EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

Scalable Partitioning for Parallel Position Based Dynamics

M. Fratarcangeli[1] and F. Pellacini[2]

[1]Chalmers University of Technology, Gothenburg, Sweden
[2]Sapienza University of Rome, Italy

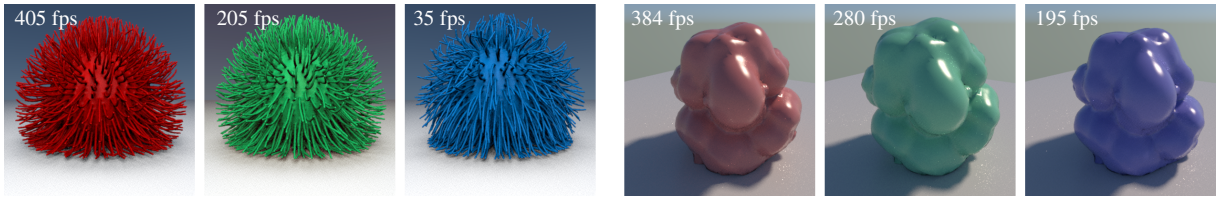


Figure 1: Interactive animation of deformable bodies, modeled using Position Based Dynamics [MHHR07] and composed of hundreds of thousands of constraints. We alter the constraints' topology for reducing the number of GPU kernel calls required for the computation, leading to significant speed-ups with negligible visual effects on the actual motion. *Color key:* Red, our approach. Green, parallel Gauss-Seidel. Blue, averaged Jacobi.

Abstract

We introduce a practical partitioning technique designed for parallelizing Position Based Dynamics, and exploiting the ubiquitous multi-core processors present in current commodity GPUs. The input is a set of particles whose dynamics is influenced by spatial constraints. In the initialization phase, we build a graph in which each node corresponds to a constraint and two constraints are connected by an edge if they influence at least one common particle. We introduce a novel greedy algorithm for inserting additional constraints (phantoms) in the graph such that the resulting topology is $\hat{q}$-colourable, where $\hat{q} \geq 2$ is an arbitrary number. We color the graph, and the constraints with the same color are assigned to the same partition. Then, the set of constraints belonging to each partition is solved in parallel during the animation phase. We demonstrate this by using our partitioning technique; the performance hit caused by the GPU kernel calls is significantly decreased, leaving unaffected the visual quality, robustness and speed of serial position based dynamics.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Parallel Processing—I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

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) [MHHR07] is a widely spread method for interactively animating rigid bodies, soft bodies and fluids [MMCK14,DCB14,MM13]. Its popularity is due to its robustness, speed and ease of implementation. In PBD, each object is modeled as a particle system, and the relationship between particles is expressed using geometrical constraints. The set of constraints is solved in a Gauss-Seidel fashion: the constraints are solved sequentially one by one. This iterative way is unsuitable for parallel computation, which however is very important because multi-core systems and massive parallel GPUs are ubiquitous today.

Graph coloring algorithms are a useful tool for extracting concurrency in parallel scientific computing, including iterative methods for sparse linear systems [JP94]. As described in [BMO*14], the serial Position-based Dynamics approach can be parallelized by building a graph with a node for each

constraint. Two nodes are connected by an edge if the corresponding constraints share at least one particle. Coloring the graph with a distance-1 algorithm such that two adjacent nodes do not have the same color, divides the set of nodes into q independent partitions, each one corresponding to a color (Fig. 2). Then, the constraints belonging to the same partition can be solved in parallel during the animation phase by using a dedicated GPU kernel call.

However, we observed that the performance hit introduced by the kernel calls drastically mitigates the performance speed up introduced by the parallelization (Table 1). In general, the partitions produced by standard coloring algorithms are fragmented and differ greatly in size, which leads to a high number of partitions (hence a high number of kernel calls), and an unbalanced workload (e.g., see Fig. 3).

nb kernel calls	constraints	ms/frame
344	200K	1.7
	400K	2.2
	800K	1.9
688	200K	3.8
	400K	4.0
	800K	3.7
1032	200K	5.3
	400K	5.5
	800K	13.0

Table 1: The overall computation time is heavily influenced by the number of GPU kernel calls rather than the size of the data (Performance measured on a Nvidia GeForce 660).

Contributions. This paper proposes a novel partitioning technique which allows to solve in parallel the original serial PBD approach. We alter the topology of the constraint graph G to make it divisible into an arbitrary number of independent partitions $\hat{q} \geq 2$, each one requiring a GPU kernel call to be solved. Allowing a low $\hat{q}$, we reduce the number of kernel calls which leads to an enhanced overall performance of the animation algorithm. Furthermore, the obtained independent partitions have similar cardinality, achieving a good load balancing.

The minimal amount of colors needed is bounded by the size $\omega(G)$ of the maximal cliques in the graph. Our goal is to use an arbitrary number $\hat{q} \leq \omega(G)$ of partitions. The core idea of our technique is to modify the topology of the cliques having size $q > \hat{q}$ in the constraint graph, such that these become $\hat{q}$-colourable. We modify the topology of the cliques by inserting *phantom* constraints and particles in appropriate places, as explained in Sec. 4. Modifying the topology, the original dynamics of the particles system is altered; however, in Sec. 5 we show that this difference is minimal.

