SIGRAD 2014
M. Obaid, D. Sjölie, E. Sintorn and M. Fjeld (Editors)

Towards a Massively Parallel Solver for Position Based Dynamics

Marco Fratarcangeli and Fabio Pellacini

Sapienza University of Rome

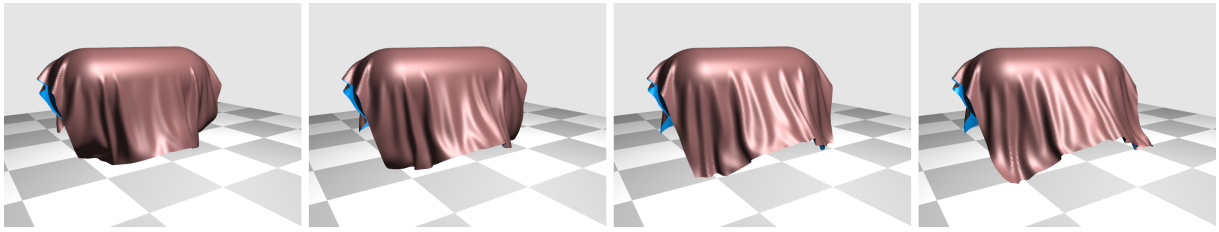

Figure 1: Interactive animations with our novel GPU-based implementation of the Position Based Dynamics approach: highly-detailed cloth model composed by 67K stretch constraints and 66K bending constraints.

Abstract

Position-based dynamics (PBD) is an efficient and robust method for animating soft bodies, rigid bodies and fluids. Recently, this method gained popularity in the computer animation community because it is relatively easy to implement while still being able to synthesize believable results at interactive rate. The animated bodies are modeled by using a large set of linearized geometrical constraints which are iteratively solved using a sequential Gauss-Seidel method on a single core CPU. However, when the animated scene involves a large number of objects, solving the constraints sequentially one after the other makes the computation of the motion too slow and not suitable for interactive applications. In this paper, we present a massively parallel implementation of position based dynamics which runs on the local GPU. In the initialization phase, the linearized geometrical constraints are divided in independent clusters using a fast, greedy coloring graph algorithm. Then, during the animation, the constraints belonging to each cluster are solved in parallel on the GPU. We employ an efficient simulation pipeline using a memory layout which favor both the memory access time for computation and batching for visualization. Our experiments show that the performance speed-up of our parallel implementation is several orders of magnitude faster than its serial counterpart.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation and Virtual Reality

1. Introduction

The interactive animation of soft and rigid bodies is still a challenging problem for the Computer Graphics community. The final users expect outstanding quality and for this reason, the mathematical models defining the animated objects have become more and more sophisticated through the years.

Position-Based Dynamics (PBD) [MHHR06] is a widely spread method for interactively animating rigid bodies, soft bodies and fluids. Its popularity is due to its robustness, speed and easiness of implementation. In PBD, each object is modeled as a particle system, and the relationship between particles is expressed using geometrical constraints. There exists several kinds of constraints. For example, the *stretch* constraint imposes the distance between two particles to an input distance. Another example is the *tetrahe-*

dral constraint, which imposes the volume of the tetrahedron formed by four particles to an input value. The set of constraints associated to an object forms a system which must be solved for each animation frame.

The Gauss-Seidel method is particularly suitable for solving such systems of linear equations. However, the underlying algorithm is sequential and thus unsuitable for parallel implementations. Previous works exploited the sparsity of the system to extract parallelism [KW03], or proposed parallel approaches requiring synchronization primitives [CA09], or aimed at the general case of sparse vector-matrix products [WBS*13]. In this paper, we propose a parallel scheme for the systems of linear, geometrical constraints employed in Position Based Dynamics for modeling animated objects, taking advantage of commodity parallel local architectures such as multi-core CPUs and GPUs.

The main feature of the graphics processing units (GPUs) is their high computational throughput. They offer a great speed-up allowing the execution of thousands of lightweight threads in parallel according to the SIMD (single instruction multiple data) paradigm: the threads execute the same instruction on different data. The introduction of the NVIDIA's Compute Unified Device Architecture (CUDA) [KH13] allowed to abstract from the underlying graphics hardware and use GPUs for general purpose programming (GPGPU).

