Smoothed Aggregation Multigrid for Cloth Simulation

Rasmus Tamstorf
Walt Disney Animation Studios

Toby Jones
Walt Disney Animation Studios

Stephen F. McCormick
University of Colorado, Boulder

Abstract

Existing multigrid methods for cloth simulation are based on geometric multigrid. While good results have been reported, geometric methods are problematic for unstructured grids, widely varying material properties, and varying anisotropies, and they often have difficulty handling constraints arising from collisions. This paper applies the algebraic multigrid method known as *smoothed aggregation* to cloth simulation. This method is agnostic to the underlying tessellation, which can even vary over time, and it only requires the user to provide a fine-level mesh. To handle contact constraints efficiently, a *prefiltered preconditioned conjugate gradient* method is introduced. For highly efficient preconditioners, like the ones proposed here, prefiltering is essential, but, even for simple preconditioners, prefiltering provides significant benefits in the presence of many constraints. Numerical tests of the new approach on a range of examples confirm $6-8\times$ speedups on a fully dressed character with 371k vertices, and even larger speedups on synthetic examples.

CR Categories: I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: cloth simulation, smoothed aggregation, algebraic multigrid, equality constrained optimization

Figure 1: *The method presented in this paper provides an* $8\times$ *speedup for a walk cycle animation of this character and* $6\times$ *for a run cycle animation. These numbers are compared to a block diagonally preconditioned CG method. The garments consist of a combined* $371,064$ *vertices.*

1 Introduction

Multigrid methods are well-known and theoretically optimal in that they promise to deliver a solution to a wide variety of discrete equations with compute time proportional to the number of unknowns. However, multigrid algorithms that obtain full computational efficiency can be difficult to design for new applications, especially when constraints are introduced.

The goal of this paper is the acceleration of high-end cloth simulation, with applications ranging from feature film production to virtual try-on of garments in e-commerce. The methods developed here also apply to the broader problem of thin-shell simulation that has many applications in engineering. Common to all of the applications is that higher resolution leads to higher fidelity, but also often prohibitive computation times for conventional methods.

Existing multigrid approaches to these types of problems typically require structured meshes and have difficulty with collisions. Furthermore, their convergence rates and scaling properties have often not attained full computational efficiency. This paper investigates several possible causes and potential remedies for these difficulties. One difficulty inherent to elasticity is that the different displacement variables are interconnected in the sense that changes in the smooth components of one variable strongly affect others. In fact, neither *geometric multigrid methods (GMG)* [Brandt 1977] nor *algebraic multigrid methods (AMG)* [Brandt et al. 1985], when designed in a standard way, work optimally when the variables are strongly coupled. We elaborate on this below with a small didactic example.

Our primary contribution here is to apply a multigrid preconditioner based on *smoothed aggregation (SA)* [Míka and Vaněk 1992] to cloth simulation. SA is designed to handle coupling between variables and is superior to simple diagonally preconditioned conjugate gradients even for relatively small problems. SA is also purely algebraic, so it is agnostic to the choice of input meshes and relieves the user of explicitly generating hierarchies or choosing meshes such that they are amenable to coarsening. In fact, SA works with completely irregular and potentially even (time) adaptive meshes. It is also agnostic to the choice of triangles or quads for the underlying discretization. To our knowledge, this work is the first time that algebraic multigrid methods have been applied to cloth simulation. The initial focus here is on the formulation introduced by [Baraff and Witkin 1998], but it has important implications on other formulations and it provides a pathway to treating other models.

While many existing literature sources address multigrid fundamentals, we are not aware of any that provide the comprehensive but condensed focus needed to address the several challenges present in cloth simulation. To this end, in sections 4 and 5, we present a concise discourse on the fundamentals of multigrid methodology and the development of smoothed aggregation, including some theoretical foundations and heuristics,

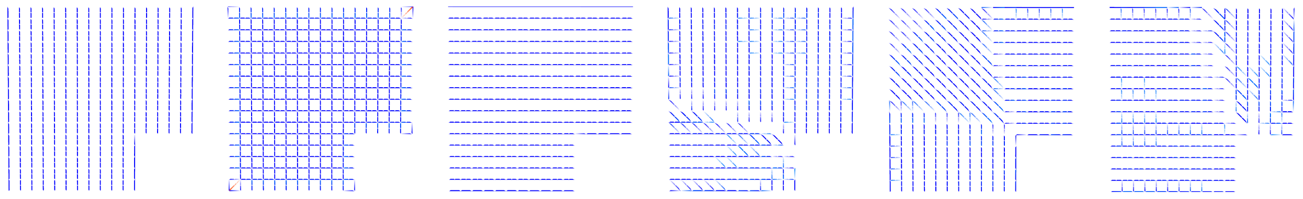


Figure 2: *Visualization of the strength of connection within a matrix for each of the three variables, (x, y, z), at two frames of a simulation. Strong negative off-diagonal connections between vertices are shown in UV space as blue lines. The two red lines in the second-to-left image indicate strong positive off-diagonal connections. For clarity, "weak" connections are omitted. In the undeformed state (three leftmost images), the x and z components are each anisotropic but in different directions. In the deformed state (three rightmost images), different directions of anisotropy appear even within a single variable.*

known and new, that guide the development of effective multigrid algorithms for novel applications. This discussion is new in that it focuses on the connection between existing multigrid theory and the development of effective algebraic multigrid solvers applied to thin shells, especially with collisions. Our experiences with the failure of standard AMG and SA for these systems led us to reconsider basic multigrid concepts and principles, and this discussion will hopefully act as a guide to anyone who might attempt to apply multigrid to other related applications. The theory is further elaborated upon in the supplemental material.

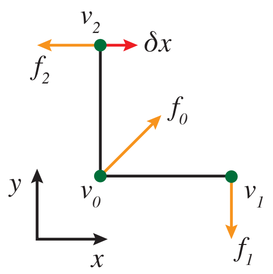
A critical component in cloth simulation is collision handling. We do not propose any new methods for collision detection or response, but we do introduce a novel way of handling the filtering step originally introduced by [Baraff and Witkin 1998]. The new method, called *Prefiltered Preconditioned CG (PPCG)*, is one of the principal contributions of this paper. PPCG is essential for maintaining the advantage of the SA preconditioner in the presence of collision constraints, but it is also advantageous when using a simple diagonal preconditioner. We introduce PPCG for treating collisions in section 8.

In section 11, we document the effectiveness of SA and PPCG with a comprehensive set of tests that show significant computation speedup over conventional methods for a wide range of cloth simulations of interest. The method has been implemented in a production-quality solver, and can be implemented in other systems simply by replacing the innermost linear solver.

2 Challenges for multigrid

Partial differential equations (PDEs) for thin shell elasticity with large deformations are complicated, yet most cloth models are approximations of these equations. Even the highly simplified models of a planar elastic membrane undergoing small deformations result in biharmonic equations. For most methods, including multigrid, these equations are substantially more complicated to find solutions for than Poisson-like problems, which is what is often considered in the multigrid literature.

Two challenges that affect our work directly are anisotropy and strong coupling between variables. Anisotropy is usually associated with constitutive models and, in fact, cloth is an example of an anisotropic material. However, in the context of multigrid methods, anisotropy simply means that certain variables are connected more strongly than others in the underlying matrix, and that there is a pattern to the directionality of these strong connections. The notion of a “strong connection” here corresponds to a large off-diagonal value of the associated matrix element relative to the diagonal. What is important to note is that this type of anisotropy occurs even with an isotropic homogeneous elastic material. This behavior is due in part to the Poisson effect. As an illustration, consider the simulation of an L-shaped piece of cloth, where the boundary along the cut-out corner is held fixed while the rest falls under gravity. The corresponding strength of connections in the associated stiffness matrix exibits not only distinct anisotropies, but also directions of anisotropy that vary in space and time (see figure 2). Standard methods like semicoarsening or line relaxation used with geometric multigrid are thus ineffective for this problem.



The second challenge related to strong coupling between variables can easily be seen by considering the simple 2D example shown in the inset figure to the right. In this example, two edges are bent from their rest configuration, which generates the set of bending forces labeled f_0, f_1, and f_2. If we apply a displacement, δx, to the top vertex, v_2, then all the bend forces increase in magnitude. In particular, this means that a change in the x coordinate of v_2 leads to a change in the x coordinate of f_2, but it also leads to a change of the y coordinate of f_1. Because the two changes have the same magnitude, the x and y variables are interpreted to be strongly coupled from a multigrid point of view.

The theory in section 4 suggests that this strong cross-variable coupling would lead to poor performance of *standard* AMG. We confirmed this numerically by running several tests for the L-shaped problem mentioned above. Let the number of vertices in a simulation be n. Standard AMG then performed well for each of the three $n \times n$ blocks associated with the individual unknowns when the coupling between unknowns was deleted in the matrix. However, for the full matrix, its convergence rate was poor even for small problems, and degraded further as the problem sizes increased. Standard AMG simply was unable to provide a significant improvement over diagonally preconditioned conjugate gradients (PCG).

3 Related work

Given the wide range of applications and the cost of the simulations, it is not surprising that multigrid methods have been studied in the graphics community for cloth simulation and, in the wider multigrid community, for the more generic linear elasticity and thin-shell problems.

Some of the earliest work on multigrid for thin shells appears in [Fish et al. 1996] where they investigated an unstructured multigrid method for thin shells amounting to an extension of geometric multigrid that coarsened with knowledge of the problem. However, they acknowledge that their method has limitations in that it does not address higher-order interpolation and it was never tested on large-scale problems.

More recently, Gee *et al.* [Gee et al. 2005] addressed more complicated shell models using a finite element discretization. They used an aggregation-based approach that has many similarities to an SA hierarchy. However, for their method, they treat the shell as a (thin) 3d solid unlike the typical 2d manifold approach used for cloth. To avoid severe ill-conditioning and to obtain convergence rates independent of conditioning, they apply "scaled director preconditioning". This also allows them to model varying thickness across the shell. In their follow-up work [Gee and Tuminaro 2006], the focus is on using a nonlinear solver, and adaptive SA is used to precondition the linearizations. In our experience, adaptive SA is currently too expensive for typical cloth problems, although this may change in the future with improvements in the adaptive algorithm and demands for much more computationally intense simulations.