2. Related Work

Position Based Dynamics. Position Based Dynamics [MHHR07, BMO*14] has been employed in a broad range of applications, from knot simulation [KPFG07] to face animation [Fra12] and automatic character skinning [DB13, RF14]. Its original formulation considered just soft bodies, like cloths and inflatable balloons. Recently, several works have been considering both rigid bodies [DCB14] and fluids [MM13]. Strain tensor constraints have been proposed in [MCKM14], which allows realistic cloth animation effects. The unified framework presented by [MMCK14] employs PBD as a building block to model in real-time the animation of gases, liquids, deformable solids and rigid bodies, including interaction and collision with each other. The computation of the dynamics is performed in parallel on a commodity graphics card. They voxelize input meshes into particles and then employ a parallel Jacobi solver combined with an under-relaxation method and a successive over-relaxation method. By contrast, in our approach we use a Gauss-Seidel solver in the same spirit of the original Position Based Dynamics approach but in a parallel fashion. Our approach is thus stable and robust, and

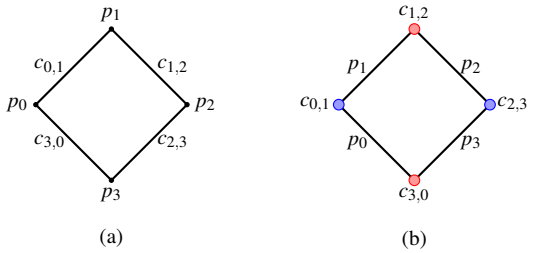


Figure 2: (a) Four particles connected by stretch constraints. (b) The corresponding dual graph: each node represents a constraint and two nodes are connected if they share at least one particle. The graph is bipartite and can be colored with two colors. Constraints belonging to the same colors can be solved in parallel.

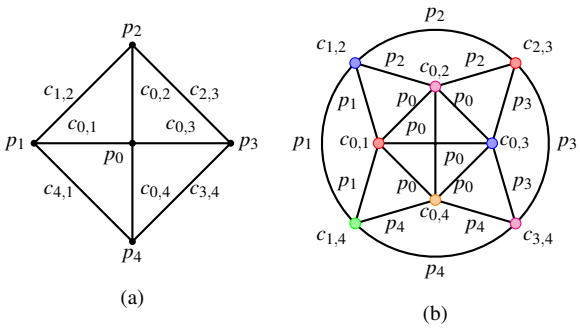


Figure 3: (a) A ring is composed of five particles connected together by stretch constraints. (b) The coloring of the corresponding dual graph results are much more fragmented than the configuration in Fig. 2b: at least five colors must be used thus the parallel computation requires more kernel calls.

allows us to directly use triangular and tetrahedral meshes without any additional preprocessing.

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 particles are independent from each other and can be solved in parallel. However both the solvers presented in [MÖ8] and [BB08] lack the generality needed to simulate volumetric objects with arbitrary topology.

Parallel Gauss-Seidel method. The Gauss-Seidel algorithm is an efficient and iterative method for solving linear systems of equations, such as the linearized geometrical constraints used in PBD. Its convergence is notoriously faster than other solvers (e.g., Jacobi), however, the underlying algorithm is inherently serial: the equations are solved one after the other in an iterative way. Each time an iteration is completed, the difference between the current solution and the optimal one decreases. Previous works have exploited sparsity in the linear system to extract parallelism and optimize the GPU implementation of popular solvers, like preconditioned conjugate gradient (PCG) [WBS*13] and GMRES [BCK11].

Iterative solvers have been extensively employed in contact resolution [BFA02]. A parallel iterative rigid body solver that avoids jitter artifacts at low iteration counts is presented in [TBV12]. The complementarity problem arising from the rigid body systems is solved iteratively by lumping the contacts in blocks, using parallel Gauss-Seidel to solve the contacts within the blocks, and Jacobi to combine the blocks together.

In [CA09, AFC*10], a parallel, coloring-based technique is presented for solving dense systems using the Gauss-Seidel method. In this method partitions are unbalanced and the thread synchronization relies on internal, potentially slow, atomic instructions. In contrast, our approach focuses on sparse systems and leads to balanced partitions which can be independently solved without explicit synchronization.

3. Background

Since we propose a partition technique for parallelizing the Position Based Dynamics (PBD) approach using a graph coloring algorithm, and because we want to keep the paper as self contained as possible, we briefly summarize the core idea of PBD and the basic theory of sequential vertex coloring.

