Vivace: a Practical Gauss-Seidel Method for Stable Soft Body Dynamics

Marco Fratarcangeli*
Chalmers University of Technology

Valentina Tibaldo
Fabio Pellacini
Sapienza University of Rome

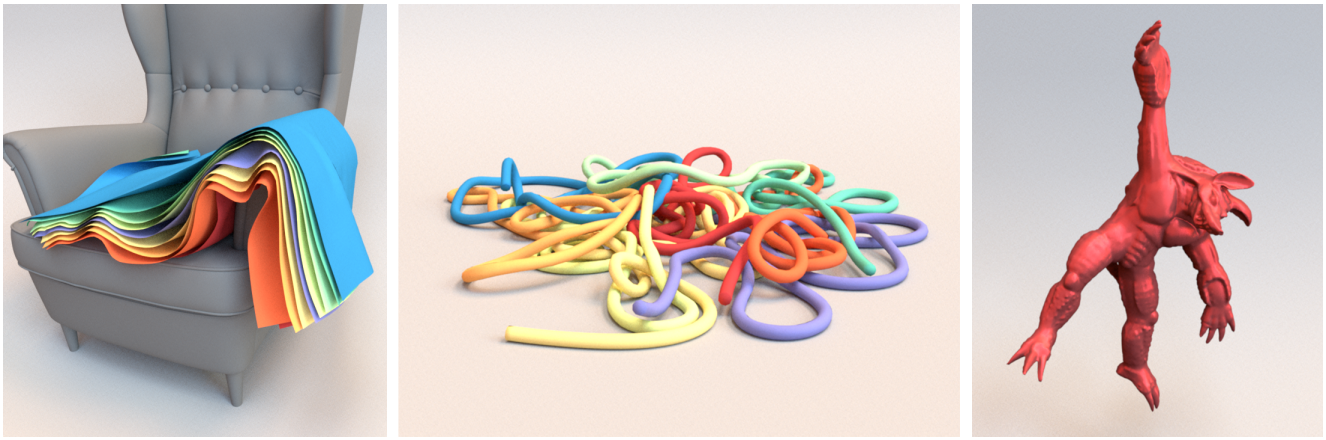

Figure 1: *Interactive animation of deformable bodies modeled using hundreds of thousands of linear constraints. For each frame, the dynamics is solved in a few milliseconds using our solver. Left. Self-colliding cloths (108K triangles, 324K constraints), 5 ms per frame. Middle. Self-colliding noodles (52K triangles, 156K constraints), 4 ms per frame. Right. Volumetric Armadillo hanging by one hand (55K tetrahedrons, 55K constraints), 15 ms per frame.*

Abstract

The solution of large sparse systems of linear constraints is at the base of most interactive solvers for physically-based animation of soft body dynamics. We focus on applications with hard and tight per-frame resource budgets, such as video games, where the solution of soft body dynamics needs to be computed in a few milliseconds. Linear iterative methods are preferred in these cases since they provide approximate solutions within a given error tolerance and in a short amount of time. We present a parallel randomized Gauss-Seidel method which can be effectively employed to enable the animation of 3D soft objects discretized as large and irregular triangular or tetrahedral meshes. At the beginning of each frame, we partition the set of equations governing the system using a randomized graph coloring algorithm. The unknowns in the equations belonging to the same partition are independent of each other. Then, all the equations belonging to the same partition are solved at the same time in parallel. Our algorithm runs completely on the GPU and can support changes in the constraints topology. We tested our method as a solver for soft body dynamics within the Projective Dynamics and Position Based Dynamics frameworks. We show how the algorithmic simplicity of this iterative strategy enables great numerical stability and fast convergence speed, which are essential features for physically based animations with fixed and small hard time budgets. Compared to the state of the art, we found our method to be faster and scale better while providing stabler solutions for very small time budgets.

Keywords: Multi-color Gauss-Seidel Method, Projective Dynamics, Position Based Dynamics, Parallel Computing

Concepts: •**Computing methodologies** → **Animation;**

*e-mail:marcof@chalmers.se

SA '16 Technical Papers, December 05 - 08, 2016, Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2982437

ACM Reference Format
Fratarcangeli, M., Tibaldo, V., Pellacini, F. 2016. Vivace: a Practical Gauss-Seidel Method for Stable Soft Body Dynamics. ACM Trans. Graph. 35, 6, Article 214 (November 2016), 9 pages.
DOI = 10.1145/2980179.2982437 http://doi.acm.org/10.1145/2980179.2982437.

1 Introduction

Many applications in Computer Graphics demand increasingly sophisticated models for animating physical phenomena. While these topics have been studied extensively, the simulation of complex systems at interactive rates is still an open problem. In this paper, we focus on real-time applications such as video games, with *fixed and small hard time budgets* available for physically-based animation and where responsiveness and stability are often more important than accuracy, as long as the results are believable. In these applications, iterative solvers have been employed effectively to provide fast, but approximated, solutions of the linear equations which govern the system. Their parallel implementation on commodity hardware, such as GPUs and multicore CPUs, is becoming increasingly important to speed up simulation.

To guarantee interactivity at 30 frames per second, the total amount of time available to update the scene is 33 ms, and only a fraction of this time can be devoted to advancing the dynamics of the objects. Often, the available time slice for physically based animation is as little as 5 ms. The accuracy of the solution of linear iterative solvers depends on the number of iterations; however only a small number of iterations can be accommodated in such a tight time budget. This forces the use of simple objects composed from few constraints whose dynamics can be solved in few iteration steps.

Some of the most popular iterative solvers in the Computer Graphics community are the Gauss-Seidel- and the Jacobi-based solvers.

These do not need computationally expensive inner products unlike the iterative conjugate gradient and GMRES, or direct methods like Cholesky decompositions. Furthermore, the Jacobi-based approaches are simple to parallelize, and thus suitable for modern GPUs or multi-core processors. It is well-known that the convergence speed of the Gauss-Seidel method is much faster than the Jacobi one under an equal number of iterations [Saad 2003; Thomaszewski et al. 2009]. However, the Gauss-Seidel method is an inherently serial algorithm, and thus it cannot be implemented easily on parallel architectures. There exists a significant body of literature on how to parallelize Gauss-Seidel, but in general these methods involve complex synchronization mechanisms and expensive communications between threads, which undermine their application in the context of interactive animation on commodity GPUs.