In the initialization phase of our method, the geometrical constraints are divided in independent clusters using a fast, greedy coloring graph algorithm. Then, during the animation, the constraints belonging to a cluster are solved in parallel on the GPU. We employ an efficient simulation pipeline using a memory layout which favor both the memory access time for computation and batching for visualization.

The presented method allows for real-time animations of complex deformable bodies. To demonstrate the performance gain in practice, we present animation of deformable bodies, like cloth and volumetric objects. We compare between implementations on the single core CPU, multi-core CPU and GPU. We observe that in some cases the performance speed up of the GPU solver is 10x w.r.t. the other platforms.

Our contributions:

- A novel algorithm which divide the set of constraints in independent clusters using a greedy coloring algorithm; all the constraints are then solved using a GPU-based parallel Gauss-Seidel method.
- A novel GPU data structure storing all the objects in a single particle system. The data structure is used for both the animation and visualization minimizing the frame update time.

This parallel implementation allows the animation of soft bodies composed by tens of thousands of constraints in real-time.

2. Related Work

Position Based Dynamics has been employed in a broad range of applications from knot simulation [KPFG07] to face animation [Fra12] and automatic body skinning [DB13, RF14]. Its original formulation considered just soft bodies, like cloths and inflatable balloons. Recently several works have been proposed to include both rigid bodies [DCB14] and fluids [MM13]. An extensive description of this method and its derivatives can be found in [BMOT13]. Popular available implementations are included, among others, in the open-source *Bullet* physics engine [Cou10] and in the proprietary *PhysX SDK* [NVI13].

One of the main issues of PBD is the intrinsic slow convergence of the employed serial Gauss-Seidel solver. The geometrical constraints are solved one by one several times in an iterative way (see Sec. 3). Each time an iteration is completed, the difference between the current solution and the optimal one decreases. In order to reach a satisfying solution, usually just a small number of iterations is needed $(2-4)$, which is suitable for interactive applications. However, for complex scenes where a substantial number of constraints is involved (e.g., several hundred of thousands), the convergence is too slow, the number of iterations increases and the performance is not suitable anymore for real-time animations.

In [MÖ8], a hierarchical *ad-hoc* position-based approach for clothes is devised in order to accelerate the convergence of the solver. In [BB08], a red-black parallel Gauss-Seidel schema is used for animating inextensible clothes using a force-based system. While 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 particle are independent from each other and can be solved in parallel. Both the solvers presented in [MÖ8] and [BB08] lack the generality needed to simulate generic, volumetric objects with arbitrary topology.

3. Position Based Dynamics

In the Position Based Dynamics approach, a soft body is represented by a set of N particles and a set of M constraints. Each particle i is defined by its position $\mathbf{p}_i \in \mathbb{R}^3$. A geometric constraint j is a mathematical relationship between particles $C_j(\mathbf{p}) = 0$. When an external force is applied to the particles, like the gravity, or some particles are displaced for some reason, e.g., by user manual interaction, the geometrical constraints may be not satisfied anymore. For each animation frame, the system of constraints must be always satisfied, or at least, the error between the current and the optimal solution must be small enough to produce believable motion.