3.1. Position Based Dynamics

Position Based Dynamics (PBD) [MHHR07, BMO*14] is a method based on Verlet integration 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 of non-linear equality and inequality equations such that:

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

where the symbol $\succ$ stands for either $=$ or $\geq$, $\mathbf{p} = \left[\mathbf{p}_1^T,\dots,\mathbf{p}_n^T\right]^T$ is the vector of particle positions, n is the number of particles and m is the number of constraints. For example, the distance constraint

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

is used to keep particles $\mathbf{p}_1$ and $\mathbf{p}_2$ at distance d. The constraints are generally non-linear, like in the just mentioned example, 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 $\Delta\mathbf{p}$:

$$C_i(\mathbf{p}+\Delta\mathbf{p}) \approx C_i(\mathbf{p}) + \nabla_{\mathbf{p}} C_i(\mathbf{p}) \cdot \Delta\mathbf{p} = 0 \tag{3}$$

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 $\nabla_{\mathbf{p}} C(\mathbf{p})$:

$$\Delta\mathbf{p} = \lambda_i \nabla_{\mathbf{p}} C_i(\mathbf{p}) \tag{4}$$

This condition implicitly conserves the linear and angular momenta for the single constraints while, at the same time, allowing us to solve the under-determined system of constraints. Combining Eq. 3 and 4 yields:

$$\lambda_i = -\frac{C_i(\mathbf{p})}{|\nabla_{\mathbf{p}} C_i(\mathbf{p})|^2} \tag{5}$$

3.2. Coloring Strategy

Partitioning as a coloring problem: To attack the problem of minimizing the number of partitions, we map the partitioning to a graph coloring problem. For this, we now recall some basic graph theory. A non-reflexive graph G is an ordered pair (V,E) where V is a finite and nonempty set of vertices, and the edges E are unordered pairs of distinct vertices:

$$E \subset \{(u,v) : u \neq v, \quad u,v \in V\}$$

A *distance-1 coloring* of a graph G is a mapping

$$\phi : V \rightarrow \{1,2,\ldots,q\} \quad \text{s.t.} \quad \phi(u) \neq \phi(v), \forall (u,v) \in E$$

If G can be colored with q colors, it is said to be q-colorable and the smallest q for which G is q-colorable is the chromatic number $\chi(G)$. Finding the chromatic number $\chi(G)$ is a NP-hard problem [GJ79]. A q-coloring ϕ of a graph G induces a partition of the vertices where each set is formed by the nodes with the same colors:

$$P_i = \{v \in V : \phi(v) = i, \quad i = 1,\ldots,q\}$$

where P_i is the partition corresponding to the i-the color.

Lower bound for the chromatic number $\chi(G)$: A graph $G_0 = (V_0, E_0)$ is a subgraph of $G = (V,E)$ if $V_0 \subset V$ and $E_0 \subset E$. Given a nonempty set V_0 of V, the subgraph $G_0 = (V_0, E_0)$ is induced by V_0 if

$$E_0 = \{(u,v) : (u,v) \in E, \quad u,v \in V_0\}$$

If each pair of distinct vertices in V_0 is adjacent then G_0 is a *clique* of G. Cliques are important in graph coloring problems because a lower bound for the chromatic number $\chi(G)$ of a graph G is the size $\omega(G)$ of the maximal clique in G.

Sequential Vertex Coloring: Most of the extensive literature on graph coloring is not relevant for the problem we are addressing. For example, algorithms for coloring planar graphs, requiring just 4 colors, are not suitable because if there is a clique of at least 5 vertices, then G is not planar (Kuratowski's theorem, [Kur30]).

Graph coloring algorithms designed for general graphs are not helpful because they are usually restricted to graphs with at most 100 vertices, and in our case the graph is generally much larger. Algorithms of time complexity $\mathcal{O}(n^2)$ or higher are not acceptable, and this requirement prunes away many coloring algorithms.

The greedy heuristic depicted in Alg. 1 is able to provide a good solution and it can be computed in $\mathcal{O}(n)$. In general, an

Algorithm 1 Greedy heuristic for coloring a graph

```
1: procedure GREEDY(G(V,E))
2:     let v_1, v_2, ..., v_n be an ordering of V
3:     for i = 1 to n do
4:         determine forbidden colors to v_i
5:         assign v_i the smallest permissible color
6:     end for
7: end procedure
```

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 fact, given ϕ an optimal coloring of G, if we feed Alg. 1 with the ordering of vertices so that $\{\phi(v_i)\}$ is not decreasing, then we obtain an optimal coloring.

We used the *smallest-last* ordering defined in [MB83, CM83]. 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 Alg. 1 produces a coloring with at most

$$max\{1 + \delta(G_0) : G_0 \text{ is a subgraph of G }\} \tag{6}$$

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

4. Partitioning Algorithm

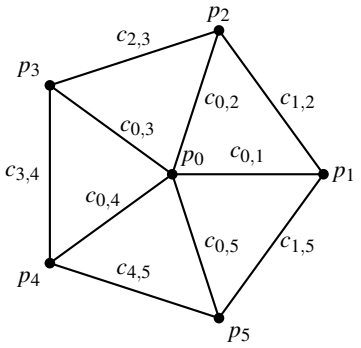
Equivalent Graph Coloring Problem: In the original PBD approach, the constraints are solved sequentially in a Gauss-Seidel fashion: the constraints are solved iteratively one after the other, from the first to the last one, then the process starts over again and is repeated a number of times n_{its} for each animation frame. Increasing n_{its} leads to a more precise solution of the system, sacrificing performance. Usually a number of iterations between 2 and 24 is used, depending on the topology of the particle system and the total number of constraints.