In this paper we present *Vivace*[1], a randomized parallel solver for sparse systems of linear constraints with the convergence speed of the Gauss-Seidel method. In *Vivace* we make use of a parallel randomized graph coloring algorithm [Grable and Panconesi 2000] for re-organizing the unknowns of the sparse linear system, such that no interdependent unknowns in a constraint share the same color. Then, unknowns that belong to the same color are solved in parallel with a single Gauss-Seidel step, leading to a significant acceleration of the convergence speed. The coloring algorithm itself is parallel and suitable for implementation on modern GPU architectures. While the use of graph coloring is known in our community, the main technical contribution of *Vivace* is the adoption of a randomized model and a careful study of its tradeoffs compared to existing models.

Vivace has three characteristics that make it particularly well suited for interactive soft-body dynamics. First, the randomized coloring algorithm produces partitions of similar size, which leads to balanced workloads, making sure that none of the computational units are overloaded or underutilized. Second, the coloring algorithm is fast enough to be rerun as needed, even for each frame. Thus the workload remains balanced even when the topology of the system changes, e.g. in case of constraints derived by collisions. Finally, the residual error in sequential Gauss-Seidel iterations depends on the order in which the equations are solved [Fratarcangeli and Pellacini 2015]. Randomizing this order, as is done in our algorithm, removes this error faster leading to a more accurate solution.

We demonstrate the benefits of this solver for the parallel implementation of Projective Dynamics [Bouaziz et al. 2014] and Position Based Dynamics [Müller et al. 2007; Macklin et al. 2014]. These two methods can be formulated as linear systems and as such can be solved efficiently with our method. Within these frameworks, we can comfortably solve hundreds of thousands of constraints in 5 ms including collisions on a modern GPU. Fig. 1 shows results from our solver. Compared to parallel Jacobi and Jacobi accelerated with Chebyshev polynomials [Wang 2015], we found *Vivace* to converge faster and remain significantly more stable for small time budgets. In conclusion, in our application domain, *Vivace* has the following benefits:

- **Convergence Speed.** *Vivace* is parallelizable, like the Jacobi-based methods, with the convergence rate of Gauss-Seidel ones.
- **Stability.** If the linear system is symmetric positive definite, e.g., in the case of mass-spring networks [Liu et al. 2013], $Vivace$ always converges even in the case of a small number of iterations used for each frame, allowing our solver to be used in case of small time budgets available for the physics computation.
- **Scalability.** The number of parallel steps to perform a full iteration is equal to the number of colors and does not depend on the number of constraints in the system. Furthermore it is considerably smaller than the maximal number of constraints influencing a single vertex in the polygonal mesh representing the animated object. In case of the triangular and tetrahedral meshes usually employed in Computer Graphics, this number is small enough to allow interactive animation of a very large of number constraints.

[1] *Vivace* is a word used for musical movements performed in a lively and brisk manner. Our solver speeds-up the computation of physics-based animation making it more "lively", hence the name.

2 Related Work

Linear Iterative Solvers Iterative solvers are the algorithms of choice for providing fast solutions of linear systems of equations found in interactive physical simulations. In these domains, the resulting linear systems, representing constrained systems, are large, irregular, and mostly sparse. This sparsity can be exploited to identify independent subsets of equations which can be solved in parallel. This strategy has been used to optimize the parallel implementation of popular Krylov subspace solvers on the GPU such as the preconditioned conjugate gradient (PCG) [Weber et al. 2013] and GMRES [Bahi et al. 2011].

Linear iterative solvers have slower convergence speeds than Krylov subspace methods, nonetheless they have been extensively employed because of their simplicity and faster computational speed. For example, they have been used for contact resolution of rigid body animation to avoid jitter artifacts [Bridson et al. 2002; Govindaraju et al. 2005; Tonge et al. 2012; Abel and Erleben 2015]. In [Allard et al. 2010], a parallel, coloring-based technique is presented for solving dense systems using the Gauss-Seidel method. In this method the thread synchronization relies on internal, potentially slow, atomic instructions.

The widely used Position Based Dynamics approach (PBD) [Bender et al. 2014] employs both the serial Gauss-Seidel method [Müller et al. 2007], and the parallel Jacobi method [Macklin and Müller 2013; Macklin et al. 2014] for the animation of rigid and deformable bodies, fluids, and their interaction (e.g., collisions). The Nucleus solver tackles the same problem, and is used within Maya [Stam 2009]. Projective Dynamics is an implicit integrator for interactive physics-based animation, recently introduced in [Bouaziz et al. 2014]. It is a generalization of [Liu et al. 2013] and [Müller et al. 2007], and shares some similarities with the optimization framework for geometric constraints proposed in [Bouaziz et al. 2012]. The main benefits of Projective Dynamics are its stability and the accuracy of the results. Unlike other implicit methods, it does not require computationally expensive differentiations, and it converges to an almost exact solution in few iterations.

Chebyshev semi-iterative method. Chebyshev polynomials can be used to accelerate the rate of convergence of iterative solvers [Golub and Van Loan 1996]. The lower and upper bounds of the eigenvalues of the matrix must be computed to find the optimal parameters of such polynomials. This is impractical because of the potentially large size of the linear system, which may also vary due to collisions or other topology changes. Recently, [Wang 2015] proposed an iterative method to quickly approximate the computation of Chebyshev polynomials, and used this to significantly accelerate the convergence speed of the parallel Jacobi method applied to both Projective Dynamics and Position Based Dynamics. However, using approximated values for the polynomial parameters may introduce instabilities if the number of iterations is too low.

3 Background

Vivace is particularly well-suited for solving the large sparse linear systems from the Projective Dynamics framework [Bouaziz et al. 2014]. In this section, we will quickly review this method, referring the reader to the original paper for further details. It is worth noticing here that Position Based Dynamics can also be cast in the Projective Dynamics framework.