During the simulation, if the particles configuration $\mathbf{p}^k$ in

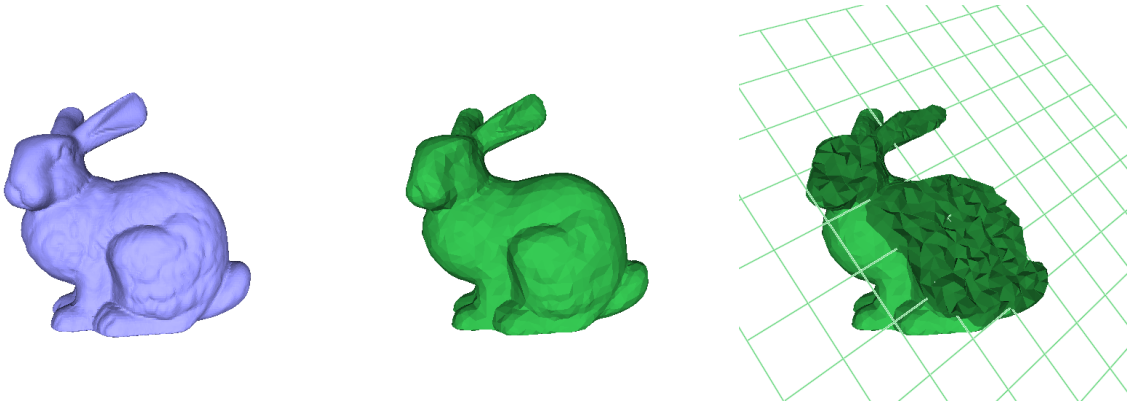

Figure 2: *Left:* An input surface mesh, composed by 34K vertices and 70 K faces. *Middle:* The corresponding tetrahedral mesh composed by 4K vertices and 16K tetrahedrals obtained using [BDP*02]. *Right:* Cross-section view showing the internal tetrahedrons of the input mesh.

a state k does not satisfy the set of constraints, then the iterative solver *projects* the particle positions in a valid state by finding a displacement $\Delta\mathbf{p}^k$ such that $C(\mathbf{p}^k + \Delta\mathbf{p}^k) = 0$. All the constraints in our method are functions $C_i(\mathbf{p}) = 0$. In this context, the constraints express a relationship (usually geometrical) between the particles. *Projecting* a set of particles according to a constraint means moving the particles such that their positions satisfy the constraint. The motion of the particles is computed inside a simulation loop. The particles are initially at rest state. This state can be perturbed by external conditions such as a force, like the gravity. The objective of the solver is to update the positions of the particles in order to keep the system of constraints satisfied.

Given $\mathbf{p}$, we want to find a correction $\Delta\mathbf{p}$ such that $C(\mathbf{p} + \Delta\mathbf{p}) = 0$. This system of non linear equations is *linearized*:

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} = 0 \tag{1}$$

and then iteratively solved. If $\Delta\mathbf{p}$ is chosen to be parallel to $\nabla_p C(\mathbf{p})$ (which is perpendicular to rigid body modes), then both linear and angular momenta are conserved. For a full mathematical description on how to solve this system, the interested reader can refer to [MHHR06,BMOT13]. For consistency, we briefly report the equations for solving distance and volume constraints in the following section.

3.1. Geometric Constraints

Starting from an input mesh like the one depicted in Fig. 2, we create one particle $\mathbf{p}_i$ for each vertex, one stretch constraint for each edge (including the internal ones), and one volume constraint for each tetrahedron. These constraints are described in the following subsections.

3.1.1. 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 mesh, including the edges of the internal tetrahedrons:

$$C(\mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d = 0 \tag{2}$$

where d is the rest length of the edge.

Given the configuration $\left(\mathbf{p}_1^k, \mathbf{p}_2^k\right)$ of two particles in the state k connected by a distance constraint, the corrections to the positions (Eq. 1) in order to satisfy a single constraint are:

$$\mathbf{p}_1^{k+1} = \mathbf{p}_1^k - \left(\left|\mathbf{p}_1^k - \mathbf{p}_2^k\right| - d\right) \frac{\mathbf{p}_1^k - \mathbf{p}_2^k}{\left|\mathbf{p}_1^k - \mathbf{p}_2^k\right|} \tag{3}$$

$$\mathbf{p}_2^{k+1} = \mathbf{p}_2^k + \left(\left|\mathbf{p}_1^k - \mathbf{p}_2^k\right| - d\right) \frac{\mathbf{p}_1^k - \mathbf{p}_2^k}{\left|\mathbf{p}_1^k - \mathbf{p}_2^k\right|} \tag{4}$$

3.1.2. 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:

$$C(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = \frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \cdot \mathbf{p}_{4,1} - V_0 \tag{5}$$

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 projection of each particle belonging to a tetrahedron is:

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

$$\Delta\mathbf{p}_2^{k+1} = \frac{s}{6}(\mathbf{p}_{2,1}^k \times \mathbf{p}_{3,1}^k) \tag{7}$$

$$\Delta\mathbf{p}_3^{k+1} = \frac{s}{6}(\mathbf{p}_{3,1}^k \times \mathbf{p}_{4,1}^k) \tag{8}$$

$$\Delta \mathbf{p}_4^{k+1} = \frac{s}{6}(\mathbf{p}_{4,1}^k \times \mathbf{p}_{2,1}^k) \tag{9}$$

where s is the scaling factor:

$$s = \frac{\frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \cdot \mathbf{p}_{4,1} - V_0}{\Sigma_{i=1}^4 \mid \nabla_{\mathbf{p}_i} C(\mathbf{p}_i) \mid^2} \tag{10}$$

and the gradients with respect to the particles are [MHHR06]:

$$\nabla_{\mathbf{p}_2} C(\mathbf{p}_2) = \frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \tag{11}$$

$$\nabla_{\mathbf{p}_3} C(\mathbf{p}_3) = \frac{1}{6}(\mathbf{p}_{3,1} \times \mathbf{p}_{4,1}) \tag{12}$$

$$\nabla_{\mathbf{p}_4} C(\mathbf{p}_4) = \frac{1}{6}(\mathbf{p}_{4,1} \times \mathbf{p}_{2,1}) \tag{13}$$

$$\nabla_{\mathbf{p}_1} C(\mathbf{p}_1) = -(\nabla_{\mathbf{p}_2} C(\mathbf{p}_2) + \nabla_{\mathbf{p}_3} C(\mathbf{p}_3) + \nabla_{\mathbf{p}_4} C(\mathbf{p}_4)) \tag{14}$$

4. Parallel Iterative Gauss-Seidel Solver

We consider scenes where thousands of constraints must be solved in real-time at least once every 16 ms to guarantee interactivity. During the animation, these constraints may be not satisfied due to external conditions, for example the user interacts with the model and move arbitrarily a set of particles, or an external force like gravity or wind is applied.

To solve the system, PBD employs a Gauss-Seidel solver. The constraints are solved iteratively one after the other, from the first to the last one. Then, the process starts over again and it is repeated a number of times, n_{its}. Increasing n_{its} leads to more precise solutions of the systems, sacrificing performance. Usually we employ a number of iterations between 2 and 24, depending on the topology of the system.

To speed-up the solving process, we implemented the Gauss-Seidel solver in a parallel fashion. 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 cluster of constraints. We solve all the constraints belonging to a cluster in parallel: we instantiate a thread for each constraint within the same cluster. This way, the system is solved in less steps than the sequential approach. Fig. 3 depicts this mechanism for a simple mesh composed by stretch constraints (Fig. 3.*Left*). The corresponding colored graph is shown on the right together with the corresponding clusters, one for each color.

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 amount of colors for coloring a generic graph G (the *chromatic number*) is known to be NP-hard [GJ79]. Usually, efficient greedy heuristics are employed to find an approximate solution. A widely-known approach is the following: let $v_1, v_2, \ldots, v_n$ be an ordering of the vertices of the graph $G = (V,E)$, for $k = 1,2,\ldots,n$ the sequential algorithm assign v_k to the smallest possible color. In general, an arbitrary ordering may perform very poorly but it is possible to show that, for any G, there exists at least one ordering of vertices for which the sequential algorithm produces an optimal coloring.

In our system, we used the *smallest-last* ordering defined in [MB83, CM83], which guarantees a coloring with at most

$$max\{1 + \delta(G_0) : G_0 \text{ is a subgraph of G }\} \tag{15}$$

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

5. Implementation

During the design of an algorithm for the GPU, it is critical to minimize the amount of data that travels on the main memory bus. The time spent on the bus is actually one of the primary bottlenecks that strongly penalize the performance [nvB13]. In fact, the transfer bandwidth of a standard PCI-express bus is 2-8 GB per second, while the internal bus bandwidth of a modern GPU is approximately 100-150 GB per second.

We designed our system in order to minimize the amount of data which 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 PCI bus is avoided.

In order to advance the animation step, the current state of the particle system must be stored in the video memory of the GPU. The current state is given by the following attributes:

1. the current position of each particle;
2. the previous position of each particle;
3. the external forces influencing the particles;

A different static array is created for each attribute during the initialization phase. Then, each array is loaded into video memory. Each array stores the attributes for all the particles. The size of each array is equal to the size of an attribute (4 floating-point values) multiplied by the number of the particles.