Our objective is to find a partitioning of the set of constraints such that the constraints belonging to each partition can be solved in parallel on commodity graphics hardware. A kernel call is invoked for solving one single subset of constraints belonging to a partition, and a lightweight thread is instantiated for solving a single constraint. Our goal is to minimize the number of kernel calls because the kernel launch overhead influences the performance significantly. Furthermore, the cardinality of all partitions must be approximately the same for achieving a good load balancing.

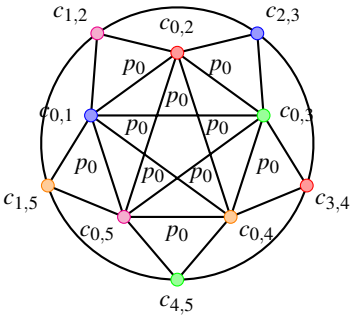
We formulate this partitioning as a coloring graph problem, where each partition (and thus a kernel call) corresponds to a color. As seen in Sec. 3.2, the chromatic number $\chi(G)$ is lower bounded by the size $\omega(G)$ of the maximal clique in the graph. Our goal is to use an arbitrary number $\hat{q}$ of partitions, which is usually smaller than $\omega(G)$.

The core idea of our algorithm is to modify the topology of the cliques having size $q > \hat{q}$ in the graph, such that these become $\hat{q}$-colorable. We modify the topology of the cliques by inserting *phantom* constraints in proper places, as explained in the following paragraph. We first introduce the concept of phantom constraints and phantom particles, and then proceed to depict the greedy strategy used to insert the phantoms into the original constraint graph.

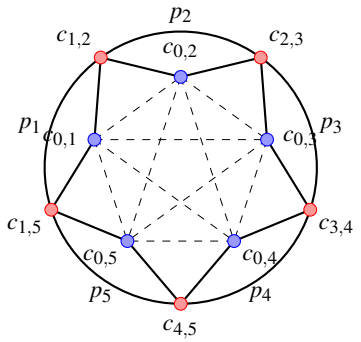
Phantoms: Given a particle $\mathbf{p}_i$, a *phantom particle* $\mathbf{p}_f$ associated to $\mathbf{p}_i$ is a particle which has initially the same state of $\mathbf{p}_i$. Given $\mathbf{p}_i$ and a set of phantom particles $F_i = \{\mathbf{p}_{f_0},\ldots,\mathbf{p}_{f_n}\}$ associated to $\mathbf{p}_i$, a *phantom constraint*



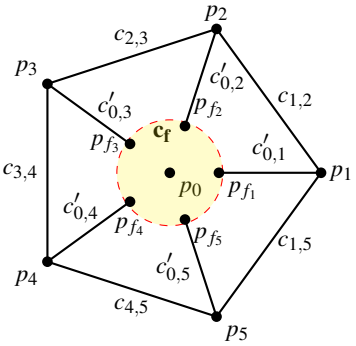
(a) A set of particles connected by stretch constraints. Here $c_{i,j}$ is the short notation for $C(p_i, p_j)$.



(b) The constraint graph corresponding to Fig. 4a. The graph contains a clique, a *complete graph* K_5, and it can not be colored with less than 5 colors.



(c) Removing the shared particle p_0 by adding a phantom constraint, the resulting constraint graph becomes bipartite and the coloring problem can be solved with just two colors.



(d) The set of particles corresponding to Fig. 4c. The phantom constraint C_f disconnects the original topology and binds the removed particle p_0 to the phantom particles $p_{f_1}, \ldots, p_{f_5}$.

Figure 4: Application of our partition technique to a simple particle system composed of 6 particles and 10 constraints.

$C_f(\mathbf{p}_i, \mathbf{p}_{f_0}, \ldots, \mathbf{p}_{f_n})$ projects the position of the particles to their barycenter:

$$\Delta\mathbf{p} = \frac{\sum_{k=1}^{N} \mathbf{p}_k}{N} - \mathbf{p}, \quad \forall \mathbf{p} \in \{\mathbf{p}_i \cup F_i\} \tag{7}$$

where $N = |\mathbf{p}_i \cup F_i|$. This corresponds to the constraint averaging step in averaged Jacobi. While averaged Jacobi performs this operation on all particles, we only apply it to a small subset. In other words, a phantom constraint is similar to a set of distance constraints with zero rest length between each particle and their barycenter. Instead of solving the distance constraints one by one, a phantom constraint projects all of its particles in a single step.

Phantoms Insertion Algorithm: Phantom particles and constraints are inserted into the topology of the graph constraint for making it $\hat{q} - colorable$ applying Algorithm 2.