Projective Dynamics. Projective Dynamics models the world as a set of constrained particles. The state of the system is defined by the position $\mathbf{q_i} \in \mathbb{R}^{3\times n}$ of the n particles, and their velocity $\mathbf{v_i} \in \mathbb{R}^{3\times n}$. The forces $\mathbf{f_{int}}$, which are generated internally by the constraints, are defined as the gradient of the sum of the energy functions W_i as $\mathbf{f_{int}} = -\sum_i \nabla_q W_i(\mathbf{q})$. Given the state $(\mathbf{q_i}, \mathbf{v_i})^{(k)}$ and the current external force $\mathbf{f_{ext}}$ at time t_k, the solver computes the internal forces $\mathbf{f_{int}}$ and the next state $(\mathbf{q_i}, \mathbf{v_i})^{(k+1)}$ with the implicit Euler scheme

$$\begin{cases} \mathbf{q}^{(k+1)} & = \mathbf{q}^{(k)} + h\mathbf{v}^{(k+1)} \\ \mathbf{v}^{(k+1)} & = \mathbf{v}^{(k)} + h\mathbf{M}^{-1}\left(\mathbf{f}_{ext} + \mathbf{f}_{int}\right) \end{cases}$$

where h is the time step, $\mathbf{M}$ is the mass matrix and the forces are evaluated in t_{k+1}. [Bouaziz et al. 2014] show that this is equivalent to solving a minimization problem that can be efficiently tackled with an iterative local/global alternation approach. For each iteration, they first project $\mathbf{q}$ on the nearest point $\mathbf{p}$ lying on the energy-free manifold defined by each constraint (the *local* step). Then, they minimize the distance of the current state $(\mathbf{q_i}, \mathbf{v_i})^{(k)}$ from the resulting local configurations $\mathbf{p_i}$ (the *global* step). The local step is inherently parallel because it is performed for each constraint independently from the other constraints. The main computational hurdle is the global step that is equivalent to solving the large sparse linear system $\mathbf{Yq} = \mathbf{b}$:

$$\underbrace{\left(\frac{\mathbf{M}}{h^2} + \sum_i \mathbf{S}_i^{\mathrm{T}}\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i\mathbf{S}_i\right)}_{\mathbf{Y}}\mathbf{q} = \underbrace{\frac{\mathbf{Ms}}{h^2} + \sum_i \mathbf{S}_i^{\mathrm{T}}\mathbf{A}_i^{\mathrm{T}}\mathbf{B}_i\mathbf{p}_i}_{\mathbf{b}} \quad (1)$$

$$\text{with } \mathbf{s}^{(k)} = \mathbf{q}^{(k)} + h\mathbf{v}^{(k)} + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$$

where $\mathbf{s}^{(k)}$ is the explicit integration of the state $\mathbf{q}^{(k)}$ ignoring the internal forces, $\mathbf{A}_i$ and $\mathbf{B}_i$ are constant matrices that define the constraint i (see [Liu et al. 2013; Bouaziz et al. 2014] for the definition of common constraints), and $\mathbf{S}_i$ is the selector matrix to select only the particles influenced by the constraint i.

Chebyshev semi-iterative approach. In [Bouaziz et al. 2014], $\mathbf{Y}$ is considered constant and, to solve the system in Equation 1, it is inverted just once during the initialization, while $\mathbf{b}$ is updated for each animation step. However, $\mathbf{Y}$ in general is not constant, in particular when new constraints are inserted into the system, e.g., constraints arising from collisions. In these cases, recomputing $\mathbf{Y^{-1}}$ for each frame is unfeasible given the large computational cost. [Wang 2015] introduced a method to solve $\mathbf{Yq} = \mathbf{b}$ using the Jacobi method accelerated with the Chebyshev semi-iterative approach [Golub and Van Loan 1996]. The proposed algorithm evaluates iteratively the approximated value of the spectral radius ρ for the matrix $\mathbf{Y}$, and uses it to compute an amplification factor ω. Then, for each iteration, the current solution is pushed towards the optimal solution using $\mathbf{q}^{(k+1)} = \omega_{k+1}(\hat{\mathbf{q}}^{(k+1)} - \mathbf{q}^{(k-1)}) + \mathbf{q}^{(k-1)}$ where $\hat{\mathbf{q}}^{(k+1)}$ is the result of the Jacobi method at the current iteration k. The resulting method is straightforward to implement in parallel, and leads to significant convergence speed-ups without adding noticeable computational penalties.

However, the value of the spectral radius ρ is approximated, and as such, it is not always possible to predict the actual acceleration factor for the convergence speed. More importantly, if the estimate of ρ is not precise enough, then divergence is likely to occur as we have observed in practice for small time steps.

Parallel Gauss-Seidel approach. We focus our effort on improving the solution of the linear system $\mathbf{Yq} = \mathbf{b}$ since this is the dominant cost in the Projective Dynamics framework. We seek a parallel solution well-suited for GPU evaluation, achieved by an iterative stable scheme that works well even for small per-frame time budgets. We base our work on the parallel Gauss-Seidel literature given the high convergence property of the base method. While Gauss-Seidel is an inherently serial algorithm, for sparse matrices it can be parallelized by dividing the set of equations in partitions, such that any pair of equations belonging to the same partition do not share any unknown. The set of unknowns in each partition can then be safely solved in parallel.

Graph Coloring. Of the many methods for partitioning a sparse linear system, graph coloring is the one mostly used in practice, both in the numerical analysis literature and in the distributed computing one [Saad 2003]. A graph G is built from the matrix $\mathbf{Y}$, where each vertex corresponds to an unknown, and two vertices are connected by an undirected edge if they belong to the same equation, i.e. they are related by a constraint.

By coloring G with a distance-1 algorithm with q colors, the unknowns assigned to the same color belong to independent equations by definition. Then, the standard lexicographic Gauss-Seidel method can be applied and, instead of solving the equations one after the other, all the equations belonging to the same partition can be solved together in one parallel step, shifting the complexity from $O(n)$ where n is the number of vertices, to $O(c)$, where c is the number of colors. We refer the reader to [Saad 2003] for a more comprehensive treatment. Since graph coloring belongs to the NP-hard class [Garey and Johnson 1979], the outstanding problem is to find an approximate and efficient algorithm to partition the vertices.

Parallel Graph Coloring. Graph coloring introduces a significant overhead when run each frame. To reduce the per-frame time budget, we investigate parallelizable graph coloring methods, a problem well-studied in the literature [Garey and Johnson 1979; Saad 2003]. Most parallel graph coloring methods are techniques that follow the structure of the method in [Luby 1985], where a solution is formed by iteratively determining an independent set I of vertices (such that no two vertices share a common edge), and color them in parallel. The colored vertices are removed from the graph and process is iterated untill all the vertices have been colored. Different methods are characterized by the manner in which they choose an independent set I. Thus, vertices belonging to the same independent set can be colored in parallel.

Practical Desiderata. When the solver is run on the GPU, q kernels are run sequentially. Therefore it is desirable to have as small a number of colors as possible, to lower the number of kernel executions. It is also desirable to haven roughly an equal number of nodes per color to ensure a balanced workflow. Furthermore, we want to be able to color the graph at each frame, to allow the solver to adapt to topology changes in the graph induces by time-varying constraints, such as collisions. Given these desiderata and that graph coloring is an NP-hard problem, we set to investigate which methods perform well in practice.