Related research within the graphics community has been focused primarily on applying various types of geometric multigrid to cloth simulation. Oh *et al.* [2008] implemented a GMG method that uses CG for the smoother on each level, with a focus on preserving energy and mass at all levels of the hierarchy. Linear interpolation is used between levels, the level hierarchy is attained through regular subdivision, and constraints are only treated on the fine grid. This produced a significant speedup for their simulations but failed to show linear scaling in the size of the problem, and the performance deteriorated in the presence of constraints. Lee *et al.*[2010] used a multi-resolution approach that focused on using adaptive meshing to only place subdivisions where needed to provide acceleration, compared to a GMG with regular subdivision. It is not clear how to use this approach in a multilevel fashion, nor how it addresses the difficulties arising from the PDE and collisions. Jeon *et al.* [2013] extended the work in [Oh et al. 2008] to handle "hard" constraints, as presented in [Baraff and Witkin 1998], by converting them to soft constraints to avoid the challenges of coarsening them. They use GMG as a direct solver, with PCG as their smoother for restriction and GS as their smoother for prolongation, but require an expensive V(5,10) cycle.

One of the exceptions to the use of geometric multigrid in the graphics literature is an AMG-like algorithm developed by Krishnan *et al.* [2013]. Their focus is on discrete Poisson equations written in terms of *M-matrices*, which (among other properties) have only nonpositive off-diagonal entries. Basically, their approach is to modify the original matrix by first selecting unknowns that represent the coarse grid and then eliminating interconnections between the remaining fine-grid-only unknowns. This elimination is accomplished in a *sparsification/compensation* process that is analogous to AMG's so-called *stencil collapsing* procedure. The difference is that, while AMG eliminates these connections to determine interpolation, their goal is to produce a modified matrix (to be used as a preconditioner) that naturally preserves M-matrix character on coarser levels. Their method is shown to be superior to two algebraic multigrid techniques that were modified to similarly preserve M-matrix character, so it is no doubt important in some applications. However, their results show a loss of efficiency compared to more standard algebraic multigrid methods. Our aim here is to develop an AMG approach that is not restricted to M-matrix discretizations.

The simulator we use for our experiments is based on a combination of the methods presented by Baraff and Witkin [1998] and Bridson et al. [2002]. As stated above we do not propose any new contributions related to the material model or the way collisions are handled. In particular, we use the material model from [Baraff and Witkin 1998], and the basic contact handling is as described in section 6 of that paper (including approximate handling of friction). Since that does not guarantee that the simulation is free of continuous time collisions at the end of a time step, we follow our linear solve by one or more loops of Bridson's method [Bridson et al. 2002, section 7.4]. If this still fails to resolve all collisions, then we apply the fail-safe in [Harmon et al. 2008]. Only the contributions from Baraff & Witkin's paper actually affect what goes into the linear system that our solver sees. This includes constraints for cloth/object collisions and repulsion forces for cloth/cloth collisions (the latter being akin to the repulsion forces in section 7.2 of [Bridson et al. 2002] but included into the implicit solve as originally suggested by [Baraff and Witkin 1998]). While multigrid methods can be used as stand-alone solvers, we found it more effective in our system to use multigrid as a preconditioner within CG. The only change we made to our system is therefore within the CG method. All other features and limitations remain the same.

This work should apply without modifications to other cloth simulators such as the one in [Narain et al. 2012], which re-tessellates the geometry adaptively throughout the simulation.

4 Multigrid fundamentals

Conventional multigrid methods, whether geometric or algebraic, tend to perform poorly for thin-shell applications. To understand the difficulties inherent in this application and develop a more advanced algebraic multigrid method with improved performance, we carefully consider the basic multigrid principles and the nature of the equations that we are treating. We document the concepts that originally guided us through this process by providing a basic overview of the multigrid methodology tailored to thin-shell equations. This is covered in the next two sections, with additional background available in the supplemental material. The aim in this development is to give insight into the steps needed to design effective multigrid solvers for thin-shell equations and provide a template for their development for future applications. We draw on relevant existing multigrid concepts and principles available in the basic tutorial presented in [Briggs et al. 2000], the additional material developed in [Trottenberg et al. 2000], and relevant theory given in [McCormick 1984] and [Vassilevski 2008]. We also include several new insights that we gained in this development effort.

Multigrid relies on two complementary processes: smoothing (or relaxation) to reduce "oscillatory" errors associated with the upper part of the spectrum, and coarsening (or coarse-grid correction) to reduce "smooth" errors associated with the lower part of the spectrum. Many choices exist for smoothing, but the various multigrid methods are distinguished mostly in how they coarsen.

Geometric multigrid methods rely on the ability to coarsen a grid geometrically and to (explicitly or implicitly) define discretizations on coarser grids, as well as interpolation operators between the grids. Unfortunately, geometric multigrid methods can be difficult to develop for problems with unstructured grids, complex geometries, and widely varying coefficients and anisotropies. As a convenient alternative to GMG methods, AMG and its cousin SA were developed to provide automatic processes for coarsening based solely on the target matrix. AMG coarsens a grid *algebraically* based on the relative size of the entries of the matrix to determine strong connections, thereby forming a hierarchy of grids from the finest, on which the original problem is defined, down to the coarsest, which typically consists of just a few degrees of freedom. The AMG coarsening process produces coarse grids whose degrees of freedom are subsets of those on the fine grid (represented by identity rows in the interpolation matrix). Thus, while AMG is an algebraic approach, a geometric representation of coarse-grid nodes in the continuum is still easily determined.

For linear finite element discretizations of Poisson's equation on regular 2D grids, the parameters for AMG can be selected to produce the usual geometric coarsening with linear interpolation. In this case, the coarse-grid matrix is essentially what FE would produce by re-discretization on the coarse grid. AMG and GMG solvers would then have similar interpolation, restriction, and coarse-grid components. It is thus often safe to make assumptions about the convergence of a standard GMG approach by looking at the convergence of an AMG implementation. Yet AMG can automatically produce effective coarse levels for many problems that do not lend themselves to geometric approaches.

Smoothed aggregation is an advanced aggregation-based method founded on algebraic multigrid principles. When coarsening the grid, these methods form agglomerates (by grouping fine-grid nodes) that each become a node of the coarse grid. The points that go into agglomerates are also formed based on relative strength between elements of the matrix. However, for standard SA, coarse nodes do not correspond to single fine-grid nodes. So, for vertex-centered discretizations, it is generally not possible to assign geometric meaning to the coarse grids that SA produces, especially for systems of PDEs. Smoothed aggregation also tends to coarsen more aggressively than AMG and GMG, so the coarse matrices and interpolation operators generally must work harder to obtain efficiency comparable to that of AMG and GMG.

Once a coarse grid has been selected, an interpolation operator, $\boldsymbol{P}$, that maps coarse-grid to finer-grid functions must be formed. Selection of the interpolation operator can be guided by classical multigrid theory. Solving a fine-grid matrix equation of the form $\boldsymbol{Au} = \boldsymbol{f}$, where $\boldsymbol{A}$ is symmetric positive definite, is equivalent to minimizing the discrete energy functional given by $F(\boldsymbol{v}) \equiv \langle \boldsymbol{Av}, \boldsymbol{v}\rangle - 2\langle \boldsymbol{v}, \boldsymbol{f}\rangle$, where $\boldsymbol{v}$ is an approximation of the exact solution $\boldsymbol{u}$. Simple algebra shows that the best coarse-grid correction to a fixed approximation, $\boldsymbol{v}$, in the sense of minimizing $F(\boldsymbol{v} - \boldsymbol{Pv}^c)$, is expressed by

$$\boldsymbol{v} \leftarrow \boldsymbol{v} - \boldsymbol{P}\left(\boldsymbol{P}^T\boldsymbol{AP}\right)^{-1}\boldsymbol{P}^T\left(\boldsymbol{Av} - \boldsymbol{f}\right).$$

This objective leads to the two so-called *variational conditions* that are used in forming AMG/SA hierarchies: restriction from the fine to the coarse grid is the transpose of interpolation, and the coarse-grid matrix is given by the "Galerkin" product $\boldsymbol{A}^c \equiv \boldsymbol{P}^T\boldsymbol{AP}$. GMG by comparison is often done by re-discretizing the problem on the coarse levels to obtain $\boldsymbol{A}^c$, and then computing

$$\boldsymbol{v} \leftarrow \boldsymbol{v} - \boldsymbol{P}(\boldsymbol{A}^c)^{-1}\boldsymbol{P}^T\left(\boldsymbol{Av} - \boldsymbol{f}\right).$$

To construct an effective interpolation operator, we must understand why conventional iterative methods tend to work well for a couple of iterations, but then soon stall. This phenomena is due to the error quickly becoming smooth and difficult to reduce. But this *smoothing property* is precisely all that we ask of relaxation. The basic idea for multigrid is that smooth error can then be eliminated efficiently on coarse levels. For coarsening to work well this way, interpolation must adequately approximate smooth errors, as articulated in the following property [Vassilevski 2008]. Here and in what follows, $\|\cdot\|$ and $\|\cdot\|_A \equiv \|\boldsymbol{A}^{\frac{1}{2}}\cdot\|$ denote respective Euclidean and "energy" norms.

Definition 4.1 *The* **Strong Approximation Property (SAP)** *holds if a fine-grid error, $\boldsymbol{e}$, can be approximated in the energy norm by the coarse grid with accuracy that depends on the Euclidean norm of its residual, $\boldsymbol{Ae}$:*

$$\min_{\boldsymbol{u}^c} \|\boldsymbol{e} - \boldsymbol{Pu}^c\|_A^2 \leq \frac{C}{\|\boldsymbol{A}\|}\langle \boldsymbol{Ae}, \boldsymbol{Ae}\rangle.$$

Let any vector $\boldsymbol{e}$ be called a *near-kernel component* if either $\frac{\langle \boldsymbol{e}, \boldsymbol{Ae}\rangle}{\|\boldsymbol{A}\|}$ is small compared to $\|\boldsymbol{e}\|^2 = \langle \boldsymbol{e}, \boldsymbol{e}\rangle$ or $\frac{\langle \boldsymbol{Ae}, \boldsymbol{Ae}\rangle}{\|\boldsymbol{A}\|}$ is small compared to $\|\boldsymbol{e}\|_A^2 = \langle \boldsymbol{e}, \boldsymbol{Ae}\rangle$. The SAP is sufficient to guarantee fast convergence of the most cost-effective form of a multigrid solver called the *V-cycle*. (See the supplemental material for a detailed description.) A consequence of this guarantee is that the SAP creates a clear goal for the interpolation operator in that it must, at the very least, adequately approximate near-kernel components. This is an essential observation in what follows. However, the global nature of the energy norm that the SAP is based on makes it difficult to use as a design tool. It is more common in practice to develop schemes with estimates based on the Euclidean norm, as the following weaker property provides.

Definition 4.2 *The* **Weak Approximation Property (WAP)** *holds if a fine-grid error, $\boldsymbol{e}$, can be approximated in the Euclidean norm by the coarse grid with accuracy that depends on its energy norm:*

$$\min_{\boldsymbol{u}^c} \|\boldsymbol{e} - \boldsymbol{Pu}^c\|^2 \leq \frac{C}{\|\boldsymbol{A}\|}\langle \boldsymbol{Ae}, \boldsymbol{e}\rangle.$$

Developing schemes based on the WAP is easier because estimates involving the Euclidean norm can be made locally in neighborhoods of a few elements or nodes. This locality provides the basis for the classical development of both AMG and SA. Unlike the SAP, however, the WAP itself is not enough to ensure V-cycle convergence, which is why interpolation-operator smoothing (see the next section) is used in aggregation-based methods.

