-1156-3760

FUNDING

This publication was made possible by a National Priorities Research Program grant from the Qatar National Research Fund (a member of The Qatar Foundation).

AUTHORS' NOTE

The statements made herein are solely the responsibility of the author[s].

REFERENCES

[Abu Rumman and Fratarcangeli, to appear] Nadine Abu Rumman and Marco Fratarcangeli. "Position-Based Skinning for Soft Articulated Characters." *Computer Graphics Forum*. Preprint available from World Wide Web (http://dx.doi.org/10.1111/cgf.12533).

[Bender and Bayer 08] Jan Bender and Daniel Bayer. "Parallel Simulation of Inextensible Cloth." In *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Eurographics Association, 2008.

[Bender, Koschier, et al. 14] Jan Bender, Dan Koschier, Patrick Charrier, and Daniel Weber. "Position-Based Simulation of Continuous Materials." *Computers & Graphics* 44 (2014), 1–10.

[Bender, Müller, et al. 14] Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. "A Survey on Position-Based Simulation Methods in Computer Graphics." *Computer Graphics Forum*, pp. 1–25. Available at http://dx.doi.org/10.1111/cgf.12346, 2014.

[Boissonnat et al. 02] Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. "Triangulations in CGAL." *Computational Geometry-Theory And Applications* 22:1–3 (2002), 5–19. Available at http://dx.doi.org/10.1016/S0925-7721(01)00054-2.

[Bridson et al. 02] 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.

[Coleman and More 83] Thomas F. Coleman and Jorge J. More. "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems." *Journal of Numerical Analysis* 20 (1983), 187–209.

[Deul and Bender 13] Crispin Deul and Jan Bender. "Physically-Based Character Skinning." In *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Lille, France: Eurographics Association, 2013.

[Deul et al. 14] Crispin Deul, Patrick Charrier, and Jan Bender. "Position-Based Rigid-Body Dynamics." *Computer Animation and Virtual Worlds*. Available at http://dx.doi.org/10.1002/cav.1614, 2014.

[Fratarcangeli 11] 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/.

[Fratarcangeli 12] Marco Fratarcangeli. "Position-Based Facial Animation Synthesis." *Computer Animation and Virtual Worlds* 23:3–4 (2012), 457–466. Available at http://dx.doi.org/10.1002/cav.1450.

[Garey and Johnson 79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[Harada 11] 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 at http://doi.acm.org/10.1145/2077378.2077406.

[Kubiak et al. 07] B. Kubiak, N. Pietroni, M. Fratarcangeli, and F. Ganovelli. "A Robust Method for Real-Time Thread Simulation." In *ACM Symposium on Virtual Reality Software and Technology (VRST)*. Newport Beach, CA, USA: ACM, 2007.

[Macklin and Müller 13] Miles Macklin and Matthias Müller. "Position Based Fluids." *ACM Transactions on Graphics* 32:4 (2013), 1–12.

[Macklin et al. 14] 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).

[Matula and Beck 83] David W. Matula and Leland L. Beck. "Smallest-Last Ordering and Clustering and Graph Coloring Algorithms." *Journal of the ACM* 30:3 (1983), 417–427. Available at http://doi.acm.org/10.1145/2402.322385.

[Müller et al. 07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. "Position Based Dynamics." *Journal of Visual Communication and Image Representation* 18:2 (2007), 109–118. Available at http://dx.doi.org/10.1016/j.jvcir.2007.01.005.

[Müller et al. 14] Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. "Strain Based Dynamics." In *Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA)*. New York, NY: ACM, 2014.

[Müller 08] Matthias Müller. "Hierarchical Position Based Dynamics." In *Virtual Reality Interactions and Physical Simulations (VRIPhys2008)*. Eurographics Association, 2008.

[nvB 13] *NVIDIA CUDA Compute Unified Device Architecture - Best Practices Guide*, Version 5.5 edition, 2013. Available at https://developer.nvidia.com/cuda-zone.

[Reinders 07] James Reinders. *Intel Threading Building Blocks*, First edition. Sebastopol, CA, USA: O'Reilly & Associates, 2007.

[Tonge et al. 12] 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.

[Verlet 67] L. Verlet. "Computer Experiments on Classical Fluids. Ii. Equilibrium Correlation Functions." *Physical Review* 165 (1967), 201–204.