



Surface Multigrid via Intrinsic Prolongation

HSUEH-TI DEREK LIU, University of Toronto, Canada

JIAIYI ERIS ZHANG, University of Toronto, Canada

MIRELA BEN-CHEN, Technion - Israel Institute of Technology, Israel

ALEC JACOBSON, University of Toronto, Canada

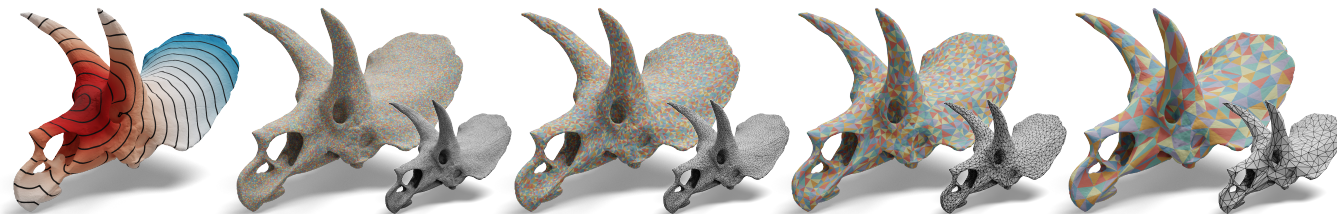


Fig. 1. We introduce a novel geometric multigrid solver for curved surfaces. Our key ingredient is an *intrinsic* prolongation operator computed via parameterizing the high resolution shape via its coarsened counterpart, visualized using colored triangles. By recursively applying this self-parameterization, we obtain a hierarchy (from left to right) for our multigrid method (e.g., to solve heat geodesics [Crane et al. 2017], far left). ©model by Benoît Rogez under CC BY-NC.

This paper introduces a novel geometric multigrid solver for unstructured curved surfaces. Multigrid methods are highly efficient iterative methods for solving systems of linear equations. Despite the success in solving problems defined on structured domains, generalizing multigrid to unstructured curved domains remains a challenging problem. The critical missing ingredient is a prolongation operator to transfer functions across different multigrid levels. We propose a novel method for computing the prolongation for triangulated surfaces based on intrinsic geometry, enabling an efficient geometric multigrid solver for curved surfaces. Our surface multigrid solver achieves better convergence than existing multigrid methods. Compared to direct solvers, our solver is orders of magnitude faster. We evaluate our method on many geometry processing applications and a wide variety of complex shapes with and without boundaries. By simply replacing the direct solver, we upgrade existing algorithms to interactive frame rates, and shift the computational bottleneck away from solving linear systems.

CCS Concepts: • **Computing methodologies** → **Mesh models**; • **Applied computing**;

Additional Key Words and Phrases: computer graphics, geometric multigrid

ACM Reference Format:

Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface Multigrid via Intrinsic Prolongation. *ACM Trans. Graph.* 40, 4, Article 80 (August 2021), 13 pages. <https://doi.org/10.1145/3450626.3459768>

1 INTRODUCTION

Linear solvers are the heart of many geometry processing algorithms. For positive (semi-)definite problems defined on surface meshes,

Authors' addresses: Hsueh-Ti Derek Liu, University of Toronto, Canada, hsuehtil@cs.toronto.edu; Jiayi Eris Zhang, University of Toronto, Canada, jiayieris.zhang@mail.utoronto.ca; Mirela Ben-Chen, Technion - Israel Institute of Technology, Israel, mirela@cs.technion.ac.il; Alec Jacobson, University of Toronto, Canada, jacobson@cs.toronto.edu.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2021 Copyright held by the owner/author(s).

0730-0301/2021/8-ART80

<https://doi.org/10.1145/3450626.3459768>

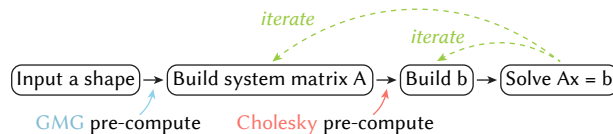


Fig. 2. Many geometry processing algorithms that involve solving linear systems $Ax = b$ often consist of four steps: (1) loading a geometry, (2) building the left-hand-side A , (3) building the right-hand-side b , and (4) solving the system $Ax = b$. Direct solvers (e.g., Cholesky) perform pre-computation after building A , making it suitable for applications where only b is changing. Geometric multigrid methods perform pre-computation solely based on the geometry. Thus, even the entire system A , b are changing, geometric multigrid solvers can still leverage the same pre-computed hierarchy to solve the system efficiently.

direct solvers (e.g., Cholesky factorization) are commonplace. Unfortunately, direct solvers do not scale and often become the bottleneck for problems on high-resolution surface meshes. Especially for applications where the linear system changes at every iteration (e.g., simulation), direct solvers require an expensive re-factorization.

For problems on structured domains (e.g., 2D/3D regular grids), an excellent alternative is *geometric multigrid* methods. Geometric multigrid solvers perform pre-computation solely based on the geometry without knowing the linear system of interest (see Fig. 2). This enables multigrid methods to solve the system efficiently in linear time even when the system changes at each time step. Multigrid solvers already become non-trivial for unstructured grids (e.g., arbitrary triangle meshes in 2D or tetrahedral meshes in 3D), the added complexity of immersing triangle meshes in 3D has left a “black-box” multigrid solver for curved surfaces elusive until now.

In this paper, we propose a *Galerkin geometric multigrid* solver for manifold surface meshes with or without boundaries. Our key ingredient is a method for computing the *prolongation* operator based on the intrinsic geometry. Our multigrid solver achieves a better convergence rate compared to alternative multigrid methods. Replacing direct solvers with our black-box surface multigrid solver

leads to orders of magnitude speed-up. We show our method is effective in a variety of applications ranging from data smoothing to shell simulation, with linear systems of different sparsity patterns and density. Our contributions turn existing algorithms into interactive applications (e.g., Fig. 19) and shift the bottleneck away from solving linear systems (e.g., Fig. 23).

2 RELATED WORKS

Multigrid methods [Brandt 1977] have earned a reputation as one of the fastest numerical solvers for solving linear systems. On structured domains (e.g., 2D/3D grid), multigrid is very well-studied both theoretically [Hackbusch 2013; Trottenberg et al. 2000] and practically [Brandt and Livne 2011]. In graphics, multigrid has been an attractive solution for interactive and large-scale applications on structured domains, most prominently for image processing [Kazhdan and Hoppe 2008; Krishnan and Szeliski 2011] and simulating fluids on large grids [Aanjaneya et al. 2019; Lai et al. 2020; McAdams et al. 2010]. Even for problems where the original representation is unstructured, an auxiliary background grid can be introduced for multigrid to perform efficient computation and transfer the solution back to the unstructured representation. For example, one can run multigrid on a background hexahedral mesh to simulate elastic deformations [Dick et al. 2011; Zhu et al. 2010] and character skinning [McAdams et al. 2011]. Chuang et al. [2009] deploy multigrid on a background voxel grid to solve Poisson problems defined on the surface mesh. To reduce the complexity of using structured representations, adaptive multigrid methods are developed for subsurface scattering [Haber et al. 2005], smoke simulation [Setaluri et al. 2014], and other graphics applications [Kazhdan and Hoppe 2019].