Even with the above requirements there are many ways to construct the interpolation operator, $\boldsymbol{P}$. Standard GMG and AMG are examples of so-called *unknown-based* multigrid methods, where the degrees-of-freedom (DOF) are coarsened and interpolated separately. To illustrate this approach, note that a cloth simulation typically involves three *displacement* unknowns. The resulting matrix, $\boldsymbol{A}$, can therefore be written in the form

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{1,3} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} & \boldsymbol{A}_{2,3} \\ \boldsymbol{A}_{3,1} & \boldsymbol{A}_{3,2} & \boldsymbol{A}_{3,3} \end{bmatrix}, \tag{1}$$

where each block is of size $n \times n$, with n the number of nodes used in the discretization. When using an unknown-based method, interpolation and coarsening are constructed based on the block diagonals to form the full interpolation operator:

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{P}_{1,1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P}_{2,2} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}_{3,3} \end{bmatrix}. \tag{2}$$

The coarse-grid matrix is then formed using the Galerkin operator $\boldsymbol{P}^T\boldsymbol{AP}$. This works well when the PDE is dominated by the connections within the unknowns. However, for problems like elasticity with strong off-diagonal connections, unknown-based approaches can suffer poor convergence and scaling.

Another choice for applying multigrid to systems of PDEs is the *nodal* approach, where the fine-grid matrix is structured in blocks of the same size as the number of unknowns in the PDE at each grid point. This approach complicates the determination of strength between nodes and, in turn, coarsening, but it provides a bridge to smoothed aggregation. Instead of coarsening a grid by identifying a set of nodes that becomes the coarse grid, SA partitions the grid into aggregates that are strongly interconnected. Akin to how finite elements use local functions, SA then assigns each aggregate a basis of local vectors that can be linearly combined with each other and bases of the other aggregates to adequately (according to the WAP) approximate all fine-grid smooth error. The coefficients in these linear combinations constitute the coarse-grid unknowns, and the vectors themselves represent an approximation of the near-kernel for the coarse problem. This form gives SA the natural ability to interpolate across unknowns, and it has the added benefit of being able to fit a much richer set of errors.

In summary, multigrid methodology involves approximating the algebraically smooth error left by relaxation by forming a coarse grid and an interpolation operator from it to the fine grid that adequately represents these errors in accordance with the WAP. SA, in particular, approximates algebraically smooth errors by choosing aggregates of nodes that are connected strongly enough to enable one or a few basis elements to represent these errors locally, with the WAP guiding the choice of the basis elements that properly approximate these smooth errors.

5 Smoothed Aggregation

Construction in SA of a hierarchy of matrices and the corresponding interpolation operators between successive levels proceeds in three stages: selection of aggregates ($\mathscr{A}_i$ on level $i = 1, 2, \ldots, m$ from fine to coarse grids), forming interpolation operators ($\boldsymbol{P}_i$), and then forming coarse-grid operators ($\boldsymbol{A}_{i+1} = \boldsymbol{P}_i^T \boldsymbol{A}_i \boldsymbol{P}_i$). Since SA is a nodal approach, on any given level i of the hierarchy, $\boldsymbol{A}_i$ is assumed to have n_i nodes, each corresponding to $b_i \times b_i$ blocks. At the finest level, b_1 is the number of unknowns in the original PDE (i.e., 3 displacements in our case). The dimensions of $\boldsymbol{A}_i$ is block $n_i \times n_i$ when considered nodally, and $(n_i \cdot b_i) \times (n_i \cdot b_i)$ when all unknowns are considered.

Smoothed aggregation assumes that we are given a set of near-kernel components that constitute the columns of a matrix, $\boldsymbol{K}$. This near-kernel matrix is used below to construct bases for the agglomerates. $\boldsymbol{K}$ must have the property that any near-kernel component $\boldsymbol{e}$ must be adequately approximated (according to the WAP) in each aggregate by a linear combination of the columns of $\boldsymbol{K}$ restricted to that aggregate. For scalar Poisson equations, one near-kernel component (typically the constant vector) is usually enough to obtain good performance. For 2D linear elasticity, three components (typically a constant for each of the two displacement unknowns and a rotation) are usually needed. In section 6, we return to the problem of choosing $\boldsymbol{K}$.

The first step in aggregating nodes is to form a strength-of-connection (SOC) matrix, $\boldsymbol{S}$, which serves multiple purposes. Its primary function is to provide a structure where “strength" between any pair of nodes in the “grid" is stored. This is used to decide which nodes are strongly interconnected so that they can be grouped together into small local aggregates. Another purpose of $\boldsymbol{S}$ is to treat a problem caused by anisotropy. The problem arises because the interpolation should be in the direction of strength, but the smoothing that is used to improve the interpolation operator can smear in the direction of weakness. $\boldsymbol{S}$ can be used to identify the potential for this smearing and filter smoothing by eliminating the weak connections in the matrix.

The SOC matrix is usually chosen with a sparsity pattern that is a subset of the original nodal matrix. This can be advantageous in the implementation because the size of the necessary memory allocation is known at the beginning of the construction process. In general, $\boldsymbol{S}$ is not needed after setup, so it can be discarded after that phase. Usually, the strength between nodes is defined in a way that allows $\boldsymbol{S}$ to be symmetric, and the cost of assembling $\boldsymbol{S}$ is reduced to constructing the upper (or lower) triangular part of the matrix. Classically, the strength between nodes is defined as

$$s_{ij} = \begin{cases} 1, & i = j, \\ 1, & \rho\left(\boldsymbol{A}_{ii}^{-1/2}\boldsymbol{A}_{ij}\boldsymbol{A}_{jj}^{-1/2}\right) > \theta \cdot \rho_{i,\max}, \\ 0, & \text{otherwise,} \end{cases}$$

where $\rho(\cdot)$ denotes the spectral radius of a matrix, $\rho_{i,\max} = \max_{j \neq i} \rho\left(\boldsymbol{A}_{ii}^{-1/2}\boldsymbol{A}_{ij}\boldsymbol{A}_{jj}^{-1/2}\right)$, and $\theta \in (0,1)$. s_{ij} effectively determines strength relative to other off-diagonals in row i. Also, $\boldsymbol{A}_{ij}$ here refers to the block (of size $b_1 \times b_1$) associated with a nonzero in the matrix between nodes i and j.

Based on $\boldsymbol{S}$, define the set $\mathscr{S}$ to be the *special nodes*, by which we mean those that are not strongly connected to any other node or that correspond to a row in the matrix that is very diagonally dominant. Relaxation leaves little to no error at the special nodes, so they need not be interpolated to or from coarser levels and are therefore gathered into one aggregate that is not affected by interpolation from coarser levels.

Next, to facilitate description of the aggregation process, let the set of fine-level nodes be represented by their indices, $\mathscr{D}_1 = \{1, 2, \ldots, n_1\}$. The next phase then constructs a fine-level partition, $\{\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_{n_2}\}$, of $\mathscr{D}_1 \setminus \mathscr{S}$ into disjoint aggregates:

$$\mathscr{D}_1 \setminus \mathscr{S} = \bigcup_{i=1}^{n_2} \mathscr{A}_i, \quad \mathscr{A}_i \cap \mathscr{A}_j = \emptyset, \quad \forall i \neq j.$$

Each aggregate here forms a node on the coarse level. Given the set of special nodes, this phase is accomplished by two passes through the nodes. In the first pass, an initial set of aggregates is formed and, in the second, all unaggregated nodes are assigned to existing aggregates.

Algorithm 1 Form aggregates, pass 1

```
 1: input Set of nodes, $\mathscr{D}$, and SOC matrix, $\boldsymbol{S}$.
 2: $\mathscr{R} = \mathscr{D}$
 3: $k = 0$
 4: for $i = 1, \ldots, n_1$ do
 5:     Form $\mathscr{N}_i = \{j : s_{ij} = 1, i \neq j\}$
 6:     if $\mathscr{N}_i \cap \mathscr{R} = \mathscr{N}_i$ then
 7:         $\mathscr{A}_k = \mathscr{N}_i \cup \{i\}$
 8:         $k = k + 1$
 9:         $\mathscr{R} = \mathscr{R} \setminus (\mathscr{N}_i \cup \{i\})$
10:     end if
11: end for
12: $n_2 = k$
13: return Aggregates, $\mathscr{A}_1, \ldots, \mathscr{A}_{n_2}$, and still un-aggregated nodes
    $\mathscr{R}$.
```

The goal of the first pass is to create a set of aggregates from a maximally independent set of strongly connected nodes. One way to do this is outlined here. Each node is examined once in turn, in any logical order. If none of the current node's strongly connected neighbors are in an aggregate, then they join the current node to form a new aggregate. Otherwise, the current node is left alone and the next node is examined similarly. More specifically, let $\mathscr{R}$ be the set of node indices that are not currently assigned to an aggregate. Initially, $\mathscr{R} = \mathscr{D}_1 \setminus \mathscr{S}$. Let $\mathscr{N}_i = \{j : s_{ij} = 1, i \neq j\}$ be the set of points that are strongly connected to point i. The first pass is outlined in Algorithm 1.