Algorithm 1 Simulation Step in *Vivace*

1: $\mathbf{q}^0 \leftarrow \mathbf{q}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$
2: *Graph coloring:* $V = \{\mathbf{q}_i, i = 1, \ldots, N\}$ is partitioned into p colors $C_1, \ldots, C_p$, such that $\forall (\mathbf{q}_i, \mathbf{q}_j) \in C_i$, $\mathbf{q}_i$ and $\mathbf{q}_j$ are not shared by any constraint
3: **for** $k = 0 \ldots K-1$ **do**
4: **for** each partition $C_i \subset V$ **do**
5: **for** each $\mathbf{q}_i \in C_i$ **do in parallel**
6: $\hat{\mathbf{q}}_i^{k+1} \leftarrow solve\left(\mathbf{q}_i^0, \mathbf{q}_i^k\right)$
7: $\mathbf{q}_i^{k+1} \leftarrow \omega\left(\hat{\mathbf{q}}_i^{k+1} - \mathbf{q}_i^{k-1}\right) + \mathbf{q}_i^{k-1}$
8: $\mathbf{q}_{t+1} \leftarrow \mathbf{q}^K$
9: $\mathbf{v}_{t+1} \leftarrow \left(\mathbf{q}_{t+1} - \mathbf{q}_t\right)/h$

4 *Vivace*: Parallel Gauss-Seidel by Randomized Graph Coloring

Alg. 1 shows pseudocode for the *Vivace* solver. The core idea of *Vivace* is the parallelization of lexicographic Gauss-Seidel by using graph coloring. Whenever the topology of the graph induced by the constraints network changes, then the coloring algorithm divides the set of vertices into independent partitions; each one corresponding to a color. Vertices belonging to the same partition are solved in parallel.

The simulation step starts by advancing the dynamics of the system with an explicit Euler step (step 1). Then, the coloring algorithm divides the set of vertices in independent partitions; each one corresponding to a color (step 2). The solver iterates K times over all the constraints. Each particle $\mathbf{q}_i$ belonging to partition C_j is processed in parallel. All the corrections induced by all the constraints sharing the particle are computed and summed together (steps 3-6). Then, a Successive Over-Relaxation (SOR) is applied to further accelerate the convergence speed (step 7). In all our tests, we have used $\omega = 1.9$. Higher values lead to spurious deformation modes or instabilities. Finally, in steps 8-9, the solution of the solver is assigned to the current position of the particles, and the velocity is updated accordingly.

4.1 Parallelization Strategy

In Alg. 2, we define a simple, randomized algorithm belonging to the class of Brooks-Vizing vertex coloring algorithms [Grable and Panconesi 2000]. These algorithms are appealing because they are simple to implement, fast, and use considerably fewer than Δ colors, where Δ is the maximal degree of the graph. As input, we consider graphs representing physically animated objects discretized as triangular or tetrahedral meshes. Such meshes are composed of hundreds of thousands of vertices connected by constraints (e.g., distance, bending and volume constraints), however Δ is low enough to enable real-time colorings.

The input of the algorithm is the undirected graph G corresponding to the matrix $\mathbf{Y}$ in Equation 1. Every vertex v is initially assigned a palette of available colors denoted as P_u. Colors are identified by consecutive natural numbers. V denotes the set of vertices, and U denotes the set of currently uncolored vertices.

During the *initialization* phase (steps 1-3), a list of Δ_v/s colors is given to the palette of each vertex v, where Δ_v is the degree of v and $s > 1$ is the palette *shrinking* factor, which is constant for the whole graph. Then, the actual coloring round procedure starts and is repeated until all the vertices have been colored. Each coloring round comprises three parallel steps. In the *tentative coloring* round (steps 5-6), a color $c(v)$ is randomly chosen among the available colors in the palette P_v, and assigned to v. Then, in *conflict resolution* (steps 7-12), each vertex checks that none of its neighbors has selected the same tentative color. If this occurs, the coloring of v is accepted and $c(v)$ is removed from the palette of the neighbors. In the *feed the hungry* phase (steps 14-16), a color is added to the palettes which have run out of colors. The maximal amount of colors allowed is $\Delta_v + 1$, but in our experiments we never reached this maximal threshold.

Algorithm 2 *Vivace* Graph Coloring Procedure [Grable and Panconesi 2000]

1: $U \leftarrow V$ ▷ Initialization
2: **for all** vertex $v \in U$ **do**
3: $P_v \leftarrow \{0, \ldots, \Delta_v/s\}$
4: **while** $|U| > 0$ **do**
5: **for all** vertices $v \in U$ **do** ▷ Tentative coloring
6: $c(v) \leftarrow$ random color in P_v
7: $I \leftarrow \varnothing$
8: **for all** vertices $v \in U$ **do** ▷ Conflict resolution
9: $S \leftarrow \{$colors of all the neighbors of $v\}$
10: **if** $c(v) \notin S$ **then**
11: $I \leftarrow I \cup \{v\}$
12: remove $c(v)$ from palette of neighbors of v
13: $U \leftarrow U - I$
14: **for all** vertices $v \in U$ **do** ▷ Feed the hungry
15: **if** $|P_v| = 0$ **then**
16: $P_v \leftarrow P_v \cup \{|P_v| + 1\}$

The effect of the shrinking factor is to reduce the number of colors; however increasing it too much leads to slower colorings without meaningful gains in terms of reducing the number of colors. In our case, we found that using the minimal degree of the graph as the value of the shrinking factor leads to the best colorings. To speed up the *conflict resolution* phase, we employed the *Hungarian heuristic* [Luby 1985]: in case of conflict, if the node has the higher index among its neighbors then the coloring is considered legitimate. We have found this strategy to greatly reduce the number of coloring rounds needed by the algorithm to color all the vertices in the graph.

4.2 Comparison with Other Graph Coloring

Other Graph Coloring Methods. In this section, we compare the randomized coloring used in *Vivace* with other parallel algorithms used in literature. [Luby 1985] proposed a Monte Carlo method to find a *maximal independent set (MIS)* in parallel, that is the largest possible independent set of vertices in the graph. In this approach, a random weight is assigned to each vertex. The weights are a random permutation of the integers $1, 2, \ldots, |V|$. Then, each maximal independent set is constructed in parallel by choosing vertices which are local maxima i.e., that have a weight greater than any other neighbors in $|V|$. Once a maximal independent set is identified, all the vertices belonging to such set are assigned the same color and removed from the graph. The procedure is repeated, using different colors for different maximal independent sets, until all the vertices are colored.