Unstructured Euclidean Domains. Directly deploying multigrid to unstructured “grids” in Euclidean domains (e.g., 2D triangle meshes and 3D tetrahedral meshes) has also been an important problem for decades. The main difficulties lie in how to construct the multigrid hierarchy and how to transfer signals back-and-forth across different grid levels. In graphics, unstructured multigrid for 2D triangle meshes is widely applied to cloth simulation where the design pattern is prescribed by a 2D boundary curve. In the methods proposed in [Jeon et al. 2013; Oh et al. 2008; Wang et al. 2018], they generate the hierarchy in a coarse-to-fine manner by triangulating the 2D design pattern and then recursively subdividing it to get finer resolutions. Wang et al. [2018] generate the multigrid hierarchy from fine-to-coarse by clustering vertices on the fine mesh and re-triangulating the 2D domain. When it comes to 3D tetrahedral meshes, multigrid is commonly used to simulate deformable objects. Georgii and Westermann [2006] build the hierarchy with different tetrahedralizations of the same 3D domain. Otaduy et al. [2007] repetitively compute the offset surface of the boundary mesh, decimate the offset surface, and tetrahedral-mesh the interior to obtain the hierarchy. Sacht et al. [2015] follow a similar technique but with more elaborate and tighter fitting offsets. Adams and Demmel [1999] recursively remove the maximum independent set of vertices and tetrahedralize the interior. These unstructured multigrid methods for the Euclidean domains rely on the fact that every level in the hierarchy is a triangulation or tetrahedralization of the same space. Thus, they can easily define linear prolongation using barycentric

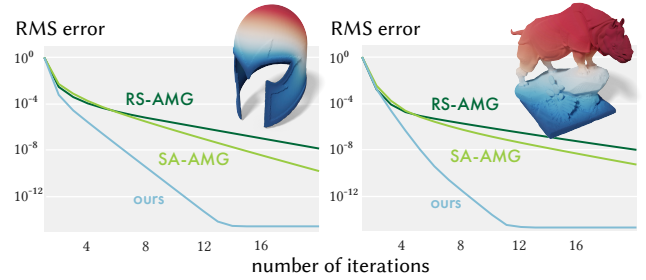


Fig. 3. When solving a Poisson problem defined on the surface mesh (top right), we demonstrate that our multigrid based on the intrinsic prolongation (blue) converges faster than the algebraic multigrid methods (green), including the classic Ruge-Stüben (RS-AMG) [Ruge and Stüben 1987] and the Smoothed Aggregation algebraic multigrid (SA-AMG) [Vanek et al. 1996]. Note that we use an off-the-shelf implementation from PyAMG [Olson and Schroder 2018] with their default multigrid hyperparameters. ©models by 3DWP (right) under CC BY-SA.

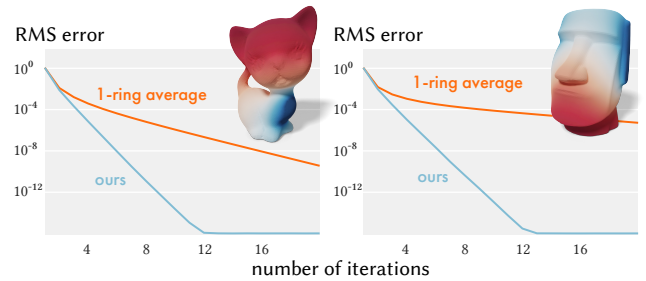


Fig. 4. Compared to the prolongation based on the vertex 1-ring average [Aksoylu et al. 2005] (orange), our prolongation (blue) leads to a faster convergence rate when solving a Poisson problem on the surface (top right).

coordinates on triangles or tetrahedra. Recently, Xian et al. [2019] propose a “mesh-free” alternative for tetrahedral meshes. They propose to use farthest point sampling to get “meshes” at coarser levels, and define the prolongation using piecewise constant interpolation which only requires the closest point query.

Algebraic Multigrid. A popular alternative to deal with unstructured meshes is to use *algebraic multigrid* [Brandt et al. 1985] which builds the hierarchy by treating the linear system matrix as a weighted graph and coarsening it. This approach makes no assumptions on the structure of the geometry. Thus, it is directly applicable to any unstructured domain. For this reason, algebraic methods are deployed to mesh deformation [Shi et al. 2006], cloth simulation [Tamstorf et al. 2015], and other graphics applications [Krishnan et al. 2013]. However, the cost of algebraic multigrid’s generality is the need to re-build the hierarchy whenever the system matrix changes (see Fig. 2). Furthermore, defining the inter-grid transfer operators for algebraic methods is more challenging and leads to worse performance compared to our method (see Fig. 3).

Curved Surfaces. When it comes to surface meshes, defining the prolongation operator becomes more challenging compared to the

Euclidean case. This is because the vertices of a high-resolution surface mesh do not lie on its coarsened counterpart, thus the straightforward barycentric computation is not immediately applicable. In the special case of subdivision surfaces where the hierarchy is given, there exists efficient geometric multigrid [Green et al. 2002] and multilevel [de Goes et al. 2016] solvers that leverage the subdivision’s regular refinement process to define the prolongation. For unstructured surface meshes, Ray and Lévy [2003] build the hierarchy based on the progressive meshes [Hoppe 1996; Kobbelt et al. 1998] and define the prolongation operator using global texture coordinates. Ni et al. [2004] (and similarly Shi et al. [2009]) find a maximum independent set of vertices to build a hierarchy and compute a prolongation operator based on a weighted average among one-ring neighboring vertices. Aksoylu et al. [2005] propose several methods for hierarchy construction based on removing the maximum independent set of vertices. Similarly, they also compute the prolongation by averaging among the one-ring neighbors. These approaches either need additional information (e.g., having subdivision connectivity or texture information) or use one-ring average (combinatorial information) to define the prolongation. But one-ring average often leads to a denser system because it requires on average 6 vertices to interpolate the result on a vertex, in contrast to 3 when using the barycentric interpolation. Performance wise, in Fig. 4 we show that our prolongation leads to a better convergence compared to the multigrid based on averaging one-ring vertices.

Our method computes the prolongation based on *intrinsic geometry*. This enables us to define a linear prolongation simply using the barycentric coordinates, echoing the success of using barycentric coordinates in the Euclidean case. Furthermore, our approach allows one to plug-and-play different decimation strategies to construct multigrid hierarchies (see Fig. 13). This flexibility allows one to pick a well-suited decimation method for their tasks of interest.

Purely algebraic direct solvers (e.g., sparse Cholesky) are the *de facto* standard in geometry processing due to their reliability, scalability (as memory allows), and precision. Factorizations can be reused for changing right-hand sides (see Fig. 2), but trouble arises for the myriad applications where the system matrix also changes. Special classes of sparse changes can be efficiently executed: low-rank updates [Chen et al. 2008; Cheshmi et al. 2020] or partial re-factorizations [Herholz and Alexa 2018; Herholz and Sorkine-Hornung 2020]. However, many other scenarios trigger full numerical refactorization, such as changing parameters in a multi-objective optimization and Hessian updates for Newton’s method. Due to the overwhelming popularity of Cholesky factorization, we focus on it for many of our head-to-head comparisons.

3 MULTIGRID OVERVIEW

Multigrid is a type of iterative solver that is scalable to solve large linear systems $Ax = b$. In this paper, we propose a novel geometric multigrid method for solving linear systems defined on curved surfaces, represented as irregular triangle meshes. We refer readers to excellent resources on multigrid [Brandt and Livne 2011; Trottenberg et al. 2000], here we give only the essential components needed to understand our method. Note that we use “grid” or “mesh” interchangeably to denote the underlying geometry.

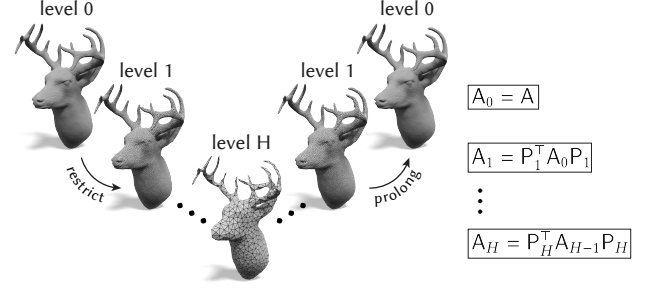


Fig. 5. The multigrid V-cycle proceeds from the finest grid (level 0) to the coarsest grid (level H) and goes back up to the finest grid again. On each level (except for the coarsest level), we pre-relax the solution, *restrict* it to the coarser grid, compute the correction, *prolong* the correction back to the finer level, post-relax the correction, and then add the correction to the current solution. Our approach belongs to the *Galerkin multigrid* where we define the system matrix at a coarser level as $A_h = P_h^T A_{h-1} P_h$. ©model by Takeshi Murata under CC BY-SA.

Multigrid methods solve a linear system in a hierarchical manner by employing two complementary processes: *relaxation* and *coarse-grid correction*. Relaxation involves applying classic iterative methods to correct the high-frequency error between the current solution and the exact solution of the system. Coarse-grid correction involves transferring the low-frequency error to a coarser mesh through *restriction*, solving a coarse-grid system of equations, then transferring the correction back to the finer mesh via *prolongation* (a.k.a. *interpolation*). This process of going from the fine grid to the coarse grid and then back to the fine grid is called the *V-cycle* (see Fig. 5). How to build the multigrid hierarchy and how to transfer information back and forth between grid levels are keys to determine the efficiency of a multigrid method.