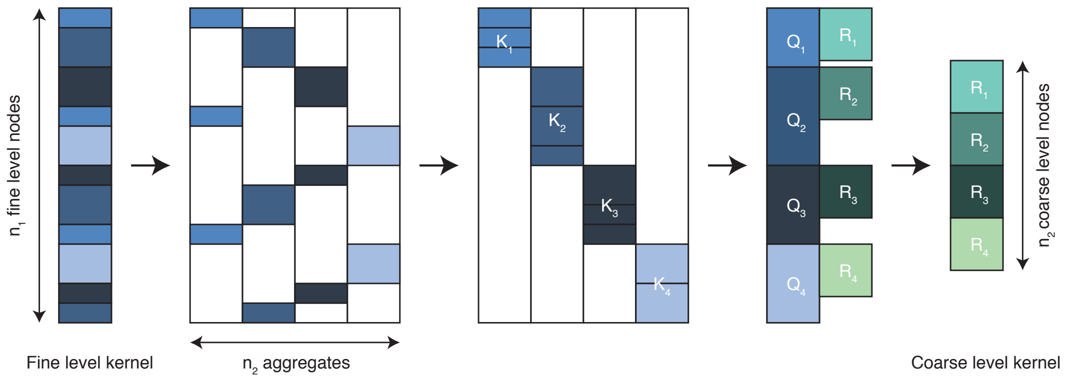


Figure 3: *To form the kernel for a coarse level, we start with the kernel for the fine level (far left), where the rows have been tagged according to which aggregate the corresponding node belongs to. All the rows with identical tags are then combined to form the* local *kernels (middle), and a thin QR decomposition is applied to each local kernel. The resulting* $\boldsymbol{Q}$ *matrices form the building blocks for the tentative interpolation operator,* $\hat{\boldsymbol{P}}$*, while the resulting* $\boldsymbol{R}$ *matrices form the building blocks for the coarse-level kernel (far right).* $\hat{\boldsymbol{P}}$ *is obtained by replacing each* $\boldsymbol{K}_i$ *matrix with the corresponding* $\boldsymbol{Q}_i$ *matrix and then permuting back to the original row ordering (second from the left).*

After the initial set of aggregates is formed, a subset of unaggregated nodes, $\hat{\mathscr{R}}$, remains. The goal now is to assign the nodes in $\hat{\mathscr{R}}$ to aggregates in the list $\mathscr{A}_1,\ldots,\mathscr{A}_{n_2}$. This assignment can be done by looping over each aggregate and assigning to it all nodes left in $\hat{\mathscr{R}}$ that are strongly connected to one of its nodes. (An alternative is to loop over each node in $\hat{\mathscr{R}}$, assigning it to the aggregate that it is most strongly connected to.) All non-special nodes are strongly connected to at least one node, so this step ensures that they will all be aggregated. Each aggregate is represented by a node on the coarse level, so that level will have size n_2. This step is outlined in Algorithm 2.

Algorithm 2 Form aggregates, pass 2

```
1: input Aggregates, $\mathscr{A}_1,\ldots,\mathscr{A}_{n_2}$, and still un-aggregated nodes $\hat{\mathscr{R}}$.
2: for $i=1,\ldots,n_2$ do
3:     Let $m_i$ be the number of elements in $\mathscr{A}_i$
4:     for $j=1,\ldots,m_i$ do
5:         Form $\mathscr{N}_j$
6:         Let $P_j=\mathscr{N}_j\cap\hat{\mathscr{R}}$
7:         for $k\in P_j$ do
8:             $\mathscr{A}_i=\mathscr{A}_i\cup\{k\}$
9:             $\hat{\mathscr{R}}=\hat{\mathscr{R}}\setminus\{k\}$
10:        end for
11:    end for
12: end for
13: return An independent set containing all nodes, $\mathscr{A}_1,\ldots,\mathscr{A}_{n_2}$.
```

Interpolation is constructed in two basic steps. The first involves choosing a *tentative* interpolation operator, $\hat{\boldsymbol{P}}$, while the second step consists of smoothing $\hat{\boldsymbol{P}}$. The tentative interpolation operator is chosen such that the set of near-kernel components, $\boldsymbol{K}$, is in the range of $\hat{\boldsymbol{P}}$, and $\hat{\boldsymbol{P}}$ does not connect neighboring aggregates, so $\hat{\boldsymbol{P}}^T\hat{\boldsymbol{P}}=\boldsymbol{I}$. The construction of $\hat{\boldsymbol{P}}$ is illustrated in Figure 3. Conceptually, assume that the nodes are ordered so that they are contiguous within each aggregate and in correspondence to the aggregate ordering. (This ordering is not necessary in practice, but used here simply to facilitate the discussion.) The near kernel can then be decomposed into n_2 blocks denoted by $\boldsymbol{K}_1,\boldsymbol{K}_2,\ldots,\boldsymbol{K}_{n_2}$ and written in block form as $\boldsymbol{K}=\begin{bmatrix}\boldsymbol{K}_1^T & \boldsymbol{K}_2^T & \cdots & \boldsymbol{K}_{n2}^T\end{bmatrix}^T$. This representation means that the number of rows of $\boldsymbol{K}_i$ equals the number of nodes in $\mathscr{A}_i$ times the nodal block size for the current level, and the number of columns equals the number, κ, of near-kernel components.

A local QR of each block can now be formed: $\boldsymbol{K}_i=\boldsymbol{Q}_i\boldsymbol{R}_i, \boldsymbol{Q}_i^T\boldsymbol{Q}_i=\boldsymbol{I}$, which yields the matrices $\boldsymbol{Q}_1,\ldots,\boldsymbol{Q}_{n2}$ and $\boldsymbol{R}_1,\ldots,\boldsymbol{R}_{n2}$. The columns of $\boldsymbol{Q}_i$ form a local basis spanning the near kernel in $\mathscr{A}_i$. Given this decomposition, the tentative interpolation operator, $\hat{\boldsymbol{P}}$, is formed via

$$\hat{\boldsymbol{P}}=\begin{bmatrix}\boldsymbol{Q}_1 & 0 & 0 & 0\\ 0 & \boldsymbol{Q}_2 & 0 & 0\\ \vdots & \ddots & \ddots & \vdots\\ 0 & \cdots & 0 & \boldsymbol{Q}_{n_2}\end{bmatrix},\quad \boldsymbol{R}=\begin{bmatrix}\boldsymbol{R}_1\\ \boldsymbol{R}_2\\ \vdots\\ \boldsymbol{R}_{n_2}\end{bmatrix}.$$

Here, $\hat{\boldsymbol{P}}^T\hat{\boldsymbol{P}}=\boldsymbol{I}$ by construction.

A coarse near kernel must be determined to allow for recursion to coarser levels. But $\boldsymbol{K}=\hat{\boldsymbol{P}}\boldsymbol{R}$ means that the fine-level near kernel can be exactly represented on the coarse level by simply choosing $\boldsymbol{K}_c=\boldsymbol{R}$. Note then that, with $\boldsymbol{A}_c=\hat{\boldsymbol{P}}^T\boldsymbol{A}\hat{\boldsymbol{P}}$, we have $\boldsymbol{A}_c\boldsymbol{K}_c=\hat{\boldsymbol{P}}^T\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{R}\approx\hat{\boldsymbol{P}}^T\boldsymbol{0}=\boldsymbol{0}$ since $\boldsymbol{A}\boldsymbol{K}\approx\boldsymbol{0}$.

As discussed in the supplemental material, this local non-overlapping use of the near kernel may generally satisfy the weak approximation property, but not the strong one needed to ensure optimal V-cycle performance. To improve accuracy of the interpolation operator, we therefore smooth it by applying the weighted Jacobi error propagation matrix: $\boldsymbol{P}=(\boldsymbol{I}-\omega\boldsymbol{D}^{-1}\boldsymbol{A})\hat{\boldsymbol{P}}$. The block diagonal matrix, $\boldsymbol{D}$, whose block diagonal agrees with that of $\boldsymbol{A}$, is used here because it does not change the sparsity pattern of $\boldsymbol{P}$ and it responds better to the local nature of $\boldsymbol{A}$. A typical choice for ω is $\frac{4}{3\rho(\boldsymbol{D}^{-1}\boldsymbol{A})}$, with care needed in estimating $\rho(\boldsymbol{D}^{-1}\boldsymbol{A})$ as discussed in section 6. Smoothed interpolation, while generally causing overlap in the aggregate basis functions so that $\boldsymbol{P}^T\boldsymbol{P}\neq\boldsymbol{I}$, often leads to an optimal V-cycle algorithm. Also, the smoothed near kernel is exactly in the range of smoothed interpolation: $\boldsymbol{P}\boldsymbol{K}_c=(\boldsymbol{I}-\omega\boldsymbol{D}^{-1}\boldsymbol{A})\hat{\boldsymbol{P}}\boldsymbol{K}_c=(\boldsymbol{I}-\omega\boldsymbol{D}^{-1}\boldsymbol{A})\boldsymbol{K}$, which generally preserves and even improves the near-kernel components in $\boldsymbol{K}$. While the finest-level matrix has nodes with b_1 degrees of freedom each, all coarser levels have κ degrees of freedom associated with each finer-level aggregate. The complexity of the coarse level is thus dictated by the number of near-kernel components and the aggressiveness of coarsening (that is, the size of the aggregates). Both choices must be controlled so that the coarse-level matrix has substantially fewer nonzero entries than the next finer-level matrix has.

The above steps outline how a given matrix and near kernel pair $\boldsymbol{A}, \boldsymbol{K}$ are decomposed to form a coarse level, the operators between the two levels, and the appropriate coarse matrix and near kernel. The combined process is summarized in Algorithm 3. The coarsening routine is applied recursively until there is a coarse matrix that can be easily inverted through iteration or a direct solver. Because aggregation coarsens aggressively, the number of levels is usually small, between three to six levels for all our tests.

Algorithm 3 Form a coarse level in the SA hierarchy

```
 1: input The matrix and kernel for this level, $A$ and $K$.
 2: Precompute inverse $D^{-1}$ of block diagonal of $A$
 3: Find spectral radius of $D^{-1}A$
 4: Smooth kernel $K$ to match boundary conditions
 5: Form a matrix $S$ to determine strength
 6: Use $S$ to form aggregates from nodes in $A$
 7: For each aggregate form local QR of $K$
 8: Use the local Q blocks to form tentative interpolation, $\hat{P}$
 9: Smooth the tentative interpolation to get $P$
10: Use $P$ to form $A_c = P^T A P$
11: Use local R blocks to form coarse kernel $K_c$
12: return Interpolation and restriction operators, $P, R$ as well as
    the coarse matrix and kernel, $A_c, K_c$.
```

6 Null space

In the previous section, we tacitly assumed that the near kernel for the fine-grid problem is known. While this is not always the case, near-kernel components can be obtained for many problems by examining the underlying PDE. For example, for elasticity, if we ignore boundary conditions, then it is well-known that a rigid-body mode (constant displacement or rotation) has zero strain and, therefore, zero stress. So rigid-body modes that are not prevented by boundary conditions from being in the domain of the PDE operator are actually in its kernel. Fixed boundary conditions prevent these modes from being admissible, so they cannot, of course, be kernel components in any global sense. But any rigid-body mode that is altered to satisfy typical conditions at the boundary becomes a near-kernel component, and they can usually be used locally to represent all near-kernel components.

To be more specific, displacements for *linear* elasticity are assumed to be small, thereby simplifying computation of the strain tensor. In particular, the strain is given by $\varepsilon = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)$, where $\boldsymbol{u}$ is the displacement field. In this case, it is easy to verify that the following vector functions (which represent rotations around each of the three axes) all lead to zero strain : $\boldsymbol{u} = (0, -z, y)$, $\boldsymbol{u} = (z, 0, -x)$, and $\boldsymbol{u} = (-y, x, 0)$. Here, (x, y, z) represents material (undeformed) coordinates. Since these rigid-body modes are linear functions, they should be well approximated in the interior of the domain by most finite element discretization methods. Indeed, if the finite element space includes linear functions, then these modes are exactly represented in the discretization except in the elements abutting fixed boundaries. All that is needed in this case is to *interpolate* these rigid-body modes at the nodes. In doing so, we make them admissible and retain their near-kernel nature by forcing their values at the boundary nodes to satisfy any fixed conditions that might be there. For further assurance that they remain near-kernel components, a relaxation sweep may be applied, using them as initial guesses to the solution of the homogeneous equation.