We employ the so-called *ping-pong* technique [Dro07] that is particularly useful when the input of a simulation step is the outcome of the previous one, which is the case in most

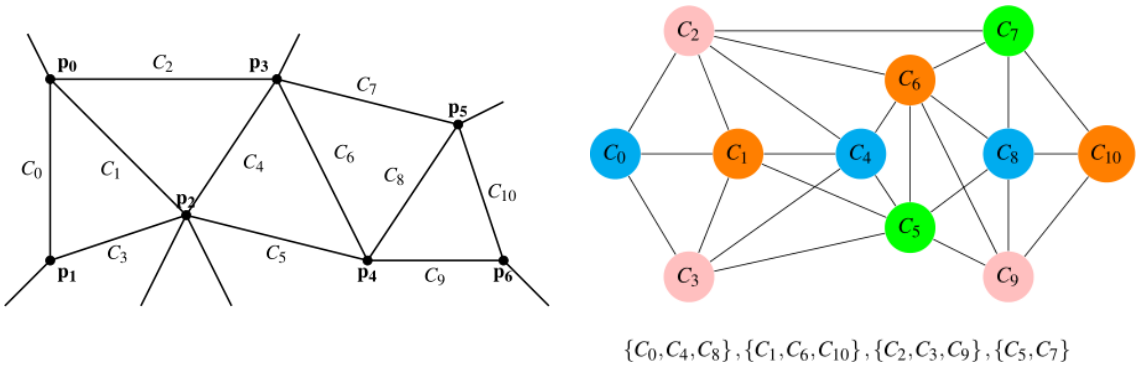

Figure 3: *Left.* A connected mesh of particles. Each edge corresponds to a stretch constraint. *Right.* Constraint graph. Each node corresponds to a constraint and two nodes are connected if the corresponding constraints share at least one particle. Nodes of the same color forms a cluster of constraints which can be solved in parallel.

of the animations based on particle systems. The basic idea is rather simple. In the initialization phase, two buffers are loaded on the GPU for each attribute, one buffer to store the input of the computation and the other to store the output. When the computation ends and the output buffers are filled with the results, the pointers to the two buffers are swapped such that in the following step, the previous output is considered as the current input.

The current results data are then stored in a Vertex Buffer Object (VBO), which is employed to render the current state of the deformable object; in this way the data never leaves the GPU achieving maximal performance. This mechanism is illustrated in Fig. 4.

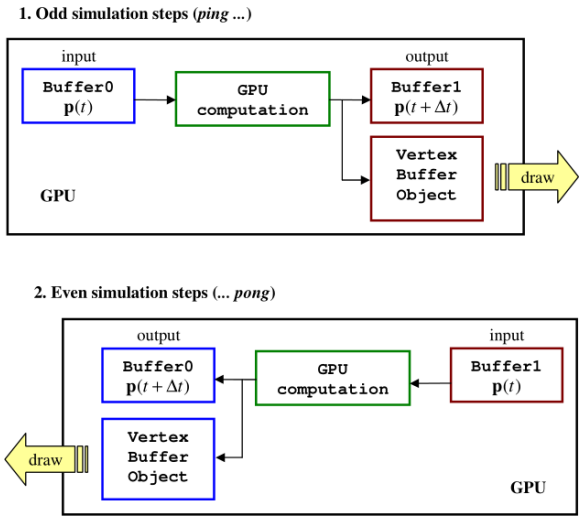

Figure 4: The ping-pong technique on the GPU. The output of a simulation step becomes the input of the following step. The current output buffer is mapped to a Vertex Buffer Object for visualization.

6. Results

We tested our proposed technique on three different scenes (Fig. 5). The first scene represents fixed cloth modeled using stretch and bending constraints under the influence of wind. The second scene represent a stack of clothes which falls under the gravity force and collides with a capsule. The third scene represents a volumetric character modeled using both stretch and tetrahedral volume constraints and deformed by user intervention. The tetrahedral meshes used in the experiments are obtained using [BDP*02].

To advance the animation, we used a 10 ms time step and 16 iterations per frame. We measured the time performances on the set of test scenes on a mass-market laptop equipped with an Intel i7 2.30 GHz processor (4 cores), 4GB RAM and a graphics card GeForce 610M with one multiprocessor and 2 GB VRAM.