Our method belongs to *geometric multigrid* based on the *Galerkin coarse grid approximation*. Geometric multigrid is a class of multigrid methods that builds the hierarchy purely based on the geometry, requiring no knowledge about the linear system. Galerkin coarse grid approximation builds the system matrix A_c on the coarsened mesh from the system matrix A on the original mesh as

$$A_c = RAP, \quad (1)$$

where R is the restriction operator to transfer signals from the fine mesh to the coarsened mesh and P is the prolongation operator to transfer signals from coarse to fine. When A is symmetric, many methods often define $R = P^T$. Thus, defining the prolongation operator P is extremely critical for Galerkin multigrid because it determines both the quality of the coarsened linear system $P^T A P$ and the quality of the inter-grid transfer (restriction P^T and prolongation P). In Alg. 2, we provide pseudo code of the Galerkin multigrid V-cycle where P plays a crucial role in the entire algorithm. An ideal prolongation must accurately interpolate smooth functions (low distortion) to ensure fast convergence. The prolongation also needs to be sparse to enhance the solver efficiency at coarser levels.

Defining a prolongation that satisfies these properties is well-studied on structured domains, but extending to unstructured curved surfaces remain a challenging problem until now. In this paper, we

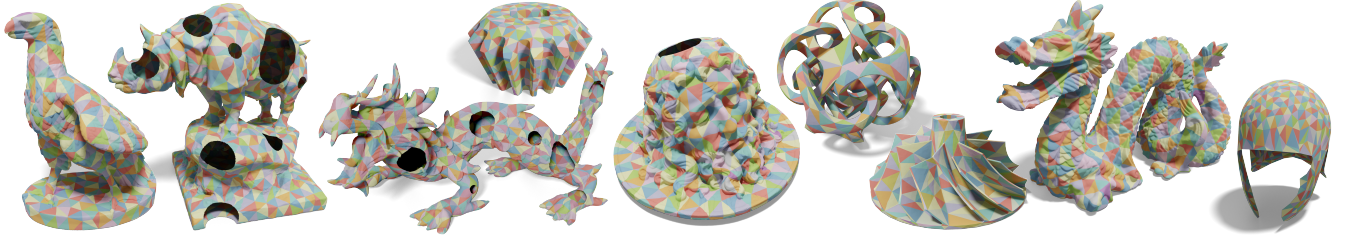


Fig. 6. We visualize the bijective map computed using our method by coloring the high-resolution shape using the coarsened triangulation (as different colors). Our method is applicable to man-made objects, organic shapes, high-genus shapes, and meshes with boundaries. ©models by Oliver Laric (left 1, 5, 8) under CC BY-NC-SA and Landru (left 7) under CC BY.

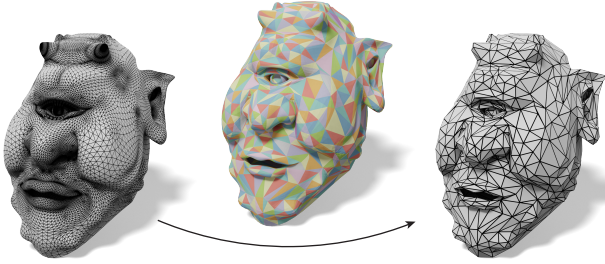


Fig. 7. Given a high-resolution shape (left) and its coarsened counterpart (right), we compute a bijective map between the two so that for any given point on the fine mesh we can compute its corresponding barycentric coordinates on the coarsened mesh, and vice versa. We visualize the map by coloring the coarse triangulation on top of the high-resolution model.

use successive self-parameterization to compute a prolongation operator P for curved surfaces based on the intrinsic geometry. Our novel joint flattening further reduces the distortion caused by the parameterization and our extension to meshes with boundaries broadens the applicability of our method to many complex geometries. When deploying our prolongation to the Galerkin multigrid framework, our method achieves better convergence over alternative multigrid methods for curved surfaces.

4 INTRINSIC PROLONGATION

The central ingredient of Galerkin multigrid is the prolongation operator to interpolate signals from a coarsened mesh to its fine version. We compute the prolongation by maintaining an intrinsic parametrization, as opposed to extrinsic prolongation based on 3D spatial coordinates (cf. [Liu and Jacobson 2019; Manson and Schaefer 2011]). Specifically, we parameterize the high-resolution mesh using the coarsened mesh to obtain a bijective map between the two (see Fig. 7). Given a point on the high-resolution mesh, we can obtain its corresponding barycentric coordinates on the low-resolution mesh, and vice versa. We can then assemble a linear prolongation operator based on the barycentric information.

We compute the bijective map using *successive self-parameterization*. The key idea is to successively build a bijective map for each decimation step and assemble the full map via compositing all the maps. Our method for computing the successive parameterization is based on the framework of [Liu et al. 2020], which can be perceived as a

combination of Lee et al. [1998] and Cohen et al. [2003]. The key differences of our method compared to [Liu et al. 2020] are a novel *joint flattening* method (see Sec. 4.2) to further reduce the distortion and a generalization to meshes with boundaries (see Sec. 4.3). For the sake of reproducibility, we reiterate the main ideas of successive self-parameterization here.

4.1 Successive Self-Parameterization

Let \mathcal{M}^0 be the input fine mesh with/without boundary, and \mathcal{M}^0 is successively simplified into a series of meshes \mathcal{M}^l with $0 \leq l \leq L$ until the coarsest mesh \mathcal{M}^L . For each pair of meshes $\mathcal{M}^l, \mathcal{M}^{l+1}$, we use $f_{l+1}^l : \mathcal{M}^l \rightarrow \mathcal{M}^{l+1}$ to denote the bijective map between them. The main idea is to compute each f_{l+1}^l on-the-fly during the decimation process and composite all the maps between subsequent levels to obtain the final map $f_L^0 : \mathcal{M}^0 \rightarrow \mathcal{M}^L$ as

$$f_L^0 = f_{L+1}^L \circ \dots \circ f_1^0. \quad (2)$$

Thus, the question boils down to the computation of the individual maps f_{l+1}^l before and after a single edge collapse.

For each edge collapse, the triangulation mostly remains the same except for the neighborhood of the collapsed edge. Thus, computing f_{l+1}^l only requires to figure out the mapping within the edge 1-ring neighborhood. Let $\mathcal{N}^l(k)$ be the neighboring vertices of a vertex k (including vertex k itself) at level l and let $\mathcal{N}^l(i, j) = \mathcal{N}^l(i) \cup \mathcal{N}^l(j)$ denote the neighboring vertices of an edge i, j . The key observation is that the boundary vertices of $\mathcal{N}^l(i, j)$ before the collapse are the same as the boundary vertices of $\mathcal{N}^{l+1}(k)$ after the collapse, where k is the newly inserted vertex after collapsing edge i, j . Hence, we compute a shared UV-parameterization for the patches enclosed by $\mathcal{N}^l(i, j)$ and $\mathcal{N}^{l+1}(k)$ with the same boundary curve. Then, for any given point $p^l \in \mathcal{M}^l$ (represented in barycentric coordinates), we can utilize the shared UV parameterization to map p^l to its corresponding barycentric coordinates $p^{l+1} \in \mathcal{M}^{l+1}$ and vice-versa, as shown in Fig. 8.

4.2 Joint Flattening

The base method proposed by Liu et al. [2020] ensures boundary consistency by first flattening the edge 1-ring $\mathcal{N}^l(i, j)$ and setting the boundary vertices as hard constraints when flattening the vertex 1-ring $\mathcal{N}^{l+1}(k)$ after the collapse. Although this method can

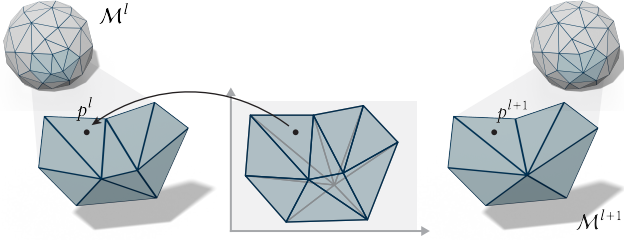


Fig. 8. Since both the edge 1-ring before the collapse (left) and the vertex 1-ring after the collapse (right) are mapped to the same 2D domain with a consistent boundary curve (middle), we can easily use the shared UV space to map a point back and forth between M^l and M^{l+1} .

ensure boundary consistency, it always favors minimizing the distortion of $N^l(i, j)$ and creates larger distortion when flattening $N^{l+1}(k)$.