Line (2) is used to initialize the set S of cliques in the graph. For this purpose, we use the Bron-Kerbosh algorithm [BK73] because it is proven to run in time $\mathcal{O}(dn3^{d/3})$ for sparse graphs, where n is the size of V and d is the *degenerancy* of of G, that is the smallest number such that every subgraph of G contains a vertex of degree at most d [ELS10]. Then, in line (3) the set S is sorted in decreasing order of the size of the cliques so that in the following lines the cliques $s \in S$ are processed from the biggest to the smallest. The processing of each clique s with size $|s| > \hat{q}$ begins from line (4). Lines (5)-(11) are used to find the particle $\mathbf{p}_{j_{max}}$ which is shared more times between the constraints $C_i(\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_n})$; the nodes forming the current clique s. The number of occurrences for each particle is counted, and then the most shared one $\mathbf{p}_{j_{max}}$ is selected. In lines (12)-(19), $\mathbf{p}_{j_{max}}$ is removed from every constraint in the clique s and is replaced by a phantom particle $\mathbf{p}_f$, which is initialized with the same state of $\mathbf{p}_{j_{max}}$. As a result of this, at the end of the process the constraints in the same clique do not share anymore $\mathbf{p}_{j_{max}}$ and the corresponding nodes disconnect. In other words, the size of the cliques decreases and its value is exactly the target number of colors $\hat{q}$. In lines (21)-(23), a phantom constraint C_f is created by grouping together the phantom particles, stored in the set F, together with the particle $\mathbf{p}_{j_{max}}$.

Fig. 4 provides an example of this process. We start from a

Algorithm 2 Greedy strategy to make a graph $\hat{q}-colorable$

```
1:  procedure MAKECOL(G, q̂)
2:      S ← AllCliquesIn(G), using [BK73]
3:      Sort S in order of decreasing size
4:      for each s ∈ S : |s| > q̂ do
5:          count[] ← 0
6:          for each C_i(p_{i_1}, ..., p_{i_n}) ∈ s do
7:              for each p_j ∈ {p_{i_1}, ..., p_{i_n}} do
8:                  count[j] ← count[j]+1
9:              end for
10:         end for
11:         j_max ← j : (count[j] = max(count[]))
12:         for each C_i(p_{i_1}, ..., p_{i_n}) ∈ s do
13:             F ← ∅
14:             if p_{j_max} ∈ {p_{i_1}, ..., p_{i_n}} then
15:                 Create p_f ← p_{j_max}
16:                 {p_{i_1}, ..., p_{i_n}} ← ({p_{i_1}, ..., p_{i_n}} − p_{j_max}) ∪ p_f
17:                 F ← F ∪ p_f
18:             end if
19:         end for
20:         if F ≠ ∅ then
21:             Create C_f(p_{j_max} ∪ F)
22:         end if
23:     end for
24: end procedure
```

simple configuration of particles connected by distance constraints $C(\mathbf{p}_i, \mathbf{p}_j)$ as shown in Fig. 4a. The corresponding constraint graph is shown in Fig. 4b: the nodes of the graph correspond to the distance constraints and the label on each edge provides the shared particle between the constraints at the endpoints. The graph contains a clique of five elements, and is 5-colorable. We select the most shared particle p_0 and remove it by adding five phantom particles and one phantom constraint (Fig. 4c). The corresponding configuration of particles is given in Fig. 4d. By doing this, the graph becomes 2-colorable and the constraints can be arranged in two partitions of approximately the same size. However, the original topology is changed and this is reflected in a different dynamic behavior of the animated object. The visual difference is minimal as assessed in Sec. 5 and in the accompanying video.

As Alg. 2 is executed, the number of vertices in a maximal clique in G (that is, the clique number $\omega(G)$), is decreased. We keep executing Alg. 2 until $\omega(G) = \hat{q}$, then we apply the sequential vertex algorithm (Alg. 1), for partitioning G. The lower bound of the number of colors (partitions) is the size of the maximal clique in G. We found in practice that these two numbers are equal in most cases.

5. Results

We used our partitioning algorithm in a constraint-centric solver written in c++/CUDA. We implemented the different types of constraints specified in [MHHR07]. Each type of constraint is solved by a dedicated GPU kernel. A kernel is called per each partition. Each thread processes a single constraint. The resulting particle positions are mapped to a vertex buffer object, and used to render in OpenGL; in this way the data always remains in video RAM, without the need of slow downloads towards the host. We defined a constraint graph for each type of constraint, partitioned it using our technique and then we ran a kernel call for each partition. All the phantom constraints were solved in parallel with a single kernel call.

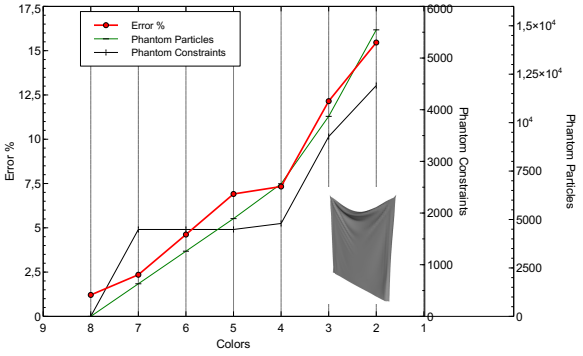


Figure 5: Relative error in cloth simulation. As the number of colors decreases, the difference with the exact solution increases. The cloth is composed of ~10K stretch constraints.