[Jones and Plassmann 1993] improved on Luby's algorithm. Instead of using the same color for each vertex in the same independent set, each vertex is colored individually with the smallest available color not already assigned to a neighboring vertex. This apparently simple improvement reduces both the number of colors and the number of rounds.

The Largest-Degree-First (LDF) algorithm is similar to the Jones-Plassmann approach but, instead of using random weights per ver-

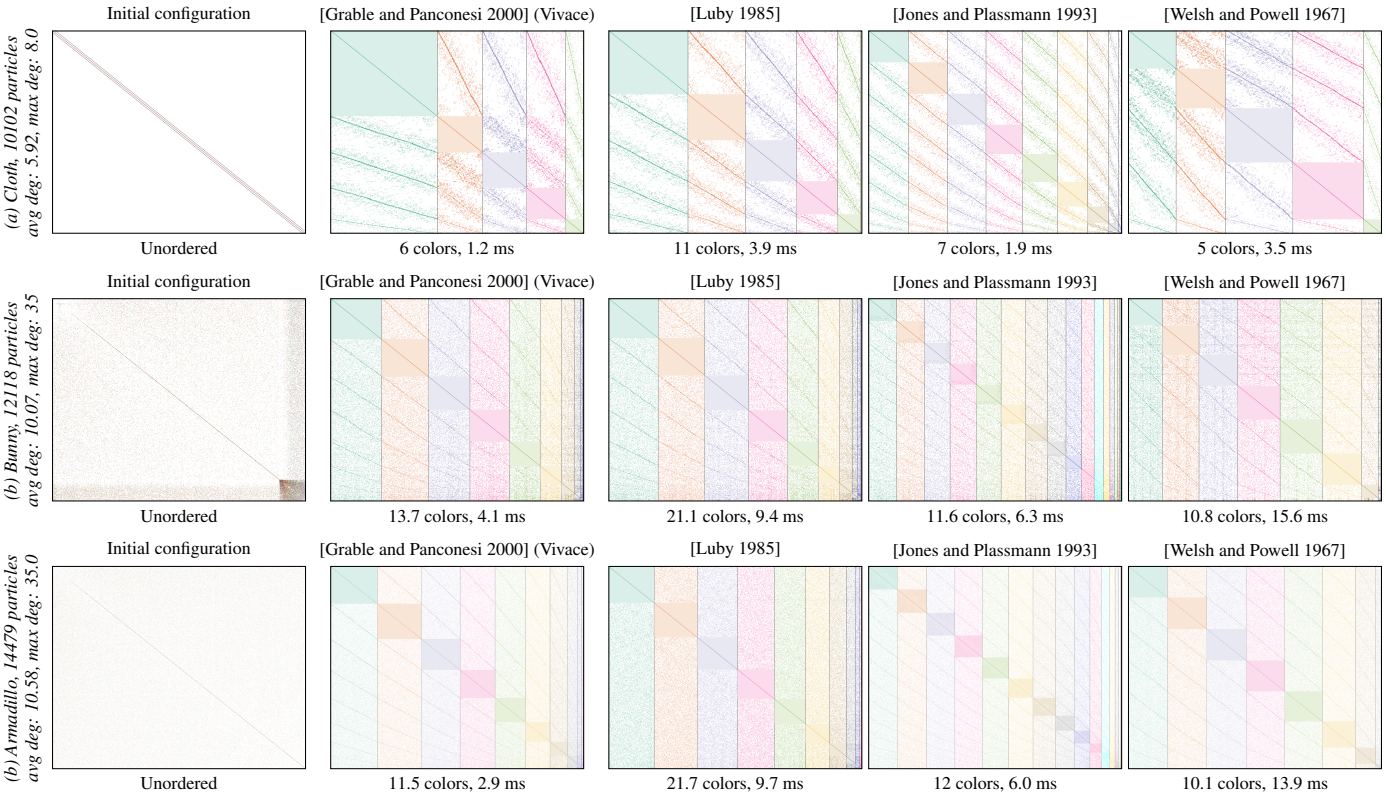

Figure 2: *Adjacency matrices corresponding to the constraint graphs. Leftmost column: initial configuration. Other columns: reordering induced by different coloring algorithms. Nodes belonging to the same color are independent from each other and can be solved in a single parallel step. The number of colors and computation time averaged over 30 frames is reported below each matrix.*

tex, the weight is defined as the current degree of the vertex [Welsh and Powell 1967]. After an independent set is defined, the colored vertices are removed from the graph and the degree of the remaining nodes is updated. A vertex with i colored neighbors requires at most $i + 1$ colors. The LDF algorithm keeps i as small as possible for each round, so that there is a better chance of keeping low the number of used colors. In fact, LDF uses fewer colors than the Jones-Plassmann algorithm; however the number of rounds to completely color the graph increases significantly.

In [Tonge et al. 2012], a greedy approach based on the Vizing's theorem [Vizing 1964] is used for reducing collision jittering of rigid bodies. Such coloring strategy leads to $2\Delta - 1$ colors, where Δ is the maximum degree of the graph, which in some cases can be high enough to compromise real-time performances (e.g., for the tetrahedral Armadillo used in our tests: $\Delta = 35$). In comparison, our approach extends the use of Gauss-Seidel not only to collision response but to the whole dynamics of deformable bodies, and it is faster because uses far less colors than Δ due to the shrinking factor s and the *feed the hungry* step (Alg. 2, steps 14-16). Graph coloring for collision handling has also been used in [Govindaraju et al. 2005], for meshes with rectangular connectivity, i.e. every vertex has four neighbors and every polygon is rectangular. Our approach can handle meshes without any restriction on connectivity.

Recently, [Naumov et al. 2015] presented a parallel coloring algorithm requiring a minimal amount of communication between the threads, making it particularly appealing for GPU implementations. This method performs better than the Jonas-Plassmann algorithm when incomplete colorings are acceptable (in this particular case if just 90% of the nodes need to be colored), making such an algorithm unsuitable for our purposes. Furthermore, this algorithm is based on custom hash functions which is not clear how to define in case of graphs derived from irregular meshes.

In [Fratarcangeli and Pellacini 2015], a method based on sequential coloring changes the topology of the graph in the initialization phase, in order to use a minimal number of colors. This method leads to balanced partitions but, being executed just at the beginning of the animation, it is not suitable for matrices varying over time. In contrast, our approach focuses on interactively changing sparse systems which require low overhead colorings.