We instead compute the shared UV-parameterization by *jointly* minimizing a distortion energy E defined on the edge 1-ring $N^l(i, j)$ before the collapse and the vertex 1-ring $N^{l+1}(k)$ after the collapse while ensuring boundary consistency. In Fig. 9, we demonstrate that our joint flattening results in a parameterization with less distortion compared to the method by Liu et al. [2020].

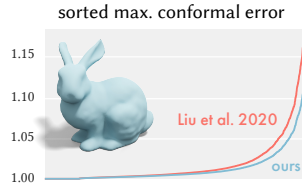


Fig. 9. For each edge 1-ring on this bunny mesh, we collapse the edge and flatten the patch using the method of [Liu et al. 2020] and our joint flattening method. We visualize the sorted quasiconformal distortion among all the 1-rings and demonstrate that our method (blue) leads to less distortion.

For the notational convenience, we use V^l, F^l to denote the vertices and faces of the local patch within $N^l(i, j)$ before the collapse, and V^{l+1}, F^{l+1} to denote the vertices and faces of the local patch within $N^{l+1}(k)$ after the collapse. We then write the joint energy optimization problem as

$$\underset{U^l, U^{l+1}}{\text{minimize}} \quad E(V^l, F^l, U^l) + E(V^{l+1}, F^{l+1}, U^{l+1}) \quad (3)$$

$$\text{subject to } u_b^l = u_b^{l+1} \quad (4)$$

where we use $U^l \in \mathbb{R}^{|V^l| \times 2}$ to represent the UV locations of V^l at level l and each $u_i^l \in \mathbb{R}^2$ denotes a UV-vertex position. We also use u_b^l, u_b^{l+1} to represent the boundary vertices of $N^l(i, j)$ and $N^{l+1}(k)$, respectively.

In order to handle the constraints, we introduce a joint variable $U = U^l \cup U^{l+1}$ (see Fig. 10) to incorporate the equalities into the degrees of freedom and turn Eq. (3) into an unconstrained problem

$$\min_U \quad E(V^l, F^l, U) + E(V^{l+1}, F^{l+1}, U) \quad (5)$$

Introducing the joint variable U allows us to minimize the distortion energy for the patch before and after the collapse simultaneously.

4.3 Boundary Edges

The creation of the joint variable in Fig. 10 is only applicable when the collapsed edge lies fully in the interior of the triangle mesh. For boundary edges, different treatment is required to ensure the

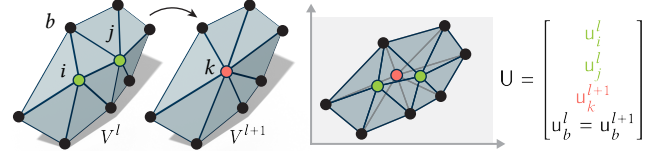


Fig. 10. Ensuring bijectivity between V^l and V^{l+1} requires their boundary vertices (black b) to have the same UV positions (right). We handle this constraint by introducing a joint variable U which contains shared degrees of freedom on the boundary.

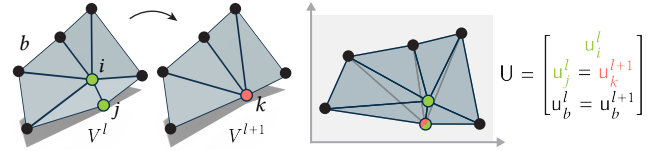


Fig. 11. When one of the edge vertices is on the boundary (vertex j in this case), we have to also constrain the vertex j and k to have the same UV location to ensure bijectivity, as shown in the joint variable U on the right.

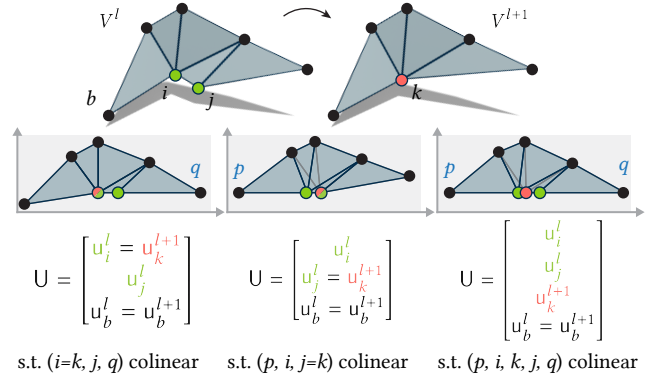


Fig. 12. When the edge i, j is a boundary edge, we consider three cases: $u_i^l = u_k^{l+1}$ (left), $u_j^l = u_k^{l+1}$ (middle), and vertex i, j, k are colinear in the UV space (right). To ensure the boundary curves remain consistent, these cases result in three different sets of colinearity constraints (see the bottom row), where we use q to represent the *next* boundary vertex of the edge i, j and we use p to represent the *previous* boundary vertex of the edge i, j .

shared parameterization has a consistent boundary curve in order to preserve the bijectivity (cf. [Liu et al. 2017]).

In the case where one of the two incident vertices lies on the boundary, we create a joint variable which snaps the UV position of the other (interior) vertex to the boundary (see Fig. 11). Note that we only perform this snapping operation in the parameterization domain, their corresponding vertices v_j, v_k in \mathbb{R}^3 are still placed at the locations which minimize the decimation error metric (e.g., appearance preservation).

In the case where both edge vertices are on the boundary, we determine the joint variable U by choosing best of three possible choices. Suppose the boundary edge i, j is collapsed to a boundary vertex k , we consider the cases where (1) vertex k lies on vertex i , (2) vertex k lies on vertex j , and (3) vertex k lies on the line defined by i, j . Even though case (1), (2) seem unnecessary when we have

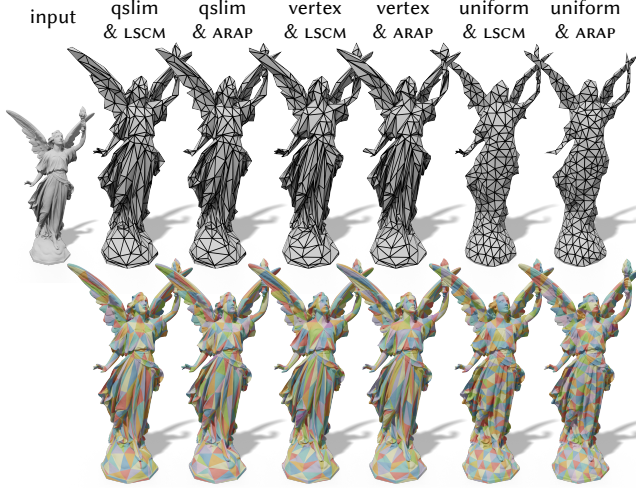


Fig. 13. Our method allows to plug-and-play different decimation strategies and parameterization algorithms. In these examples, we decimate the model using qslim [Garland and Heckbert 1997], vertex removal via half-edge collapse, and uniform mid-point edge collapse. We use ARAP [Liu et al. 2008] and LSCM [Lévy et al. 2002] as the parameterization algorithms. The influence of these combinations to the solver convergence is shown in Fig. 14.

case (3), these cases end up with different sets of constraints in the joint flattening optimization. Thus, we consider all three cases and take the one with the minimum energy value. In Fig. 12, we show how we group variables for the three cases and their corresponding colinearity constraints to maintain the same boundary curve. We impose the colinearity via adding Dirichlet constraints $(u_i)_y = 0$ for all the vertices i that are colinear.

4.4 Decimation Strategies & Distortion Energies

Our joint flattening makes no assumption on the edge collapse algorithm in use. For instance, one could use the *quadric error edge collapse* (qslim) [Garland and Heckbert 1997] to preserve the appearance, *mid-point edge collapse* to encourage the coarse triangulation to be more uniformly distributed, and the *vertex removal* (via half-edge collapses [Kobbelt et al. 1998]) to ensure that the vertices on the coarsened mesh are a subset of the fine vertices.