Fig. 7 and Fig. 8 compare the size of the partitions obtained using both plain Sequential Vector Coloring and our technique, increasingly reducing the number of colors. The histograms on the side of the pictures show the reduced fragmentation of the partitions.

Fig. 5 shows the relative error introduced by phantoms when the number of colors decreases, for a simple cloth animation. We measured the residual error versus time in the following scenes (included in the accompanying video), which converge to a rest configuration in a short amount of time (Fig. 6).

Squishy ball: A squishy ball, composed of ~452K stretch constraints and ~250K tetrahedral constraints, falls under gravity force.

Omotondo: A deformable object, composed of ~80K stretch constraints and ~65K tetrahedral constraints, is flattened and then left free to come back to its rest volume.

Tori: 400 deformable tori, for a total amount of ~493K stretch constraints and ~313K tetrahedral constraints, form a pile.

We used parallel Gauss-Seidel considering partitions obtained with plain Sequential Vertex Coloring and our algorithm, and compared the results with the solution of an averaged Jacobi solver. For this latter, we used a number of

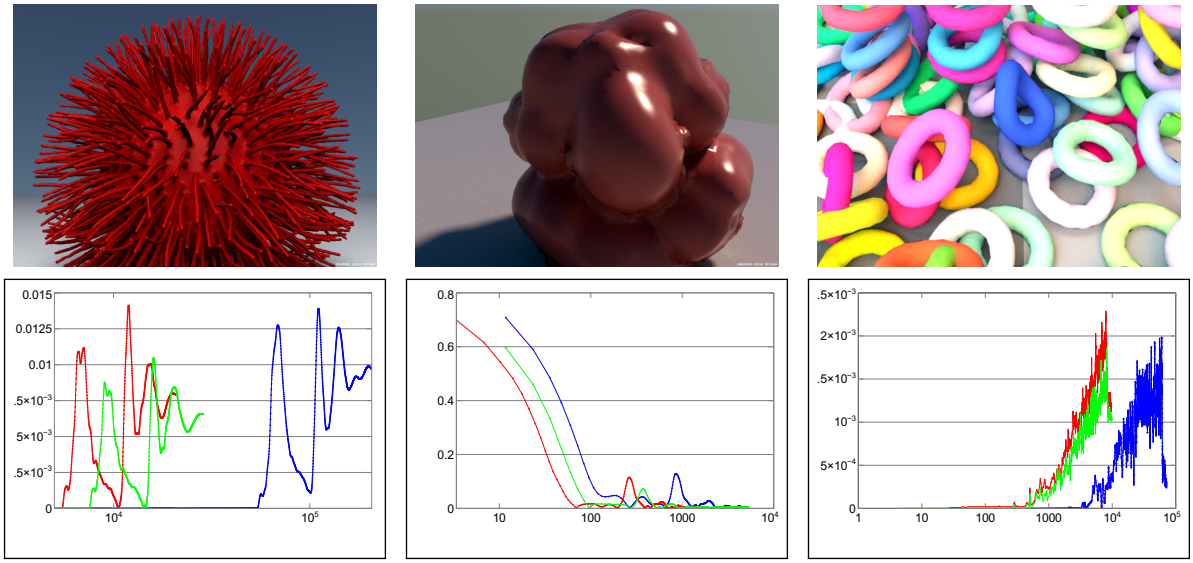


Figure 6: *Test cases (top row, left to right):* Squishy ball, Omotondo and Tori. *Convergence graphs (bottom row):* residual error on constraints versus time (in ms). *Color key:* Red, our approach. Green, Gauss-Seidel. Blue, averaged Jacobi.

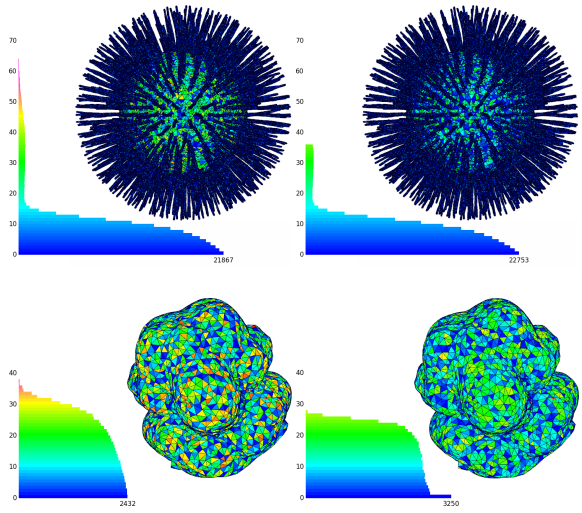


Figure 7: Partitioning of the SQUISHY BALL (~250K tetrahedral constraints) and OMOTONDO (~65K tetrahedral constraints) datasets. The side histograms show the number and size of partitions obtained with a Sequential Vertex Coloring (*left column*), and with our technique (*right column*).