As noted above, this discussion assumes linear elasticity. Our limited experience suggests that the rigid-body modes that work for those equations may be good candidates for the linearized equations of nonlinear elasticity. We have not examined this issue in any depth, so it may benefit from additional work. However, just as relaxation sweeps may improve nearness to the kernel for modes that are altered to satisfy the boundary conditions and lie in the finite element space, so too may relaxation benefit the linear elasticity modes used for the nonlinear case.

A potentially more serious concern is our assumption that the thin-shell cloth simulation is essentially a discretization of linear elasticity. This assumption may not be valid and could suggest that our slightly suboptimal convergence of SA on these problems is due to incorrect near-kernel components. If the near kernel is unknown, then there are adaptive SA methods that attempt to discover the near kernel as part of the setup process, [Brezina et al. 2004; Brandt et al. 2011]. These methods have computationally expensive setup costs, which for this problem would be on the order of three times what it is for SA, not counting the need for an additional 10 cycles. To amortize this cost, convergence from an adaptively found kernel would need to be near optimal. We did experiment with an adaptive approach for this discretization, but, while increased convergence rates were indeed attainable, the extra setup cost made time to solution slower overall.

7 Smoothing

Our algorithm uses multigrid as a preconditioner for conjugate gradients (CG) rather than as a stand-alone solver. As a consequence, the multigrid preconditioner that we use must be symmetric and positive definite. This requirement has multiple implications. To ensure symmetry, the V-cycle that we use must be energy-symmetric and, to ensure positive definiteness, it is critical that the smoother be convergent in energy. (See the supplemental material for theory that proves this claim.) While these requirements are not surprising, it is important to take the necessary steps to ensure that they are satisfied.

The basic relaxation scheme that we use is a Chebyshev smoother, which amounts to running a fixed number of Chebyshev iterations at each level based on $\boldsymbol{D}^{-1}\boldsymbol{A}$ [Golub and Varga 1961]. A nice introduction to this smoother can be found in [Adams et al. 2003]. From a theoretical point of view, it has good properties and, in practice, it also performs the best as shown in [Adams et al. 2003; Baker et al. 2011]. Most importantly, it has the property of being implementable with mat-vec operations and is thus relatively easy to parallelize compared to other smoothers like Gauss-Seidel.

Chebyshev requires an upper bound for the spectral radius of $\boldsymbol{D}^{-1}\boldsymbol{A}$ and an estimate of the smallest part of the spectrum that we wish to attenuate, the same as needed for smoothing the interpolation operator by weighted Jacobi. One approach to obtaining these estimates is to use the power method for computing the largest eigenvalue of a matrix. Unfortunately, it does not generally provide a rigorous bound on the largest eigenvalue, and its convergence rate is limited by the ratio of the two largest eigenvalues [Golub and Loan 1983]. In practice, these two eigenvalues are often very close, so that convergence is very slow, which leads to a trade-off: too loose of an approximation to the spectral radius yields slow Chebyshev smoothing rates, while tighter approximations can be costly.

Another possibility is to use Gerschgorin's theorem to estimate the largest eigenvalue, but this approximation is too loose for our requirements. A potentially better alternative is to use Lanczos's method [Golub and Loan 1983]. However, while its convergence rate is better, it is also more expensive per iteration and may require careful numerical treatment to ensure convergence (e. g., periodic re-orthogonalization of the Krylov basis).

In the end we need an approximation to the spectral radius of $\boldsymbol{D}^{-1}\boldsymbol{A}$, rather than $\boldsymbol{A}$. While $\boldsymbol{D}^{-1}\boldsymbol{A}$ is not symmetric in the traditional sense, it *is* symmetric in the energy inner product defined by $\langle \boldsymbol{u}, \boldsymbol{v}\rangle_A = \langle \boldsymbol{u}, \boldsymbol{A}\boldsymbol{v}\rangle$. In fact, for any symmetric positive definite preconditioner $\boldsymbol{M}$, we have

$$\langle \boldsymbol{u}, \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{v}\rangle_A = \langle \boldsymbol{u}, \boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{v}\rangle = \langle \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u}, \boldsymbol{A}\boldsymbol{v}\rangle = \langle \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u}, \boldsymbol{v}\rangle_A.$$

$\boldsymbol{M}^{-1}\boldsymbol{A}$ is also positive definite in energy because

$$\langle \boldsymbol{u}, \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u}\rangle_A = \langle \boldsymbol{u}, \boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u}\rangle > 0$$

for any $\boldsymbol{u} \neq \boldsymbol{0}$. Its eigenvalues are therefore positive, so its spectral radius can be computed by finding the largest λ such that $\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}, \boldsymbol{x} \neq \boldsymbol{0}$, which is clearly equivalent to the generalized eigenvalue problem $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{M}\boldsymbol{x}, \boldsymbol{x} \neq \boldsymbol{0}$. We apply the generalized Lanczos algorithm [van der Vorst 1982] to this generalized eigenvalue problem to compute the spectral radius of $\boldsymbol{D}^{-1}\boldsymbol{A}$.

8 Collision handling

A challenge for many of the existing multigrid methods developed for cloth simulation is proper handling of constraints. As shown in [Boxerman and Ascher 2004], superior performance is achieved when the preconditioner for CG is based not on the full system, but rather on the *constraint null space*, by which we mean the null space of the constraint operator. The method they proposed constructs a pre-filtered matrix restricted to the constraint null space, but it is neither symmetric nor easily treated by our multigrid approach. A reduced set of equations can be constructed based on a null-space basis for the constraints, but this leads to a system where the block size is no longer constant. While this may seem like a minor inconvenience, it leads to either a substantial increase in code complexity and a reduced ability generate highly optimized code or reduced performance due to less coherent memory access patterns. In fact, with a constant block size, we can use BSR (block sparse row) matrices, where the innermost operations are effectively BLAS3 operations with high arithmetic intensity. By comparison, to use standard libraries for matrices with varying block size, we would have to use CSR matrices with comparatively low arithmetic intensity.

To obtain a system with a constant block size, we form a reduced system, but replace all eliminated variables with dummy ones. This retains the block structure while ensuring that our preconditioner operates on the constraint null space. We refer to this method as Pre-filtered Preconditioned CG (PPCG).

To explain PPCG, the "modified" linear system solved by [Baraff and Witkin 1998] can be written in the notation from [Ascher and Boxerman 2003] as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \|\boldsymbol{S}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})\| \\ \text{s.t.} \quad & (\boldsymbol{I}-\boldsymbol{S})\boldsymbol{x} = (\boldsymbol{I}-\boldsymbol{S})\boldsymbol{z}, \end{aligned} \tag{3}$$

where: $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the matrix for the full system; $\boldsymbol{b}$ is the corresponding right-hand side; $\boldsymbol{S}$, which represents the filtering operation in [Baraff and Witkin 1998], is the orthogonal projection matrix onto the constraint null space (not to be confused with the SOC matrix in section 5); and $\boldsymbol{z}$ is a vector of the desired values for the constrained variables. It is assumed that $\boldsymbol{A}$ is symmetric positive definite.

Due to the constraint, any feasible point must satisfy

$$\boldsymbol{x} = \boldsymbol{S}\boldsymbol{x} + (\boldsymbol{I}-\boldsymbol{S})\boldsymbol{x} = \boldsymbol{S}\boldsymbol{x} + (\boldsymbol{I}-\boldsymbol{S})\boldsymbol{z}.$$

To incorporate this expression into the objective function, first note that

$$\boldsymbol{S}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{S}\boldsymbol{b} - \boldsymbol{S}\boldsymbol{A}(\boldsymbol{S}\boldsymbol{x} + (\boldsymbol{I}-\boldsymbol{S})\boldsymbol{z}) = \boldsymbol{S}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{z}) - \boldsymbol{S}\boldsymbol{A}\boldsymbol{S}(\boldsymbol{x}-\boldsymbol{z}).$$

By introducing $\boldsymbol{c} \equiv \boldsymbol{b} - \boldsymbol{A}\boldsymbol{z}$ and $\boldsymbol{y} \equiv \boldsymbol{x} - \boldsymbol{z}$, we can then rewrite the constrained minimization problem in Equation (3) as

$$\begin{aligned} \min_{\boldsymbol{y}} \quad & \|\boldsymbol{S}\boldsymbol{c} - \boldsymbol{S}\boldsymbol{A}\boldsymbol{S}\boldsymbol{y}\| \\ \text{s.t.} \quad & (\boldsymbol{I}-\boldsymbol{S})\boldsymbol{y} = \boldsymbol{0}. \end{aligned} \tag{4}$$

By construction, $\boldsymbol{S}$ is symmetric and, therefore, diagonalizable. Since it is also orthogonal, it follows that the eigenvalues must be either 0 or 1. We can thus compute another orthogonal matrix, $\boldsymbol{Q}$, such that

$$\boldsymbol{S} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{I}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}\boldsymbol{Q}^T \equiv \boldsymbol{Q}\boldsymbol{J}\boldsymbol{Q}^T, \quad r = \dim(\text{range}(\boldsymbol{S})).$$

If we now partition $\boldsymbol{Q}$ into $\boldsymbol{V} \in \mathbb{R}^{n\times r}$ and $\boldsymbol{W} \in \mathbb{R}^{n\times n-r}$ such that $\boldsymbol{Q} = (\boldsymbol{V}, \boldsymbol{W})$, then $\boldsymbol{V}$ is a basis for the constraint null space while $\boldsymbol{W}$ is a basis for the constrained subspace. From this decomposition, it follows that $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{V}^T$ and $\boldsymbol{I} - \boldsymbol{S} = \boldsymbol{W}\boldsymbol{W}^T$.