The distortion energy E in Eq. (5) provides another design parameter. In Fig. 13, we demonstrate the flexibility of our joint flattening by minimizing the *as-rigid-as-possible* (ARAP) [Liu et al. 2008] and the *least square conformal map* (LSCM) [Lévy et al. 2002] energies.

Depending on the intended application, different combinations of the decimation strategy and the parameterization algorithm may lead to different performance. For instance, in Fig. 14 we compare the convergence behavior of our Galerkin multigrid solvers constructed using these combinations. In our experiments, using the uniform decimation with LSCM leads to the best performance among these options. Other options for minimizing the distortion [Khodakovsky et al. 2003], computing the map [Friedel et al. 2004; Guskov et al. 2002, 2000], and decimation strategies (e.g., [Trettner and Kobbelt 2020]) seem attractive to combine with our joint flattening. It is however more challenging and thus left as a future work.

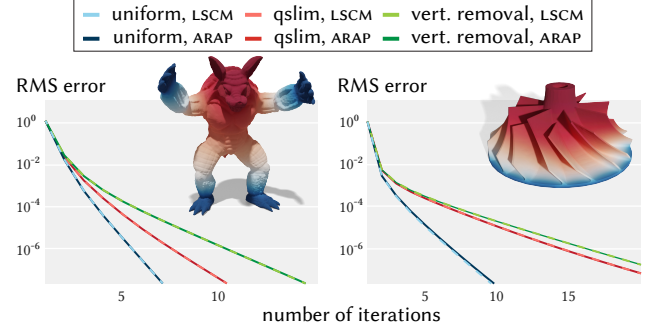


Fig. 14. Different combinations of the decimation and the parameterization methods lead to different performance in down-stream applications. For example, in the context of multigrid solvers on a Poisson problem, the uniform edge decimation with LSCM leads to a better convergence rate.

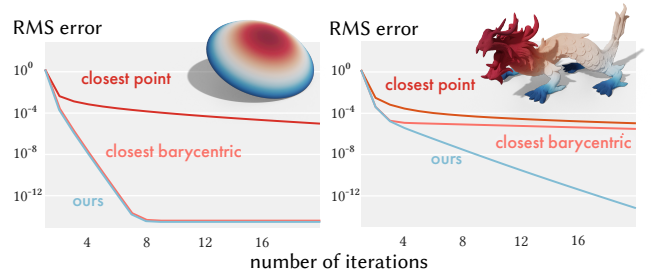


Fig. 15. We compare our intrinsic prolongation with naive extrinsic prolongations based on the closest-point projection. When evaluating on a simple shape (left), most methods can converge; when evaluating on a complex shape (right), only our intrinsic prolongation converges. ©model by Oliver Laric (right) under CC BY-NC-SA.

4.5 Prolongation Operator

The above discussion computes a bijective map between a pair of meshes that undergoes a single edge collapse. We can easily extend the method to compute a map between two consecutive multigrid levels via composition (see Eq. (2)). Given this information, we can now compute a prolongation operator for surface multigrid.

We choose linear interpolation as our prolongation operator because it is sufficient for the convergence of the second-order PDEs typically employed in computer graphics [Hemker 1990]. Although some of our experiments consider higher order PDEs, many of them are reduced to low-order systems in practice via mixed finite elements [Jacobson et al. 2010]. Empirically, we find that linear prolongation still converges in most cases.

Our linear prolongation P is a tall matrix whose size is the number of fine vertices by the number of coarse vertices. Each row of P contains 3 non-zeros corresponding to the barycentric coordinates of the fine vertex with respect to the vertices of the coarse triangle containing it. We evaluate the quality of our prolongation on solving Poisson problems on a variety meshes. We demonstrate that our intrinsic prolongation leads to faster convergence compared to the naive closest point projection (Fig. 15), an extrinsic bijective projection by Jiang et al. [2020] (Fig. 16), vertex 1-ring average [Aksoylu et al. 2005] (Fig. 4), and algebraic multigrid prolongations (Fig. 3).

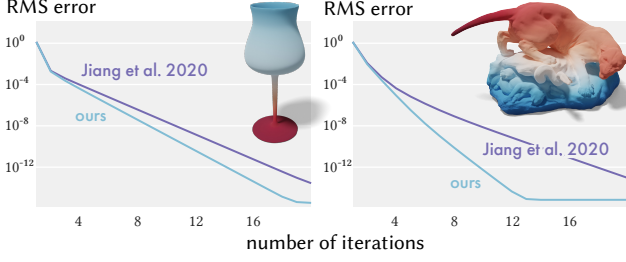


Fig. 16. Compared to an extrinsic prolongation proposed by Jiang et al. [2020], our prolongation leads to faster convergence.

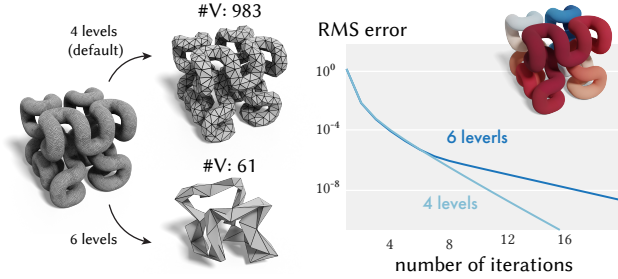


Fig. 17. We compare the multigrid convergence between our default parameters (resulting in four multigrid levels) and an extreme coarsening (6 levels). We demonstrate that the extreme coarsening hurts the performance (right) because the coarsest mesh fails to represent the target solution.

5 MULTIGRID IMPLEMENTATION

Switching from direct solvers to our multigrid method is very simple. In Alg. 1 we summarize how one can implement a Galerkin geometric multigrid method to solve a linear system. The key difference is that the pre-computation happens right after loading the input mesh. This pre-computation takes the vertex and face lists as the inputs, and outputs a series of prolongation operators P_1, \dots, P_H for different levels on the hierarchy. After building the linear system A, b , one can run the V-cycle until getting the desired accuracy.

In Alg. 2, we summarize the pseudo code of the V-cycle algorithm which consists of two procedures: relaxation and coarse-grid correction. For the relaxation step, we use the standard serial Gauss-Seidel method. In the coarse-grid correction step, the process is well-defined given the prolongation operator P . We start by restricting the residual to the coarser level via P^T , solving a coarsened linear system with the left-hand-side defined as $P^T A P$, prolonging the low-res solution back to the fine domain using P , and using it to update the current high-res solution. We can further accelerate the computation by storing the system matrix hierarchy $A_{h+1} = P_{h+1}^T A_h P_{h+1}$ to save some redundant computation.

In terms of hyperparameters of our multigrid method, we conduct an ablation study summarized in App. D. In each V-cycle, we use the Gauss-Seidel relaxation with 2 pre- and post-relaxation iterations. Our default setup coarsens the geometry down to 0.25 of the number of vertices at its previous level until we reach the coarsest mesh with no lesser than 500 vertices. Note that we do not recommend to coarsen the mesh to an extreme. In Fig. 17, we show that an extremely aggressive coarsening often hurts the performance because

Algorithm 1: Galerkin Surface Multigrid Solver

```

1.  $V, F \leftarrow \text{load triangle mesh}$ 
2.  $P_1, \dots, P_H \leftarrow \text{precompute multigrid hierarchy } (V, F)$ 
3.  $A \leftarrow \text{build left-hand-side}$ 
4.  $b \leftarrow \text{build right-hand-side}$ 
5. initialize a solution  $x$ 
6. while error is larger than  $\epsilon$  do
7.    $x \leftarrow V\text{-cycle}(A, x, b, 0)$  // see Alg. 2
```