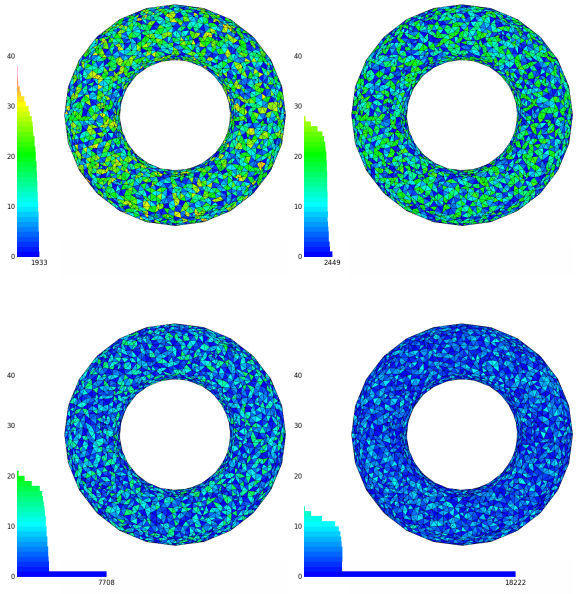


Figure 8: Partitioning of the BIG TORUS dataset (~52K tetrahedral constraints), using our technique with a decreasing number of colors.

iterations so that the residual error would be the same on the other solvers. We report the performances in Table 2. Timings were measured on a Nvidia Quadro K6000 and do not consider rendering and collision handling. As expected, the speed-up w.r.t. Sequential Vertex Coloring is governed by the number of kernel calls. Our method outperforms the parallel averaged Jacobi solver for two reasons. First, a Gauss-Seidel solver is faster to converge than Jacobi. Second, the Jacobi solver considers all of the constraints and particles for

each iteration, while a Gauss-Seidel solver considers just a subset of elements for each kernel call, which requires less computational load on the parallel processors of the GPU.

scene		its	c	kc/f	fps	s
squishy ball	GS	8	702K	880	205	5.9x
	GSPh	8	704K	448	405	11.6x
	J	110	702K	110	35	1x
omotondo	GS	16	144K	736	280	1.4x
	GSPh	16	155K	432	384	2x
	J	74	144K	74	195	1x
tori	GS	4	806K	172	225	8x
	GSPh	4	822K	148	240	8.6x
	J	43	806K	40	28	1x

Table 2: Performance results. GS: Gauss-Seidel, GSPh: our approach, J: averaged Jacobi, c: total number of constraints, kc/f: total number of kernel calls per frame, fps: frames per second, s: speed-up wrt Jacobi.

Collision detection is implemented using a sparse signed distance field, similarly to [MMCK14]. At the end of each iteration, the position of each particle is tested; if a particle is in an illegal position, then a collision constraint is generated on-the-fly, projecting the particle in a valid state. Each collision constraint affects exactly one particle, hence the corresponding constraint graph is disconnected and all the constraints can be solved in parallel with a single kernel call.

Hybrid Jacobi solver: A simpler variation of the phantom method , which we refer to as *hybrid Jacobi solver*, is described as follows. The set of constraints is first partitioned using Sequential Vertex Coloring; then during the animation phase the first $\hat{q}$ partitions are solved using parallel Gauss-Seidel and all the remaining constraints are solved with a single Jacobi iteration and constraint averaging. This method is easy to implement and avoids the computation and the insertion of the phantom particles and constraints. The convergence speed of the hybrid Jacobi solver is actually comparable with the phantom particle method (Fig. 9). However, using this approach may lead to spurious rigid modes in the final solution like in the simple case depicted in Fig. 10. This problem may depend on the order in which the constraints are solved. We observed that the magnitude of these undesirable rigid modes increases with the number of constraints (Fig. 10f).

6. Limitations and Future Work

Our technique allows to partition the set of constraints using an arbitrary number of colors $\hat{q}$, by injecting phantom particles and constraints in the topology of the constraint graph G. The constraints belonging to a partition are solved in parallel with a single kernel call, thus a low number of partitions lightens the performance overhead introduced by GPU kernel calls. However, decreasing the number of colors raises the relative error (Fig. 5), and may reduce the convergence speed of the solver. We found that usually a good trade-off between performance and visual quality is choosing $\hat{q} = \omega(G)/2$, where $\omega(G)$ is the size of the maximal cliques in G. Our method does not require any specific arrangements of constraints and it could be applied to parallelize any method employing a Gauss-Seidel solver.

In the future, we would like to explore parallel coloring algorithms which may allow the definition of partitions directly on the GPU. This would allow interactively change to the topology of the system enabling, for example, cutting and tearing of the constrained particle system.