Other approaches, such as the Smallest-Degree-First [Matula and Beck 1983], focus on how to provide better coloring rather than maximizing speed. And other good sequential algorithms, namely the Saturation-Degree-Ordering [Brélaz 1979] and Incidence-Degree-Ordering [Coleman and Moré 1983] algorithms, are not suitable for parallelization, and we do not consider them here.

Performance Comparison. We compared the randomized coloring algorithm employed in *Vivace* with other parallel coloring algorithms, namely Luby [Luby 1985], Jones-Plassmann (JP) [Jones and Plassmann 1993] and Largest-Degree-First (LDF) [Welsh and Powell 1967]. Unlike the existing literature, where the Brooks-Vizing colorings are tested on triangle- or square-free graphs, we applied the algorithms on graphs derived from irregular geometric models: 1) a triangulated cloth using a spring constraint for each edge, and 2) the Stanford Bunny and Armadillo using tetrahedral constraints for volume preservation. From each model, we generated different meshes with an increasing number of triangles and tetrahedrons, respectively.

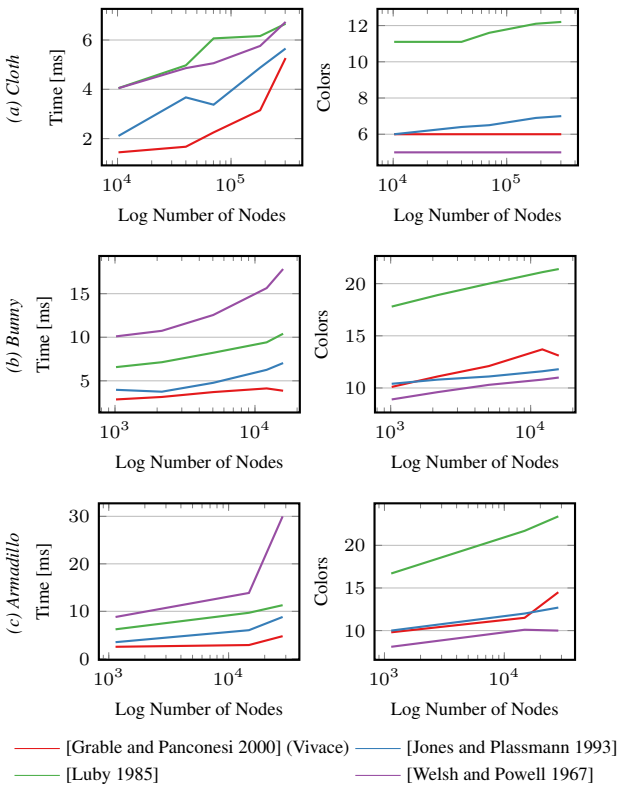

Figure 3: *Comparison of different coloring algorithms w.r.t. time to complete (left column), and number of used colors (right column).*

The re-ordering induced by the coloring is depicted in the adjacency matrices in Fig. 2, which correspond to the matrix $\mathbf{Y}$ in Equation 1, and represent the topology of the constraints in the mesh before and after the coloring. The rows and columns represent the indices of the nodes. An element of the matrix m_{ij} is not empty if the corresponding nodes i and j are shared by a constraint. A single row of the matrix represents a constraint to be solved. The initial configuration is shown in the leftmost column, labeled as *unordered*.

By reordering the indices of the nodes in the graph according to their color, the elements are "pushed" away from the diagonal. Instead of solving for one unknown after the other as in the lexicographic Gauss-Seidel, all the unknowns belonging to the same color can be solved simultaneously in parallel.

We quantitatively compare algorithms with respect to the total time for coloring and number of colors (Fig. 3). We remind the reader that we seek a solution with the smallest number of colors, to increase parallelism, and the smallest computation time. As expected, colorings using [Welsh and Powell 1967] employ the smallest amount of colors but are 10-20 times slower than the other algorithms. *In general, our tests demonstrate that the random coloring employed in* Vivace *outperforms all the other algorithms in speed while using approximately the same number of colors*.

5 Results and Discussions

In this section, we provide a qualitative and quantitative assessment of *Vivace*. All algorithms have been fully implemented on the GPU using CUDA/c++, and run on an NVIDIA GeForce GTX 970.

Performance and stability We have tested *Vivace* with respect to *stability, convergence speed and scalability*, and compared it with other two iterative solvers: the parallel Jacobi method [Macklin et al. 2014], and parallel Jacobi accelerated with the Chebyshev semi-iterative method, as presented in [Wang 2015]. For each experiment, we tested all the solvers with different time budgets, while the same conditions were used, i.e. same time step $h = 33$ms, same external forces, and same damping. The supplemental video shows the performed experiments.

The plots in Fig. 4 report the residual error over time for a triangulated cloth composed of 10K vertices and 20K triangles modeled with a spring constraint for each edge [Liu et al. 2013], and a hinge-edge constraint for each edge shared by two triangles [Bergou et al. 2006]. The accuracy of all the solvers increases with the number of iterations; however, the number of iterations must be relatively small in order to satisfy the time budgets. When the time budget is 4 or 8 ms, the residual error of the Jacobi solver is too big to be acceptable (blue curve), while the solver proposed in [Wang 2015] diverges to an unstable state (green curve). One of the main practical benefits of our solver is its capability to provide stable solutions despite the low number of iterations, while being at least one order of magnitude more accurate than the other solvers (red curve). Fig. 6 compares the corresponding visual results.

The plots in Fig. 5 show the residual error over time for a volumetric armadillo composed of 55K tetrahedral constraints preserving volume and shape. Even in this case, the convergence speed of Jacobi and the stability of [Wang 2015] is not sufficient to provide reliable results in case of small time budgets (Fig. 7). However, interestingly, in case of larger time steps (25 ms) the convergence rate of *Vivace* is the same as that the Chebyshev solver.

In principle, the acceleration technique presented in [Wang 2015] is orthogonal to our approach and can be used to speed-up the Gauss-Seidel solver; however, tuning its acceleration factor ω for optimal results is not trivial. For this reason, we based our comparison on the source code available from the authors of [Wang 2015], which uses Jacobi + approximated Chebyshev. Its stability can be increased by reducing ω, but this would also slow down the convergence speed.