We implemented standard Position Based Dynamics on a single core CPU, and the parallel version both on multithread CPU and GPU. The code of the kernels for solving the constraints is the same for each kind of platform. We used Intel Thread Building Blocks technology for implementing the multithreaded version on the CPU. The mean computation times are reported in Table 1. The corresponding animations are shown in the accompanying video.

7. Conclusions

In this paper, we introduced an early implementation of a massively parallel solver which can be used to speed up the computation of the popular Position Based Dynamics approach. It allows to simulate soft bodies, both cloths and volumetric ones, and it can be easily extended to involve rigid bodies, fluids and the two-way interactions among them. Table 1 shows the computational superiority of the GPU (even with a single multiprocessor!) against single and multi core CPUs.

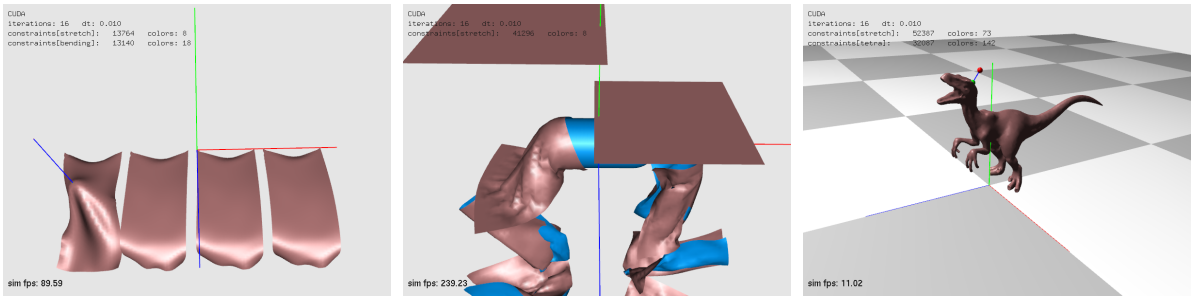

Figure 5: Test beds for our experiments. The number of constraints involved into the animation is reported in Table 1. *Left.* Fixed cloths under the influence of wind. Cloths are modeled with stretch and bending constraints. *Middle.* Cloths falling under the gravity force and colliding a capsule. *Right.* Volumetric deformation of a volumetric body.

	particles	constraints/colors			iterations	avg. computation time [ms]		
		stretch	blending	tetrahedral		CPU Single	CPU Multi	GPU
scene 0	4800	13764/8	13140/8	/	16	151.6	102.1	40.6
scene 1	14400	41296/8	/	/	16	104.2	48,8	9,7
scene 2	10452	52387/73	/	32087/142	16	256.4	122,1	82.0

Table 1: Quantitative results of our experiments on three different scenes depicted in Fig. 5. For each scene, it is reported the number and type of constraints. For each type of constraint, it is reported the number of colors, which corresponds to the independent clusters in which the constraints are grouped. We run the animations on a single core CPU, a multi core CPU and a GPU with a single multiprocessor and report the average computation time over 500 frames.

The performance of the GPU solver is bounded by the number of times the kernels are run, rather than the number of particles involved into the animation. This is depicted in the case of *scene 1* in Table 1, where the performance speed-up of the GPU is higher than in the other cases because, despite the big number of particles, the number of clusters is smaller than in the other cases.

A kernel call is required for solving each cluster. Each kernel is run a number of times equal to the number of clusters multiplied by the number of iterations. The number of clusters is equal to the number of colors required to color the graph constraint. The problem of maximizing the performance can be thus expressed as finding the minimal number of colors required to cover the graph, which is known to be NP-hard [GJ79].

We noted that the minimal number of colors is lower bounded by the size of the biggest clique in the constraint graph. In the future, we would like to explore the possibility to color the constraint graph with an arbitrarily low number of colors, allowing minimal changes in the topology, in order to maximize the performance on the GPU.