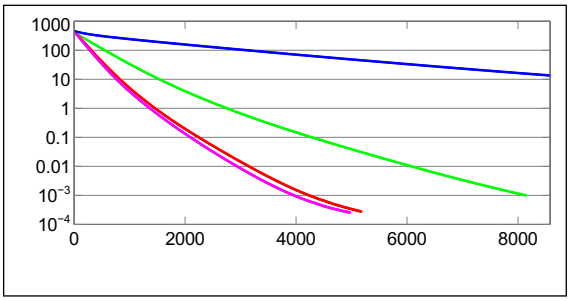


Figure 9: Residual error on constraints versus time. Comparison of the hybrid Jacobi solver with the different types of iterative solvers considered in this paper. The test case refers to the one in Fig. 10. *Color key:* Red, Gauss-Seidel with phantoms. Green, Gauss-Seidel. Blue, averaged Jacobi. Purple, hybrid Jacobi.

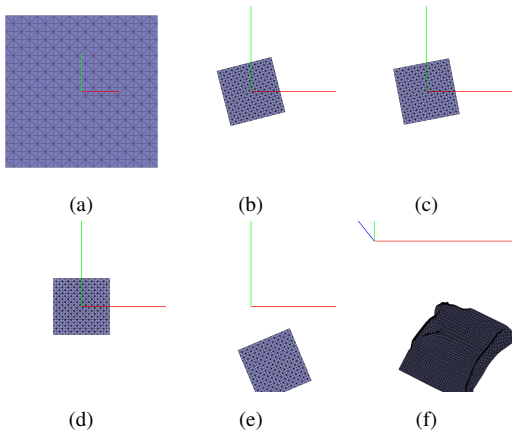


Figure 10: (a) Cloth composed of 400 particles and 1121 distance constraints, stretched to 16 times its rest area. The cloth is then left free to go back to its rest configuration without considering numerical integration. Solutions obtained with (b) Gauss-Seidel, (c) our approach, (d) averaged Jacobi and (e) hybrid Jacobi. The amount of spurious rigid modes introduced by hybrid Jacobi increases with the number of constraints (f).

7. Acknowledgements

The authors would like to thank the Visual Arena Lindholmen team, especially Thomas Hansson and Åsa Andblad, for providing the Nvidia Quadro K6000 graphics card, and Claudio Calabrese for assistance with editing. The test scenes in the accompanying video and several figures in the paper have been rendered using [Jak10]. The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement no. 290227.

References

[AFC*10] ALLARD J., FAURE F., COURTECUISSE H., FALIPOU F., DURIEZ C., KRY P. G.: Volume contact constraints at arbitrary resolution. In *ACM SIGGRAPH 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 82:1–82:10. 3

[BB08] BENDER J., BAYER D.: Parallel simulation of inextensible cloth. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (November 2008). 3

[BCK11] BAHI J. M., COUTURIER R., KHODJA L. Z.: Parallel gmres implementation for solving sparse linear systems on gpu clusters. In *Proceedings of the 19th High Performance Computing Symposia* (San Diego, CA, USA, 2011), HPC '11, Society for Computer Simulation International, pp. 12–19. 3

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (July 2002), 594–603. 3

[BK73] BRON C., KERBOSCH J.: Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM 16*, 9 (Sept. 1973), 575–577. 5, 6

[BMO*14] BENDER J., MÜLLER M., OTADUY M. A., TESCHNER M., MACKLIN M.: A survey on position-based simulation methods in computer graphics. *Computer Graphics Forum* (2014), 1–25. 1, 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. 3

[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

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

[DCB14] DEUL C., CHARRIER P., BENDER J.: Position-based rigid body dynamics. In *Proceedings of the 27th International Conference on Computer Animation and Social Agents* (May 2014). 1, 2

[ELS10] EPPSTEIN D., LÖFFLER M., STRASH D.: Listing all maximal cliques in sparse graphs in near-optimal time. In *Algorithms and Computation*, Cheong O., Chwa K.-Y., Park K., (Eds.), vol. 6506 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 403–414. 5

[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

[Jak10] JAKOB W.: Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 9

[JP94] JONES M. T., PLASSMANN P. E.: Scalable iterative solution of sparse linear systems. *Parallel Computing 20*, 5 (1994), 753 – 773. 1

[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

[Kur30] KURATOWSKI C.: Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae 15*, 1 (1930), 271–283. 4

[MÖ8] MÜLLER M.: Hierarchical position based dynamics. In *Virtual Reality Interactions and Physical Simulations (VRIPhys2008)* (Grenoble, November 2008). 3

[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

[MCKM14] MÜLLER M., CHENTANEZ N., KIM T.-Y., MACKLIN M.: Strain based dynamics. In *Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA)* (Copenhagen, July 21-23 2014). 2

[MHHR07] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr. 2007), 109–118. 1, 2, 3, 6

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

[MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified particle physics for real-time applications. *ACM Transactions on Graphics (SIGGRAPH 2014) 33*, 4 (2014). 1, 2, 8

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

[TBV12] TONGE R., BENEVOLENSKI F., VOROSHILOV A.: Mass splitting for jitter-free parallel rigid body simulation. *ACM Trans. Graph. 31*, 4 (July 2012), 105:1–105:8. 3

[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. 3