Algorithm 2: $x_{\text{new}} = V\text{-cycle}(A, x_{\text{old}}, b, h)$

```

Param.:  $P_1, P_2, \dots, P_H$ , // hierarchy of prolongations
           $\mu_{\text{pre}}, \mu_{\text{post}}$  // pre- and post-relaxation iterations
Input :  $A$ , // left-hand-side system matrix
           $x_{\text{old}}$ , // current solution
           $b$ , // right-hand-side of the linear system
           $h$ , // current multigrid level
Output:  $x_{\text{new}}$  // new solution

1. if  $h$  is not the coarsest level  $H$  then
2.   // PRE-RELAXATION
3.    $x'_{\text{old}} \leftarrow \text{Relaxation}(A, x_{\text{old}}, b, \mu_{\text{pre}})$ 
4.   // COARSE-GRID CORRECTION
5.    $r_{h+1} \leftarrow P_{h+1}^T (b - A x'_{\text{old}})$  // restrict residual
6.    $c_{h+1} \leftarrow V\text{-cycle}(P_{h+1}^T A P_{h+1}, 0, r_{h+1}, h+1)$ 
7.    $c_h \leftarrow P_{h+1} c_{h+1}$  // prolong correction
8.    $x'_{\text{new}} \leftarrow x_{\text{old}} + c_h$  // update solution
9.   // POST-RELAXATION
10.   $x_{\text{new}} \leftarrow \text{Relaxation}(A, x'_{\text{new}}, b, \mu_{\text{post}})$ 
11. else
12.   $\text{solve } A x_{\text{new}} = b$  // direct solve
13. return  $x_{\text{new}}$ 
```

the coarsest mesh fails to represent the target solution. In terms of stopping criteria, the accuracy ϵ that allows us to get visually distinguishable results compared to the ground truth depends on the problem and the size of the mesh. In our experiments, we set it to be $10^{-3} \geq \epsilon \geq 10^{-5}$. Our experiments suggest that the optimal set of parameters that minimizes the wall-clock runtime depends on the geometry and the PDE of interest. But we use our default parameters for all our experiments in Sec. 6 for consistency.

We implement our algorithm in C++ with Eigen and evaluate our method on a MacBook Pro with an Intel i5 2.3GHz processor. In comparison with the Cholesky solver where pre-factorization is required whenever the system matrix A is changed, our multigrid solver leads to orders of magnitude speed-ups (see Fig. 21).

Boundary conditions. Our implementation currently supports natural boundary conditions (Fig. 21 right), zero Neumann boundary conditions (Fig. 21 left), and the Dirichlet boundary condition. We handle the Dirichlet constraints by reformulating the system using

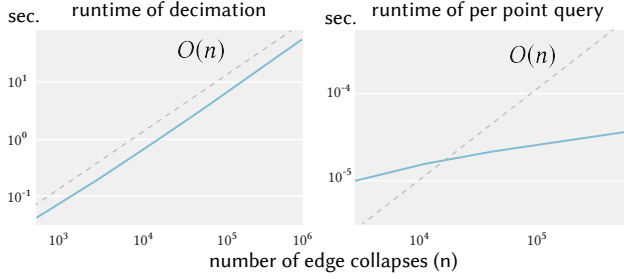


Fig. 18. On the left, we report the runtime of our successive self-parameterization for constructing the bijective map. On the right, we report the query time of a single point through the bijective map.

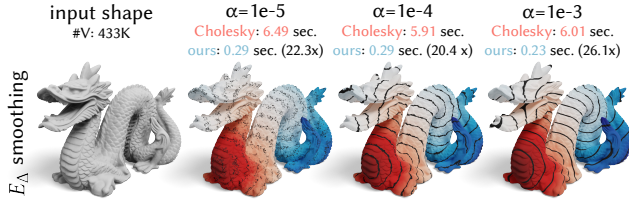


Fig. 19. In the data smoothing application, one would adjust the smoothness parameter α until getting the desired smoothness. Using direct solvers (e.g., Cholesky) would need to recompute the expensive pre-factorization, but our multigrid solver can reuse the same hierarchy and leads to interactive performance.

only the unknown variables. This results in a reduced and unconstrained linear system, allowing us to solve it as usual. For more details, please refer to App. B.

Successive Self-Parameterization. We report the runtime of our pre-computation (self-parameterization and the query) in Fig. 18 and detail the implementation in App. C. Note that this pre-computation solely depends on the geometry. We only need to do this computation once for each shape and we can reuse the same hierarchy for many different linear systems. Thus, in our runtime comparisons in Sec. 6, we do not include the runtime of this pre-computation.

6 APPLICATIONS

We evaluate our method on a variety of geometry processing applications that involve solving linear systems as a subroutine. We especially focus on the case where the system matrix A is changing due to different time steps (e.g., simulation) or user interactions (e.g., data smoothing). In our experiments, we ignore the multigrid pre-computation and compare our multigrid V-cycle (in blue) against the runtime of both the factorization and the solving time combined of the Cholesky solver (in red) because these steps are required when both A, b are changing. We also pick the applications that involve different system matrices with different sparsity patterns. This includes the cotangent Laplacian (1-ring sparsity), the Bilaplacian (2-ring sparsity), the squared Hessian (2-ring sparsity) [Stein et al. 2020], a system matrix derived from Lagrange multipliers [Azencot et al. 2015], and also the Hessian matrices from shell simulation which has $3|V|$ -by- $3|V|$ dimensionality.

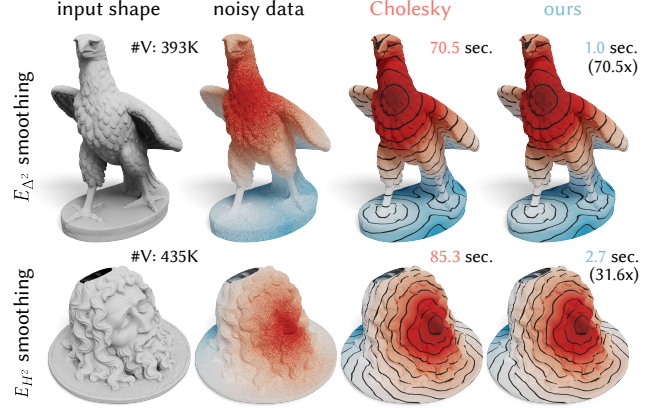


Fig. 20. We evaluate our method on data smoothing with different smoothness energies, including the Bilaplacian E_{Δ^2} and the squared Hessian E_{H^2} . Our method is orders of magnitude faster than the direct solver.

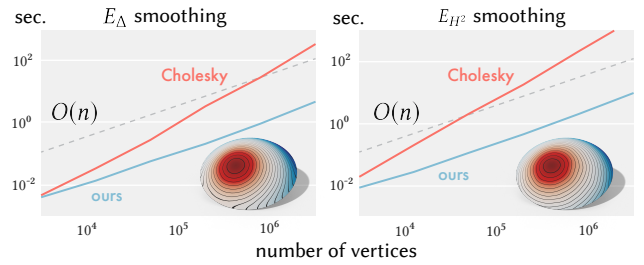


Fig. 21. We compare the runtime of our multigrid solver against the Cholesky solver on smoothing the noisy data on a sphere cap at different resolutions until reaching a sufficiently small mean squared error (visually indistinguishable). We evaluate the smoothing with the Dirichlet energy E_{Δ} (1-ring sparsity) and with the squared Hessian energy E_{H^2} [Stein et al. 2020] (2-ring sparsity). Our method is asymptotically faster than the direct solver. On meshes with 200K vertices and 3 million vertices (using E_{H^2}), a serial implementation of our method is 39 \times and 231 \times faster, respectively.

Data smoothing. Smoothing data on the surface is a fundamental task in geometry processing. We often treat it as an energy minimization problem

$$x = \arg \min_x \alpha E_s(x) + (1 - \alpha) \int_{\Omega} \|x - f\|^2 dx, \quad (6)$$

where α is the parameter controlling the smoothness, f is the input noisy function, and E_s is an energy of choice, measuring the smoothness of the output signal x . As a different input f may contain a different amount of noise, one would typically adjust the α or the smoothness energy E_s until getting the desired smoothness. However, these adjustments boil down to solving a different linear system. When using direct solvers, this requires to recompute the factorization in order to solve the system. In comparison, using our multigrid allows one to reuse the same precomputed multigrid hierarchy and leads to orders of magnitude speed-ups (see Fig. 19). We evaluate our method on different smoothness energies, including the Dirichlet energy E_{Δ} (Fig. 19), the squared Laplacian energy E_{Δ^2} (Fig. 20 top), and the squared Hessian energy E_{H^2} [Stein et al. 2020]

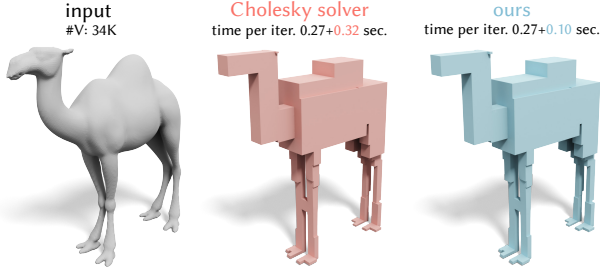


Fig. 22. We compare our multigrid against the direct solver on the polycube deformation proposed by [Zhao et al. 2017]. Although the vertex positions are changed at every iteration, we can still reuse the precomputed multigrid hierarchy because the connectivity remains the same. We report the runtime of other steps in the algorithm in black and the runtime for solving linear systems in red and in blue.