Complex scenarios. The stability and the speed of *Vivace* enabled us to model more complex scenarios as depicted in Figures 8 and 9, where we demonstrate our solver's scalability by handling objects composed of a large number of interacting constraints. Here we modeled the system using Position-Based Dynamics. In Fig. 8 we animated a stack of deformable noodles, composed of 30K vertices, 52K triangles and 150K constraints falling on the floor and colliding with each other. In Fig. 9, we represent a heap of cloths falling on a static armchair. Each cloth is composed of 36K constraints, for a total number of 324K constraints. The armchair is modeled as a Signed Distance Field which is used for the collision tests. The scene is updated interactively at 30 fps, so that it is possible to pinch and drag the cloths. In these cases the topology of the graph changes due to collisions, so recoloring is necessary and *Vivace* handles this very well. We are not aware of any other existing solver able to handle such complex test cases within the considered time budgets.

Limitations *Vivace* has three main limitations. First, only a restricted number of iterations can be accommodated in the considered time budgets. Thus, *Vivace* can deliver only approximate solutions which, in some cases, may lead to artifacts; for example, in Fig. 6 the cloth flexes excessively near the anchor points due to the theoretical limits of the convergence speed of relaxation methods. The perceived animation is still visually acceptable in the domain of real-time applications (e.g., games). In case more accu-

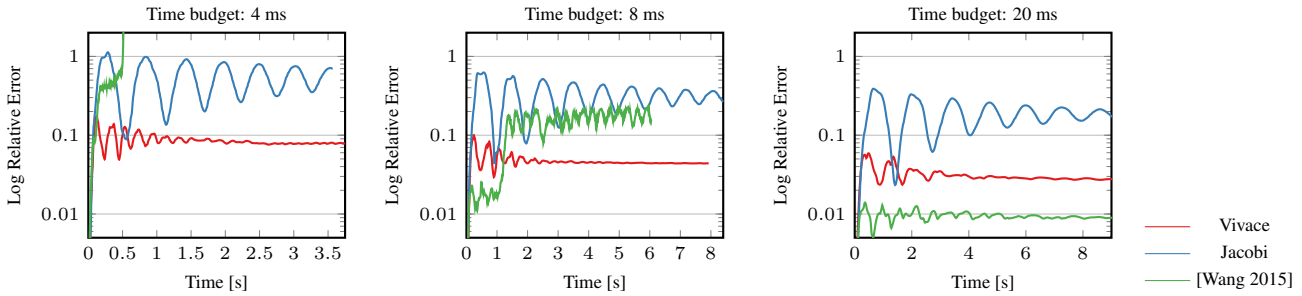

Figure 4: *Relative error vs. time in the Tablecloth animation in the accompanying video (triangulated model composed by 10K vertices, 20K triangles, 30K springs and 30K hinge-edge constraints; time step $h = 33$ms.). The convergence speed of our method (red curve) is compared with the Jacobi method (blue curve) and [Wang 2015] (green) using different time budgets.*

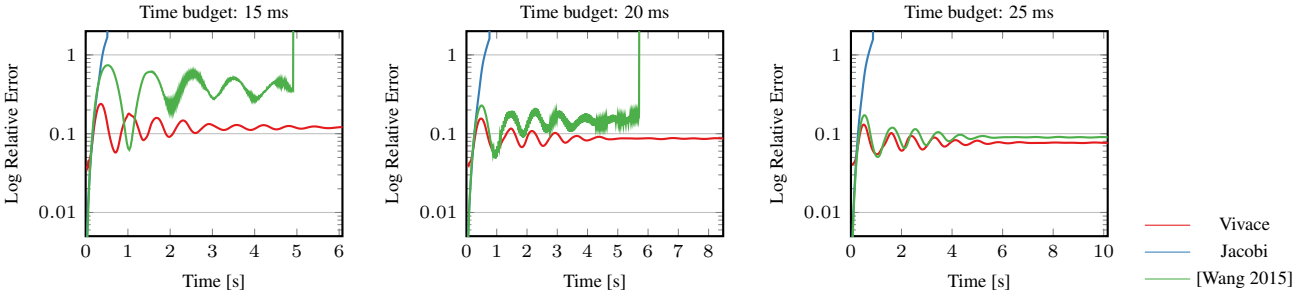

Figure 5: *Relative error vs. time in the Armadillo animation in the accompanying video (volumetric model composed by 10K vertices, 55K tetrahedral constraints; time step $h = 33$ms.). The convergence speed of our method (red curve) is compared with the Jacobi method (blue curve) and [Wang 2015] (green) using different time budgets.*

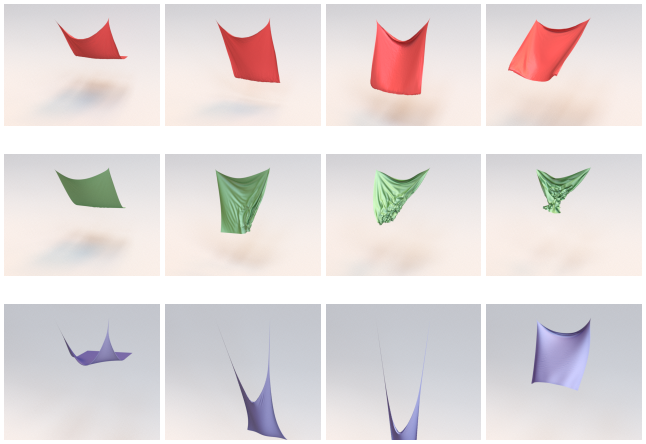

Figure 6: *Still frames from the animation corresponding to the plot in Fig. 4b. Solver time budget: 8 ms. Upper row: our method, middle row: Jacobi + Chebyshev [Wang 2015], bottom row: Jacobi.*

Figure 7: *Still frames from the animation corresponding to the plot in Fig. 5a. Solver time budget: 15 ms. Upper row: our method, bottom row: Jacobi + Chebyshev [Wang 2015].*

racy is needed, it may be interesting to use our solver as a preconditioner in a parallel algebraic multigrid system (e.g., [Tamstorf et al. 2015]), for further accelerating the overall convergence speed. Second, while we have tested *Vivace* with deformable bodies, where the topology of the system changes due to collisions, we did not yet investigate its performance in handling fluids, where the constrained system exhibits dramatic changes of topology for each frame. Third, the actual implementation of *Vivace* is more complex with respect to Jacobi-based solvers because it requires the graph coloring step. This additional complexity is mitigated by the simplicity of the randomized coloring algorithm.