References

[BB08] BENDER J., BAYER D.: Parallel simulation of inextensible cloth. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (November 2008). 2

[BDP*02] BOISSONNAT J.-D., DEVILLERS O., PION S., TEILLAUD M., YVINEC M.: Triangulations in cgal. *Comput. Geom. Theory Appl. 22*, 1-3 (May 2002), 5–19. 3, 5

[BMOT13] BENDER J., MÜLLER M., OTADUY M. A., TESCHNER M.: Position-based methods for the simulation of solid objects in computer graphics. In *EUROGRAPHICS 2013 State of the Art Reports* (2013), Eurographics Association. 2, 3

[CA09] COURTECUISSE H., ALLARD J.: Parallel dense gauss-seidel algorithm on many-core processors. In *High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on* (June 2009), pp. 139–147. 2

[CM83] COLEMAN T. F., MORE J. J.: Estimation of sparse jacobian matrices and graph coloring problems. *Journal of Numerical Analysis 20* (1983), 187–209. 4

[Cou10] COUMANS E.: *Bullet 2.76 physics sdk manual*, 2010. 2

[DB13] DEUL C., BENDER J.: Physically-based character skinning. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (2013). 2

[DCB14] DEUL C., CHARRIER P., BENDER J.: Position-based rigid body dynamics. In *Computer Animation & Social Agents (CASA)* (2014). accepted. 2

[Dro07] DRONE S.: Real-time particle systems on the gpu in dynamic environments. In *ACM SIGGRAPH 2007 Courses* (New York, NY, USA, 2007), SIGGRAPH '07, ACM, pp. 80–96. 4

[Fra12] FRATARCANGELI M.: Position-based facial animation synthesis. *Computer Animation and Virtual Worlds 23*, 3-4 (2012), 457–466. 2

[GJ79] GAREY M. R., JOHNSON D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 4, 6

[KH13] KIRK D. B., HWU W.-M. W.: *Programming Massively*

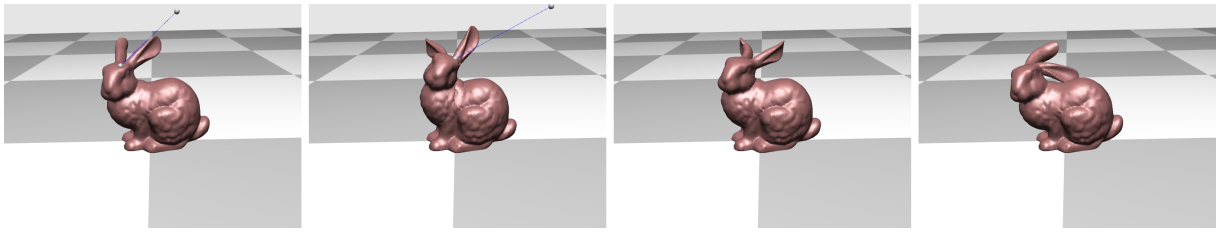

Figure 6: volumetric deformation of the Stanford Bunny model, composed by 80K stretch constraints and 50K tetrahedral constraints.

Parallel Processors: A Hands-on Approach, 2 ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2013. 2

[KPFG07] KUBIAK B., PIETRONI N., FRATARCANGELI M., GANOVELLI F.: "A Robust Method for Real-Time Thread Simulation". In *ACM Symposium on Virtual Reality Software and Technology (VRST)* (New Port Beach, CA, USA, November 2007), ACM, (Ed.). 2

[KW03] KRÜGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. *ACM Trans. Graph. 22*, 3 (July 2003), 908–916. 2

[MÖ8] MÜLLER M.: Hierarchical position based dynamics. In *Virtual Reality Interactions and Physical Simulations (VRIPhys2008)* (Grenoble, November 2008). 2

[MB83] MATULA D. W., BECK L. L.: Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM 30*, 3 (July 1983), 417–427. 4

[MHHR06] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Madrid, November 6-7 2006). 1, 3, 4

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (July 2013), 104:1–104:12. 2

[nvB13] : *NVIDIA CUDA Compute Unified Device Architecture - Best Practices Guide*, version 5.5 ed., 2013. 4

[NVI13] NVIDIA: *Physx sdk 9.13 documentation*, 2013. 2

[RF14] RUMMAN N. A., FRATARCANGELI M.: Position-based skinning. In *Eurographics/ACM Spring conference on Computer Graphics (SCCG)* (2014). accepted. 2

[WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D.: Efficient gpu data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26. 2