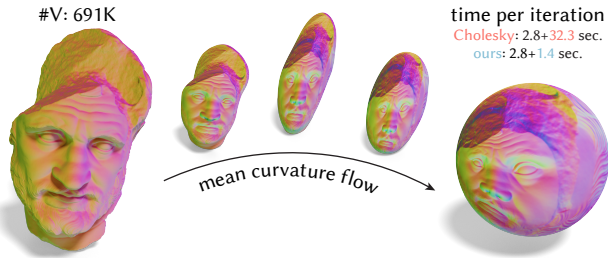


Fig. 23. Running the mean curvature flow [Kazhdan et al. 2012] requires to update the system matrix at every step according to the mass matrix of the current mesh. By reusing the hierarchy computed on the input shape (left), our multigrid method is orders of magnitude faster than the direct solver. We report the runtime of other subroutines in black and the time for solving the linear system in red (direct solver) and in blue (our multigrid). ©model by Oliver Laric under CC BY-NC-SA.

(Fig. 20 bottom). In Fig. 21, we quantitatively evaluate the runtime on the same shape at different resolutions obtained via subdivision. On a mesh with millions of vertices, our approach has over 100× speed-ups. With our multigrid setup, the precomputed prolongation operator can be reused not only when changing the value of α (full rank update to A), but also when swapping between energies E_s .

Mesh deformation. We also evaluate our method on mesh deformations to demonstrate that even though the vertex positions have changed, as long as the connectivity of the mesh remains the same, we can still reuse the same multigrid hierarchy computed on the rest mesh. One possible intuition is to view the deformation field on vertices as a function on the rest shape. Thus, a hierarchy built on the rest shape could still be used to compute the deformation “function”. In Fig. 22, we evaluate our method on a polycube deformation method proposed by [Zhao et al. 2017] whose system matrix is re-built at every iteration based on the current deformed mesh. Our method accelerates the algorithm by $3.2 \times$ on a relatively low-resolution mesh. In Fig. 23, we replace the Cholesky solver with our method on a mean curvature flow method proposed in [Kazhdan et al. 2012] and achieve $23 \times$ speedup.

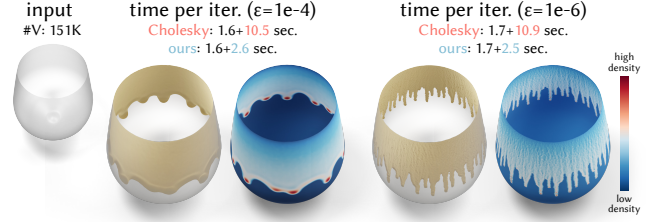


Fig. 24. The surface fluid simulation [Azencot et al. 2015] involves solving different linear systems at each time step. Our method reuses the precomputed hierarchy and leads to a faster solver in contrast to the direct solver. We split the runtime of other procedures (black) and the runtime of solving the linear system (red and blue). Note that this runtime comparison is in MATLAB using the original implementation from [Azencot et al. 2015].

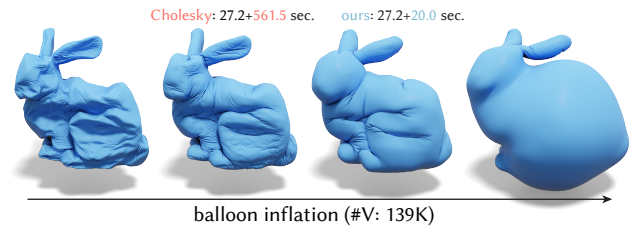


Fig. 25. Replacing the Cholesky solver with our surface multigrid method, we can accelerate the linear solve part in the balloon simulation proposed by [Skouras et al. 2012] by $28 \times$ so that solving linear systems becomes no longer the bottleneck of the algorithm.

In many simulation algorithms, the system matrix A changes at every time step. In Fig. 24, we demonstrate the speed-up of our multigrid solver on a surface fluid simulation [Azencot et al. 2015]. Note that the surface fluid simulation is evaluated in MATLAB (for both the direct solver and our multigrid) respecting the original implementation. In Fig. 25, we evaluate our method on a balloon simulation method proposed by [Skouras et al. 2012]. Due to the speedup of our multigrid method, we shift the bottleneck of balloon simulation away from solving linear systems.

7 DISCUSSION

In our experiments in Sec. 6, we evaluate our runtime using a simple serial implementation of our method. In Table 1, we further provide a detailed runtime decomposition of the experiment in Fig. 23 as a representative example. We can observe that preparing the matrix hierarchy P^TAP and doing the relaxation take most of the time when solving a linear system. Thus, we can achieve even more speedup if we leverage the structure of the problem when computing the matrix hierarchy, such as the data smoothing detailed in App. E, or a parallel implementation of the entire V-cycle. To validate our hypothesis,

Table 1. Multigrid runtime.

profile (sec.)	Fig. 23
precompute	50.6
total solve time	1.44
1. prepare P^TAP	0.72
2. relaxation	0.38
3. prolong & restrict	0.10
4. get residual norm	0.10
5. others	0.14

our initial attempt uses CPU to parallelize the Gauss-Seidel method based on graph coloring. This reduces the runtime of our relaxation from 0.38 seconds down to 0.17 seconds (2.2 \times speedup) for the experiment in Fig. 23. Similarly, for the top and the bottom examples in Fig. 20, the fast Gauss-Seidel accelerates the relaxation by 1.8 \times and 3.3 \times repetitively. We provide details about our graph coloring Gauss-Seidel in the App. A for completeness. An even higher speed-up can be expected via a GPU implementation of the Gauss-Seidel relaxation (cf. [Fratarcangeli et al. 2016]). Besides the Gauss-Seidel relaxation, parallelizing the entire solver could also be an interesting future direction to accelerate our method.

8 LIMITATIONS & FUTURE WORK

We present a geometric multigrid solver for triangulated curved surfaces. Multigrid methods are asymptotically faster than direct solvers, thus it offers a promising direction for *scalable* geometry processing. Our multigrid method can obtain a fast approximation of the solution with orders of magnitude speedup. However, obtaining a highly accurate solution would require more iterations which results in a less significant speed-ups. For higher-order problems, our method may not converge to high accuracy because our choice of linear interpolation is insufficient [Hemker 1990]. Thus, exploring high-order prolongation (e.g., subdivision barycentric coordinates [Anisimov et al. 2016]) or learning-based prolongation (e.g. [Katrutsa et al. 2020]) would also be valuable directions to improve the solver. Another interesting direction to improve the solver is to use our multigrid solver as the pre-conditioner for other solvers such as the *conjugate gradient* method.

Developing a reliable and robust surface multigrid solver would be an important next step. Our current solver is more sensitive to the triangle quality of the input mesh compared to the existing direct solver. In our experiments, we remesh troublesome input shapes using available methods [Hu et al. 2020; Jakob et al. 2015; Schmidt and Singh 2010]. A better future approach would be extending our self-parameterization to the entire remeshing process, to maintain bijectivity from the remeshed object to the input mesh. Having a deeper understanding of the relationship between the convergence and mesh quality would give insights towards developing a suitable remeshing algorithm for surface multigrid solvers. Achieving this may also require theoretical tools to estimate the convergence property, such as extending the *Local Fourier Analysis* from subdivision meshes [Gaspar et al. 2009] to generic unstructured meshes. Once surface multigrid has become a reliable solver for linear systems on manifold meshes, generalizing it to non-manifolds or point clouds would be another exciting future direction.