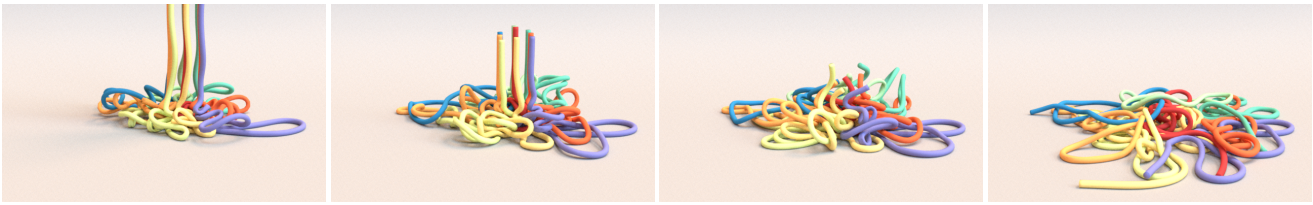

Figure 8: *Self-colliding noodles falling on the floor. 30K vertices, 52K elements, 150K constraints. Solver time budget including collision handling: 15ms per frame.*

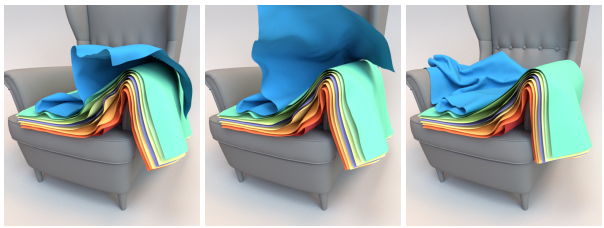

Figure 9: *Pinching and dragging a cloth on top of a heap. Solver time budget including collision handling: 15ms per frame.*

6 Conclusions

We demonstrated a solver for soft body dynamics that can handle hundreds of thousands of constraints in a few milliseconds without noticeable visual artifacts. Being based on a randomized algorithm, it minimizes the communication among threads while retaining the necessary simplicity in implementation. These features make our solver particularly suitable for high-performance computations on modern graphics hardware, with a focus on real-time applications in entertainment and industrial design.

Acknowledgements

We would like to thank IKEA Communication AB, in particular Martin Enthed and Anton Berg, for fundings and equipment, and anonymous reviewers for their valuable comments. Marco Fratarcangeli was supported in part by the Swedish Research Council (project number: 2015-05345). Fabio Pellacini and Valentina Tibaldo are partially supported by generous grants from Sapienza, MIUR and Intel.

References

ABEL, S., AND ERLEBEN, K. 2015. *Numerical methods for linear complementarity problems in physics-based animation: synthesis lectures on computer graphics and animation*. Morgan & Claypool Publishers, United States.

ALLARD, J., FAURE, F., COURTECUISSE, H., FALIPOU, F., DURIEZ, C., AND KRY, P. G. 2010. Volume contact constraints at arbitrary resolution. *ACM Trans. Graph. 29*, 4 (July), 82:1–82:10.

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

BENDER, J., MÜLLER, M., OTADUY, M. A., TESCHNER, M., AND MACKLIN, M. 2014. A survey on position-based simulation methods in computer graphics. *Computer Graphics Forum 33*, 6, 228–251.

BERGOU, M., WARDETZKY, M., HARMON, D., ZORIN, D., AND GRINSPUN, E. 2006. A quadratic bending model for inextensible surfaces. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '06, 227–230.

BOUAZIZ, S., DEUSS, M., SCHWARTZBURG, Y., WEISE, T., AND PAULY, M. 2012. Shape-up: Shaping discrete geometry with projections. *Comput. Graph. Forum 31*, 5 (Aug.), 1657–1667.

BOUAZIZ, S., MARTIN, S., LIU, T., KAVAN, L., AND PAULY, M. 2014. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. 33*, 4 (July), 154:1–154:11.

BRÉLAZ, D. 1979. New methods to color the vertices of a graph. *Commun. ACM 22*, 4 (Apr.), 251–256.

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

COLEMAN, T. F., AND MORÉ, J. J. 1983. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis 20*, 1, 187–209.

FRATARCANGELI, M., AND PELLACINI, F. 2015. Scalable Partitioning for Parallel Position Based Dynamics. *Computer Graphics Forum (Eurographics) 34*, 2 (May), 405–413.

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

GOLUB, G. H., AND VAN LOAN, C. F. 1996. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

GOVINDARAJU, N. K., KNOTT, D., JAIN, N., KABUL, I., TAMSTORF, R., GAYLE, R., LIN, M. C., AND MANOCHA, D. 2005. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24*, 3 (July), 991–999.

GRABLE, D. A., AND PANCONESI, A. 2000. Fast distributed algorithms for brooksvizing colorings. *Journal of Algorithms 37*, 1, 85 – 120.

JONES, M. T., AND PLASSMANN, P. E. 1993. A parallel graph coloring heuristic. *SIAM J. Sci. Comput. 14*, 3 (May), 654–669.

LIU, T., BARGTEIL, A. W., O'BRIEN, J. F., AND KAVAN, L. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph. 32*, 6 (Nov.), 214:1–214:7.

LUBY, M. 1985. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, STOC '85, 1–10.

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

MACKLIN, M., MÜLLER, M., CHENTANEZ, N., AND KIM, T.-Y. 2014. Unified particle physics for real-time applications. *ACM Trans. Graph. 33*, 4 (July), 153:1–153:12.

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

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.

NAUMOV, M., CASTONGUAY, P., AND COHEN, J. 2015. Parallel Graph Coloring with Applications to the Incomplete-LU Factorization on the GPU. Tech report, NVIDIA.

SAAD, Y. 2003. *Iterative Methods for Sparse Linear Systems*, 2nd ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

STAM, J. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference on*, 1–11.

TAMSTORF, R., JONES, T., AND MCCORMICK, S. F. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph. 34*, 6 (Oct.), 245:1–245:13.

THOMASZEWSKI, B., PABST, S., AND STRAER, W. 2009. Continuum-based strain limiting. *Computer Graphics Forum 28*, 2, 569–576.

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

VIZING, V. G. 1964. On an estimate of the chromatic class of a *p*-graph. (russian). *Diskret. Analiz. 3*, 25–30.

WANG, H. 2015. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph. 34*, 6 (Oct.), 246:1–246:9.

WEBER, D., BENDER, J., SCHNOES, M., STORK, A., AND FELLNER, D. 2013. Efficient gpu data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1, 16–26.

WELSH, D. J. A., AND POWELL, M. B. 1967. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal 10*, 1, 85–86.