Similarly, let

$$\boldsymbol{d} \equiv \begin{pmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \end{pmatrix} = \boldsymbol{Q}^T\boldsymbol{c}, \quad \boldsymbol{u} \equiv \begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{pmatrix} = \boldsymbol{Q}^T\boldsymbol{y}.$$

Using the last definition, we have $\boldsymbol{y} = \boldsymbol{V}\boldsymbol{u}_1 + \boldsymbol{W}\boldsymbol{u}_2$ and, therefore,

$$\boldsymbol{V}\boldsymbol{V}^T\boldsymbol{y} = \boldsymbol{V}\boldsymbol{u}_1 \quad \text{and} \quad \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{y} = \boldsymbol{W}\boldsymbol{u}_2. \tag{5}$$

Combining the above definitions and substituting $\boldsymbol{Q}\boldsymbol{J}\boldsymbol{Q}^T$ for $\boldsymbol{S}$, we can rewrite the objective function in Equation (4) as follows :

$$\begin{aligned} \phi &= \|\boldsymbol{S}\boldsymbol{c} - \boldsymbol{S}\boldsymbol{A}\boldsymbol{S}\boldsymbol{y}\| \\ &= \|\boldsymbol{Q}\boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{c} - \boldsymbol{Q}\boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q}\boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{y}\| \\ &= \|\boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{c} - \boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q}\boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{y}\| \\ &= \|\boldsymbol{J}\boldsymbol{d} - \boldsymbol{J}\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q}\boldsymbol{J}\boldsymbol{u}\| \\ &= \left\| \begin{pmatrix} \boldsymbol{d}_1 \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}\begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{pmatrix} \right\|. \end{aligned}$$

This system can clearly be reduced by eliminating $\boldsymbol{u}_2$ since any value of $\boldsymbol{u}_2$ produces the same value of ϕ. However, eliminating $\boldsymbol{u}_2$ creates a smaller system, which means that if $\boldsymbol{A}$ is block sparse with fixed block size, then the reduced system will in general be block sparse with different block sizes. To keep the original size of the system we could leave all the zero blocks in place, but the resulting system would then be singular, thereby introducing other problems. On the other hand, we know from Equation (4) that $(\boldsymbol{I}-\boldsymbol{S})\boldsymbol{y} = \boldsymbol{0}$ and, since $(\boldsymbol{I}-\boldsymbol{S})\boldsymbol{y} = \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{y} = \boldsymbol{W}\boldsymbol{u}_2$, it follows that $\boldsymbol{u}_2 = \boldsymbol{0}$. Minimizing ϕ subject to the desired constraint is therefore equivalent to solving the following linear system :

$$\begin{pmatrix} \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}\begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{d}_1 \\ \boldsymbol{0} \end{pmatrix}. \tag{6}$$

Rotating this back to our original coordinates yields

$$\boldsymbol{Q}\begin{pmatrix} \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}\boldsymbol{Q}^T\boldsymbol{Q}\begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{pmatrix} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{d}_1 \\ \boldsymbol{0} \end{pmatrix}$$

or, equivalently,

$$\left(\boldsymbol{V}\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T + \boldsymbol{W}\boldsymbol{W}^T\right)\boldsymbol{y} = \boldsymbol{V}\boldsymbol{d}_1 = \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{c}.$$

Since $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{V}^T$ and $\boldsymbol{I} - \boldsymbol{S} = \boldsymbol{W}\boldsymbol{W}^T$, we finally arrive at

$$(\boldsymbol{S}\boldsymbol{A}\boldsymbol{S} + \boldsymbol{I} - \boldsymbol{S})\boldsymbol{y} = \boldsymbol{S}\boldsymbol{c}. \tag{7}$$

The importance of Equation (7) is that we now have a *symmetric positive definite* (in particular, full rank) system of the same dimensions as the original system, but which correctly projects out all of the constraints. From this system, the solution to Equation (3) is easily recovered as $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{z}$.

Furthermore, the condition number of the new system is no worse than that of $\boldsymbol{A}$, and may in fact be better. This conclusion is based on the assumption that 1 is in the *field of values* of $\boldsymbol{A}$, which is defined to be the real numbers inclusively between the smallest and largest eigenvalues of $\boldsymbol{A}$. (This assumption often holds for PDEs; otherwise, we can simply multiply $\boldsymbol{I} - \boldsymbol{S}$ by a scalar that is in $\boldsymbol{A}$'s field of values.) To prove this assertion, first note that our assumption implies that the field of values of $\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}$ is the same as that of the matrix in Equation (6), which is in turn the same as that of the matrix in Equation (7) because they are related by a similarity transformation. By Cauchy's interlacing theorem [Horn and Johnson 1985], the field of values of $\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}$ is a subset of that of $\boldsymbol{A}$, which immediately implies that the condition number of $\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}$ is bounded by the condition number of $\boldsymbol{A}$.

For the constraints considered by Baraff and Witkin [1998], $\boldsymbol{S}$ is block diagonal, so the computation of $\boldsymbol{SAS}$ amounts to simple blockwise row and column scaling of $\boldsymbol{A}$, while the addition of $\boldsymbol{I}-\boldsymbol{S}$ only affects the block diagonal. It should be noted that, while the derivation required $\boldsymbol{S}$ to be diagonalized, the final result does not. We refer to the system in Equation (7) as the prefiltered system, to which we apply a standard preconditioned CG algorithm.

The constraints considered by Baraff and Witkin [1998] are limited to (cloth) vertices against (collision) objects. However, the above method easily generalizes to any kind of equality constraint, including the face-vertex and edge-edge constraints used in other collision response algorithms (e.g., [Harmon et al. 2008], [Otaduy et al. 2009]).

Finally, it should be noted that none of the derivations in this section depend on multigrid methods: the results can be used with any type of linear solver. However, by using Equation (7) with smoothed aggregation, we have an effective way of coarsening not just the dynamics, but also the constraints.

9 Implementation

Our implementation of SA depends critically on Intel's MKL v11.3 for highly optimized implementations of most of the basic linear algebra operations like sparse matrix-vector and matrix-matrix products as well as QR decompositions. To achieve good performance, many structures are pre-allocated and re-used throughout the simulation, leading to slightly higher memory consumption, but not excessively so. The code is somewhat parallelized, but is generally limited by memory bandwidth rather than CPU resources. Additional improvements are definitely possible.

One of the most expensive operations in any Galerkin multigrid algorithm is forming the coarse-grid operator since it is based on a triple-matrix product. Despite the symmetric nature of this product, it is currently most efficiently implemented using two sparse mat-mat multiply operations. We use MKL for this purpose, while others have explored doing it on a GPU [Dalton et al. 2015]. Regardless of the implementation of the matrix products, care has to be taken when constructing the interpolation matrix in the presence of special nodes. The range of interpolation is the entire fine level and must contain zeros for all special nodes. However, in a naive implementation, if these zero entries in the interpolation matrix are not pruned, then the mat-mat products will introduce structural fill-in that can significantly affect the run time, especially after smoothing the interpolation operator.

10 Examples

To evaluate and compare our approach to existing methods, we consider five procedurally generated examples at different resolutions, and a high-resolution production example to show its practical applicability. The five simple examples are shown in figure 4, while the production example is shown in figure 1. The animation for all of these can be seen in the supplemental video.

The first example is a fully pinned piece of cloth subjected to gravity. The corresponding PDE is fully elliptic with a full Dirichlet boundary, meaning in part that it is positive definite with an order 1 constant. The cloth that is pinned on two sides in the second example has a smaller constant and thus provides a test to see how sensitive our methods are to variations in the strength of ellipticity. The third example has a re-entrant corner, which is more difficult yet because it does not possess full ellipticity, which means that the standard approximation properties discussed above do not apply. Standard discretization and solution methods have serious difficulties in the presence of re-entrant corners, so they provide an especially challenging test problem for our methodology. In the fourth example, the cloth falls flat on a flat box with a hole in the middle. This generates many contact constraints and thus illustrates the performance of our PPCG method well. Finally, in the last example, we drop the cloth on the same box, but this time from a vertical configuration. In this way, we observe plenty of buckling and also lots of cloth-cloth collisions. The examples are referred to as *pinned*, *drooping*, *reentrant*, *dropHorizontal*, and *dropVertical*, respectively. Each of the five simple examples were run with both regular and irregular tessellations.

11 Evaluation

All simulations were run on dual socket systems with Intel(R) Xeon(R) E5-2698 v3 @ 2.30GHz processors, each processor with 16 cores and each system configured with 64 GB DDR4 RAM. All simulations were run with a time step of $\Delta t = 2$ ms. The stopping criterion used in the CG method is a small relative residual error : $||\boldsymbol{r}_i||_{\boldsymbol{M}^{-1}}/||\boldsymbol{r}_0||_{\boldsymbol{M}^{-1}} < \epsilon$, where $\boldsymbol{M}$ is the chosen preconditioner, $||\cdot||_{\boldsymbol{M}^{-1}} \equiv ||\boldsymbol{M}^{-\frac{1}{2}}\cdot||$, $\boldsymbol{r}_i$ is the residual after i iterations, and ϵ is a given tolerance, which we set to 10^{-5}.

An important point to keep in mind is that the relative residual error used to judge convergence gives an unfair advantage to poor preconditioners like the block diagonal ones. This observation comes from first realizing that the relative residual in practice is only as good as how tight it bounds the relative energy norm of the error. Remembering that $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{e}$, we have the bound

$$\frac{||\boldsymbol{e}_i||_{\boldsymbol{A}}}{||\boldsymbol{e}_0||_{\boldsymbol{A}}} = \frac{||\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{r}_i||}{||\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{r}_0||} = \frac{||\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{M}^{\frac{1}{2}}\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{r}_i||}{||\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{M}^{\frac{1}{2}}\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{r}_0||} \le C\frac{||\boldsymbol{r}_i||_{\boldsymbol{M}^{-1}}}{||\boldsymbol{r}_0||_{\boldsymbol{M}^{-1}}},$$

where $C = \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}}$, with $\lambda_{\max}$ and $\lambda_{\min}$ the respective largest and smallest eigenvalues of $\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{M}^{-\frac{1}{2}}$ or, equivalently, of $\boldsymbol{M}^{-1}\boldsymbol{A}$. This bound means that the practical error measure that we use is sharp only up to the square root of spectral condition number of $\boldsymbol{M}^{-1}\boldsymbol{A}$. If $\boldsymbol{M}$ is a reasonable approximation to $\boldsymbol{A}$ as it hopefully is for our multigrid scheme, then this bound is pretty tight. If $\boldsymbol{M}$ is not a very accurate approximation to $\boldsymbol{A}$, as when $\boldsymbol{M}$ is its diagonal part, then we may believe that we have converged well before the true relative energy norm of the error is sufficiently small. Said differently, diagonally preconditioned iteration has the smoothing property that the residuals they produce are very small compared to the actual error. More precisely, as we show in

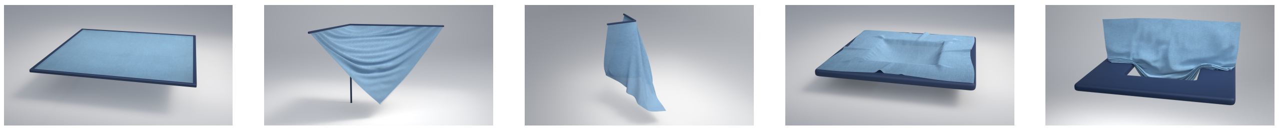


Figure 4: *The five examples shown here represent problems of increasing difficulty that we use for benchmarking. They are generated procedurally, with the vertex count ranging from* 1,000 *to* 100,000. *The ones shown here have* 40,000 *vertices.*

the supplemental material, relaxation produces error whose relative energy norm is much larger than its relative residual norm. In other words, the smooth error left after relaxation is hidden in the residual because it consists mostly of low eigenmodes of $\boldsymbol{A}$. SA tends much more to balance the error among the eigenmodes, so a small relative residual for SA is much more reflective of small relative energy error. This should be kept in mind in the consideration of the results we present below.

The size of the cloth in all our examples is 1m $\times$ 1m and the material parameters are the defaults used by our production solver, which approximates cotton. We use $\theta = 0.48$, but have experimented with other values and found the results to be fairly insensitive to the exact choice. If a suboptimal value is chosen, then the convergence rate still tends to be good, but the time to solution suffers from excessive amounts of fill-in on the coarser levels. For smoothing, we use a single sweep of Chebyshev with a second-order polynomial. Higher-order polynomials improve the convergence rate significantly, but at too high a computational price. For the spectral radius estimation, we use 10 iterations of the generalized Lanczos method. Once again, the convergence rate improves if this number is increased, but not in a computationally cost-effective way. For the kernel, we picked six vectors, one constant for each of the unknowns and the three rotations described in section 5.

The benchmark numbers are averages over all the solves required for 10 frames of animation for the pinned, drooping, and re-entrant examples, while we used 15 frames for dropHorizontal and 25 frames for dropVertical. These durations are chosen to capture most of the interesting behavior for each of the examples. In the following, we refer to block diagonally preconditioned conjugate gradients simply as *Diag-PCG*, while we refer to our smoothed aggregation preconditioned conjugate gradient method with pre-filtering as *SA+PPCG*.

11.1 Convergence rates

The expectation for a multigrid preconditioner is that it should improve the convergence rate of the conjugate gradient method significantly and that the convergence rate should be independent of (or only weakly dependent on) the problem size. To investigate this, define the convergence rate at iteration i by $\rho_i = \|r_i\|_{\boldsymbol{M}^{-1}}/\|r_{i-1}\|_{\boldsymbol{M}^{-1}}$. The average convergence rate, ρ, within a single solve is then the geometric mean of these values. For an entire animation, we refer to the average convergence rate as the average over all solves of ρ.

Figure 5 shows average convergence rates of SA+PPCG and Diag-PCG for each of our five examples. While Diag-PCG approaches a convergence rate close to 1 very quickly, our method stays bounded away from 1 at significantly lower values. Note that the pinned example converges faster than the drooping example, which in turn converges faster than the re-entrant example. This observation is in agreement with our expectations based on the underlying PDEs. Note the irregularity of the curve for the re-entrant corner example, due perhaps to the loss of ellipticity.

11.2 Setup time

To achieve better convergence rates, we need both setup and pre-filtering. The computational cost of both of these is illustrated in figure 6. The setup cost for AMG-type methods is often dominated by the cost of the triple-matrix product when forming the coarse matrices. In our case, this is still a significant cost, but not the dominant one. The computation of the spectral radius estimates is currently expensive and the computation of aggregates remains entirely serial. As such, there is room for improvement of these numbers.

To reduce the setup time, it is possible to only update the preconditioner periodically or adaptively based on the observed convergence rate. We experimented with only performing an update if the convergence rate after four iterations was substantially less than in previous solve. However, we found the cost of restarting the solver to outweigh the benefit. A more sophisticated controller design might make adaptive setup more beneficial, but this remains future work.

11.3 Time to solution

Ultimately, the most important metric is usually time to solution. However, making fair comparisons is not entirely straightforward because different solvers have different strengths. As an example, Diag-PCG is generally attractive for systems that are highly diagonally dominant or when the required accuracy is very modest. For cloth simulations, diagonally dominant systems generally occur with small timesteps, soft materials, and/or large masses. At the other end of the spectrum, we compare our results to those of Intel MKL's highly optimized version of PARDISO. As a sparse direct solver, it is attractive if very high accuracy is required or if the problem is very ill-conditioned.

For our comparisons, we use a moderate accuracy (relative error less than 10^{-5}). We note that our results do look better with stiffer materials and/or larger time steps, but do not report on that here.

The results for four of the examples are shown in figure 7. As can be seen, Diag-PCG is consistently best for small problems, but our method is superior for problems with roughly 25k vertices or more. Somewhat surprisingly, PARDISO shows very good scaling behavior considering that it is a direct solver, but it remains about twice as slow as SA+PPCG. The empirical complexity of our method can be seen to be close to linear in the size of the problem as expected.

The outlier in the above set of results is the dropVertical example. As already seen in figure 5, the convergence rate for this problem is worse than for the others, as is the time to solution. The reason for the different behavior in this example is the large number of cloth repulsion springs added to handle cloth-cloth collisions. While our method handles this without any special cases, they do not stem from an underlying and well-behaved PDE, so we lose some optimality. However, we still see the same basic scaling behavior as the problem size increases, and the method remains superior to Diag-PCG for large problems.

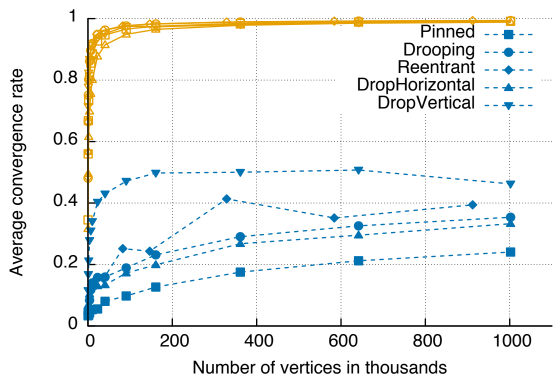


Figure 5: *The average convergence rate over the length of the entire animation for each of our examples. The orange curves are for Diag-PCG while the blue curves are for SA+PPCG.*

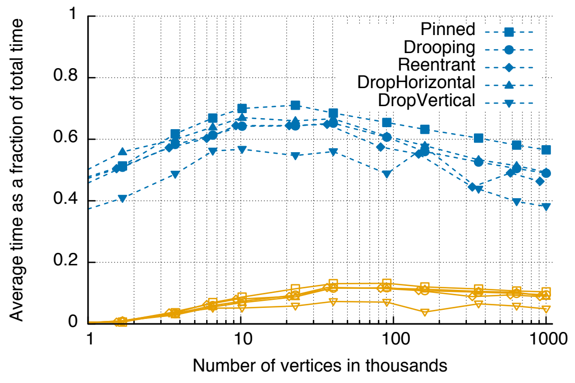


Figure 6: *The average time for prefiltering and setup as a percentage of the total solve time. The combined preprocessing time is the sum of the two. Setup is shown in blue while prefiltering is shown in orange. The corresponding total solve time is shown in figure 7.*

In figure 8, we consider the solve time for our horizontal drop example, including now two additional solvers. The first is smoothed aggregation *without* our prefiltering method. The second is Diag-PCG with prefiltering added. For low resolutions, we see that prefiltering further improves the superior performance of Diag-PCG and, for all resolutions, we see that prefiltering is essential for the good performance of SA. In fact, without prefiltering, SA at large resolutions is no better than Diag-PCG and may in fact be worse. However, with prefiltering, SA+PPCG continues to be the best solution for large problems. The reason prefiltering turns out to be so critical for SA is that SA is a very good preconditioner, so a single step without any knowledge about the constraints can bring the solver far from the constraint manifold, and the subsequent projection step is likely to undo much of the progress just made by the solver.

Finally, we present a production example in figure 9. Overall, we observe an average 8× speedup for a walk-cycle animation and a 6× speedup for a run-cycle. However, as seen from the figure, the variation in the solve time from frame to frame was substantially less with our method than with Diag-PCG.

The speedups for all our examples are summarized in table 1.

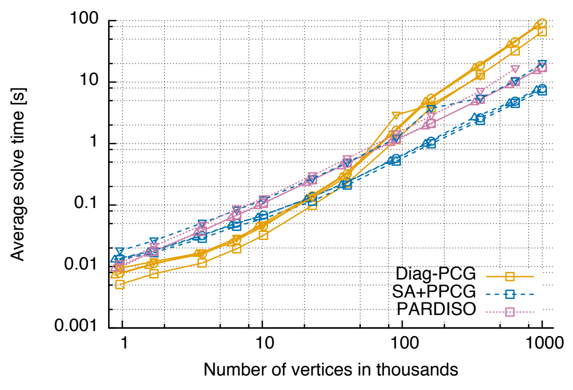


Figure 7: *Average time for one linear solve using Diag-PCG (orange), SA+PPCG (blue), and PARDISO (pink). The graphs are shown for four examples : pinned (square markers), drooping (circle markers), re-entrant (diamond markers), and verticalDrop (triangle markers)1.*

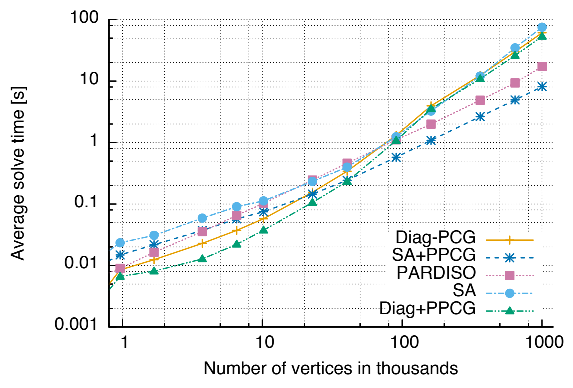


Figure 8: *Average time for one linear solve in our horizontal drop example. Note that the fastest time is always obtained using prefiltering.*

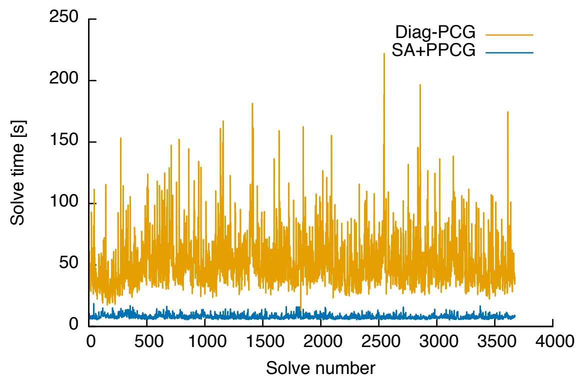


Figure 9: *Time for each linear solve in our production example using a walk-cycle animation.*

	Num. vertices	Pinned		Drooping		Re-entrant		DropHorizontal		DropVertical	
Regular tessellation	961	0.39	0.76	0.58	0.72	0.55	0.72	0.57	0.61	0.53	0.61
	1681	0.45	1.06	0.63	0.99	0.60	0.95	0.57	0.75	0.51	0.80
	3721	0.40	1.31	0.50	1.18	0.54	1.17	0.62	0.96	0.35	0.94
	6561	0.43	1.51	0.53	1.35	0.50	1.31	0.65	1.14	0.35	1.04
	10201	0.54	1.78	0.66	1.57	0.72	1.53	0.77	1.37	0.40	1.06
	22801	0.85	2.23	1.03	1.89	1.02	1.86	1.07	1.69	0.50	1.14
	40401	1.07	2.29	1.35	2.03	1.36	2.01	1.40	1.89	0.60	1.18
	90601	2.23	2.27	2.87	2.07	2.67	1.95	2.27	1.92	2.35	1.11
	160801	3.89	2.14	5.01	1.95	4.87	1.92	3.64	1.84	1.14	0.77
	361201	5.48	2.21	7.10	2.01	6.38	1.76	4.70	1.85	2.29	1.32
	641601	7.12	2.26	9.28	2.10	8.84	1.96	6.09	1.89	3.46	1.53
	1002001	9.20	2.40	11.64	2.20	10.89	2.04	7.53	2.13	3.01	1.80
Irregular tessellation	961	0.43	0.87	0.49	0.80	0.51	0.80	0.52	0.64	0.56	0.71
	1681	0.46	1.17	0.52	1.12	0.62	1.04	0.60	0.94	0.54	1.01
	3721	0.43	1.61	0.46	1.45	0.55	1.39	0.56	1.21	0.38	1.06
	6561	0.46	1.86	0.50	1.65	0.56	1.72	0.69	1.44	0.44	1.32
	10201	0.56	2.06	0.61	1.87	0.67	1.93	0.89	1.74	0.53	1.42
	22801	0.81	2.49	0.87	2.11	0.97	2.20	1.13	2.13	0.57	1.43
	40401	1.22	2.61	1.19	2.21	1.43	2.32	1.33	2.23	0.76	1.56
	90601	2.32	2.54	2.51	2.21	2.36	2.33	2.20	2.19	1.29	1.39
	160801	3.78	2.45	4.20	2.18	4.22	2.16	3.26	2.10	2.03	1.39
	361201	5.32	2.49	5.98	2.22	6.13	2.21	4.63	2.13	2.63	1.57
	641601	7.02	2.56	7.84	2.24	7.72	2.19	5.93	2.21	2.39	1.68
	1002001	8.97	2.73	9.82	2.56	9.59	2.47	5.74	2.08	2.81	1.71

Table 1: *The speedup factors of SA+PPCG relative to Diag-PCG (first column for each example) and PARDISO (second column for each example). Shaded cells indicate speedups greater than* 1. *Diag-PCG is on average* 11% *faster with the irregular tessellation, while PARDISO on average is* 7% *slower and SA+PPCG is* 8% *faster.*

12 Limitations and future work

One limitation of SA compared to Diag-PCG is its somewhat larger memory overhead. However, the overhead is generally less than 50 percent. SA+PPCG also comes with a higher coding complexity than a simple Diag-PCG, and an unoptimized implementation may often be slower.

The current algorithm is limited by memory bandwidth and, in practice, we saw negligible speedups from parallelization after eight cores. But it remains superior to Diag-PCG and, for our problems, was comparable to the parallel efficiency of MKL's PARDISO. Even so, we believe there is room for improvement, either by exploring techniques such as those presented in [Bell et al. 2012] for forming aggregates in parallel or by algorithmic changes with better parallel implications. Incremental setup along the lines of what [Hecht et al. 2012] did for Cholesky factorizations might be another way to reduce the performance hit of time spent in setup.

While the results show near optimal behavior, they are not perfect. The convergence rates degrade (slowly) as the problem sizes grow, leading to more iterations while cycling, so time to solution is not quite scaling linearly. To attain ideal convergence and subsequently linear scaling, further improvements should be made to the construction of the kernel components to ensure that they match the discretization at all times. Also, a more accurate discretization of the underlying PDEs than that provided by Baraff and Witkin [1998] might offer a better foundation for building multigrid methods. We intend to investigate this in the future.

Finally, we note that the present work only considers linear problems, but a natural extension is to use the proposed solver in the inner loop of an outer nonlinear iteration that involves a linearization strategy such as Newton's method.

13 Conclusion

This paper presented a new preconditioner for linear systems formed in implicit cloth simulations by developing an algebraic multigrid hierarchy based on the underlying PDEs and discretization. The SA solver provides a faster solution than existing methods for typical problems with 25,000 vertices or more. For problems that are stiffer, have smaller mass, or are subjected to larger time steps, the advantages of the method increase and show speedups for smaller problems as well.

To realize the full potential of the SA in cloth simulation and attain near optimal scaling in the presence of collisions, we had to pair it with our new PPCG method. SA+PPCG is attractive because no changes need to be made to existing collision detection and response methods to handle the linear solves.

Acknowledgements

The authors would like to thank Intel's MKL team for their support and development during this project. We are also grateful for the assistance from Jeff MacNeill, Garrett Raine, Benjamin Huang, Heather Pritchett, and Angela McBride (WDAS) in setting up and rendering our simulations. Finally, Eitan Grinspun, Bernhard Thomaszewski, Marian Brezina, and John Ruge provided valuable discussions. All images are ©Disney 2015.

References

Adams, M., Brezina, M., Hu, J., and Tuminaro, R. 2003. Parallel Multigrid Smoothing: Polynomial Versus Gauss–Seidel. *J. Comput. Phys. 188*, 2 (July), 593–610.

Ascher, U. M., and Boxerman, E. 2003. On the modified conjugate gradient method in cloth simulation. *The Visual Computer 19*, 7-8, 526–531.

Baker, A. H., Falgout, R. D., Kolev, T. V., and Yang, U. M. 2011. Multigrid Smoothers for Ultraparallel Computing. *SIAM Journal on Scientific Computing 33*, 5, 2864–2887.

Baraff, D., and Witkin, A. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 43–54.

Bell, N., Dalton, S., and Olson, L. 2012. Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods. *SIAM Journal on Scientific Computing 34*, 4, C123–C152.

Boxerman, E., and Ascher, U. 2004. Decomposing Cloth. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, SCA '04, 153–161.

Brandt, A., McCormick, S. F., and Ruge, J. W. 1985. Algebraic multigrid (AMG) for sparse matrix equations. In *Sparsity and its Applications*, D. J. Evans, Ed. Cambridge University Press, 257–284.

Brandt, A., Brannick, J., Kahl, K., and Livshits, I. 2011. Bootstrap AMG. *SIAM J. Sci. Comput. 33*, 2 (Mar.), 612–632.

Brandt, A. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation 31*, 138, 333–390.

Brezina, M., Falgout, R., MacLachlan, S., Manteuffel, T., McCormick, S., and Ruge, J. 2004. Adaptive Smoothed Aggregation (αSA). *SIAM Journal on Scientific Computing 25*, 6, 1896–1920.

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.

Briggs, W., Henson, V., and McCormick, S. 2000. *A Multigrid Tutorial, Second Edition.* Society for Industrial and Applied Mathematics.

Dalton, S., Olsen, L., and Bell, N. 2015. Optimizing Sparse Matrix-Matrix Multiplication for the GPU. *ACM Transactions on Mathematical Software 41*, 4.

Fish, J., Pan, L., Belsky, V., and Gomaa, S. 1996. Unstructured multigrid methods for shells. *International Journal for Numerical Methods in Engineering 39*, 7, 1181–1197.

Gee, M. W., and Tuminaro, R. S. 2006. Nonlinear Algebraic Multigrid for Constrained Solid Mechanics Problems Using Trilinos. Tech. Rep. SAND2006-2256, Sandia National Laboratories, April.

Gee, M., Ramm, E., and Wall, W. A. 2005. Parallel multilevel solution of nonlinear shell structures. *Computer Methods in Applied Mechanics and Engineering 194*, 21-24, 2513–2533. Computational Methods for Shells.

Golub, G. H., and Loan, C. F. V. 1983. *Matrix Computations*, 3rd ed. The Johns Hopkins University Press.

Golub, G. H., and Varga, R. S. 1961. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods. *Numerische Mathematik 3*, 1, 157–168.

Harmon, D., Vouga, E., Tamstorf, R., and Grinspun, E. 2008. Robust Treatment of Simultaneous Collisions. *ACM Trans. Graph. 27*, 3 (Aug.), 23:1–23:4.

Hecht, F., Lee, Y. J., Shewchuk, J. R., and O'Brien, J. F. 2012. Updated Sparse Cholesky Factors for Corotational Elastodynamics. *ACM Transactions on Graphics 31*, 5 (Oct.), 123:1–13. Presented at SIGGRAPH 2012.

Horn, R. A., and Johnson, C. R. 1985. *Matrix Analysis.* Cambridge University Press. Cambridge Books Online.

Jeon, I., Choi, K.-J., Kim, T.-Y., Choi, B.-O., and Ko, H.-S. 2013. Constrainable Multigrid for Cloth. *Computer Graphics Forum 32*, 7, 31–39.

Krishnan, D., Fattal, R., and Szeliski, R. 2013. Efficient Preconditioning of Laplacian Matrices for Computer Graphics. *ACM Trans. Graph. 32*, 4 (July), 142:1–142:15.

Lee, Y., Yoon, S.-e., Oh, S., Kim, D., and Choi, S. 2010. Multi-Resolution Cloth Simulation. *Computer Graphics Forum 29*, 7, 2225–2232.

McCormick, S. 1984. Multigrid Methods for Variational Problems: Further Results. *SIAM Journal on Numerical Analysis 21*, 2, 255–263.

Míka, S., and Vaněk, P. 1992. Acceleration of convergence of a two-level algebraic algorithm by aggregation in smoothing process. *Applications of Mathematics 37*, 5, 343–356.

Narain, R., Samii, A., and O'Brien, J. F. 2012. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Trans. Graph. 31*, 6 (Nov.), 152:1–152:10.

Oh, S., Noh, J., and Wohn, K. 2008. A physically faithful multigrid method for fast cloth simulation. *Computer Animation and Virtual Worlds 19*, 3-4, 479–492.

Otaduy, M. A., Tamstorf, R., Steinemann, D., and Gross, M. 2009. Implicit Contact Handling for Deformable Objects. *Computer Graphics Forum 28*, 2, 559–568.

Trottenberg, U., Oosterlee, C. W., and Schuller, A. 2000. *Multigrid.* Academic press.

van der Vorst, H. A. 1982. A Generalized Lanczos Scheme. *Mathematics of Computation 39*, 160 (Oct), pp. 559–561.

Vassilevski, P. S. 2008. *Multilevel Block Factorization Preconditioners, Matrix-based Analysis and Algorithms for Solving Finite Element Equations.* Springer.