Another avenue for future work is to further optimize each component of the prolongation construction and multigrid solver routines. Although our method outputs a bijective map in most cases, bijectivity is not guaranteed. A more rigorous analysis is required to identify potential edge cases that may result in non-bijective maps. Currently, we use off-the-shelf simplification and distortion objectives (as-rigid-as-possible [Liu et al. 2008] and conformal [Lévy et al. 2002] energies), but these methods that are designed for other purposes may not be the optimal ones for surface multigrid methods. For instance, we notice that the distortion in the self-parameterization is

not closely correlated to the convergence of our multigrid solver (see Fig. 14). We however use the off-the-shelf parameterization energy designed to measure the distortion in our multigrid solver. Developing simplification and parameterization methods tailored-made for multigrid solver performance could further improve eventual solver speed.

The relationship between multigrid convergence and bijectivity requires a deeper understanding. Although we empirically demonstrate the superior performance of our prolongation compared to other non-bijective prolongations, bijectivity is not required for a multigrid method to converge. In our construction, we even pay the price of high distortion to achieve bijectivity along the boundary (zoom in Fig. 6). Thus, a deeper understanding of the connections between distortion, bijectivity, and multigrid convergence is important to reach optimal performance.

ACKNOWLEDGMENTS

Our research is funded in part by New Frontiers of Research Fund (NFRFE-201), German-Israeli Foundation for Scientific Research and Development (I-1339-407.6/2016), European Research Council (714776 OPREP), NSERC Discovery (RGPIN2017-05235, RGPAS-2017-507938), Israel Science Foundation (504/16), the Ontario Early Research Award program, the Canada Research Chairs Program, the Fields Centre for Quantitative Analysis and Modelling and gifts by Adobe Systems, Autodesk and MESH Inc. We thank Uri Ascher, Tiantian Liu, and Irad Yavneh for sharing inspiring lessons on multigrid methods; Zhongshi Jiang for helps on experiments. We thank members of Dynamic Graphics Project at the University of Toronto; Michael Tao and Kazem Cheshmi for valuable discussions; Benjamin Chislett for proofreading; and John Hancock for the IT support. We thank all the artists for sharing a rich variety of 3D models.

REFERENCES

- Mridul Aanjaneya, Chengguizi Han, Ryan Goldade, and Christopher Batty. 2019. An Efficient Geometric Multigrid Solver for Viscous Liquids. *Proceedings of the ACM in Computer Graphics and Interactive Techniques* (2019).
- Mark Adams and Jim Demmel. 1999. Parallel Multigrid Solver for 3D Unstructured Finite Element Problems. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1999, November 13-19, 1999, Portland, Oregon, USA*. ACM, 27.
- Burak Aksoylu, Andrei Khodakovskiy, and Peter Schröder. 2005. Multilevel Solvers for Unstructured Surface Meshes. *SIAM J. Sci. Comput.* 26, 4 (2005), 1146–1165.
- Dmitry Anisimov, Chongyang Deng, and Kai Hormann. 2016. Subdividing barycentric coordinates. *Computer Aided Geometric Design* 43 (2016), 172–185.
- Omri Azencot, Orestis Vantzos, Max Wardetzky, Martin Rumpf, and Mirela Ben-Chen. 2015. Functional thin films on surfaces. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA 2015, Los Angeles, CA, USA, August 7-9, 2015*. Jernej Barbic and Zhigang Deng (Eds.). ACM, 137–146.
- Achi Brandt. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation* 31, 138 (1977), 333–390.
- Achi Brandt and Oren E Livne. 2011. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. SIAM.
- A Brandt, S McCoruick, and J Huge. 1985. Algebraic multigrid (amg) for sparse matrix equations. *Sparsity and its Applications* 257 (1985).
- Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw.* 35, 3, Article 22 (Oct. 2008), 14 pages. <https://doi.org/10.1145/1391989.1391995>
- Kazem Cheshmi, Danny M. Kaufman, Shoaib Kamil, and Maryam Mehri Dehnavi. 2020. NASQ: Numerically Accurate Sparsity-Oriented QP Solver. *ACM Trans. Graph.* 39, 4, Article 96 (July 2020), 17 pages. <https://doi.org/10.1145/3386569.3392486>
- Ming Chuang, Linjie Luo, Benedict J. Brown, Szymon Rusinkiewicz, and Michael M. Kazhdan. 2009. Estimating the Laplace-Beltrami Operator by Restricting 3D Functions. *Comput. Graph. Forum* 28, 5 (2009), 1475–1484.

- Jonathan D. Cohen, Dinesh Manocha, and Marc Olano. 2003. Successive Mappings: An Approach to Polygonal Mesh Simplification with Guaranteed Error Bounds. *Int. J. Comput. Geom. Appl.* 13, 1 (2003), 61.
- Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2017. The Heat Method for Distance Computation. *Commun. ACM* 60, 11 (Oct. 2017), 90–99. <https://doi.org/10.1145/3131280>
- Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. 2016. Subdivision exterior calculus for geometry processing. *ACM Trans. Graph.* 35, 4 (2016), 133:1–133:11.
- Christian Dick, Joachim Georgii, and Rüdiger Westermann. 2011. A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simul. Model. Pract. Theory* 19, 2 (2011), 801–816.
- Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (2016), 214:1–214:9.
- Ilja Friedel, Peter Schröder, and Andrei Khodakovskiy. 2004. Variational normal meshes. *ACM Trans. Graph.* 23, 4 (2004), 1061–1073.
- Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997*, G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal (Eds.). ACM, 209–216.
- Francisco José Gaspar, Jose L. Gracia, and Francisco Javier Lisbona. 2009. Fourier Analysis for Multigrid Methods on Triangular Grids. *SIAM J. Sci. Comput.* 31, 3 (2009), 2081–2102.
- Joachim Georgii and Rüdiger Westermann. 2006. A multigrid framework for real-time simulation of deformable bodies. *Comput. Graph.* 30, 3 (2006), 408–415.
- Seth Green, George Turkiyyah, and Duane W. Storti. 2002. Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *Seventh ACM Symposium on Solid Modeling and Applications, Max-Planck-Institut für Informatik, Saarbrücken, Germany, June 17-21, 2002*, Hans-Peter Seidel, Vadim Shapiro, Kunwoo Lee, and Nick Patrikalakis (Eds.). ACM, 265–272.
- Igor Guskov, Andrei Khodakovskiy, Peter Schröder, and Wim Sweldens. 2002. Hybrid meshes: multiresolution using regular and irregular refinement. In *Proceedings of the 18th Annual Symposium on Computational Geometry, Barcelona, Spain, June 5-7, 2002*, Ferran Hurtado, Vera Sacristán, Chandrajit Bajaj, and Subhash Suri (Eds.). ACM, 264–272.
- Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. 2000. Normal meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000*, Judith R. Brown and Kurt Akeley (Eds.). ACM, 95–102.
- Tom Haber, Tom Mertens, Philippe Bekaert, and Frank Van Reeth. 2005. A computational approach to simulate subsurface light diffusion in arbitrarily shaped objects. In *Proceedings of the Graphics Interface 2005 Conference, May 9-11, 2005, Victoria, British Columbia, Canada*, Kori Inkpen and Michiel van de Panne (Eds.). Canadian Human-Computer Communications Society, 79–86.
- Wolfgang Hackbusch. 2013. *Multi-grid methods and applications*. Vol. 4. Springer Science & Business Media.
- PW Hemker. 1990. On the order of prolongations and restrictions in multigrid procedures. *J. Comput. Appl. Math.* 32, 3 (1990), 423–429.
- Philipp Herholz and Marc Alexa. 2018. Factor Once: Reusing Cholesky Factorizations on Sub-Meshes. *ACM Transaction on Graphics (Proc. of Siggraph Asia)* 37, 6 (2018), 9. <https://doi.org/10.1145/3272127.3275107>
- Philipp Herholz and Olga Sorkine-Hornung. 2020. Sparse cholesky updates for interactive mesh parameterization. *ACM Trans. Graph.* 39, 6 (2020), 202:1–202:14.
- Hugues Hoppe. 1996. Progressive Meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*, John Fujii (Ed.). ACM, 99–108.
- Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.* 39, 4 (2020), 117.
- Alec Jacobson, Elif Tosun, Olga Sorkine, and Denis Zorin. 2010. Mixed Finite Elements for Variational Surface Modeling. *Comput. Graph. Forum* 29, 5 (2010), 1565–1574.
- Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant field-aligned meshes. *ACM Trans. Graph.* 34, 6 (2015), 189:1–189:15.
- In-Yong Jeon, Kwang-Jin Choi, Tae-Yong Kim, Bong-Ouk Choi, and Hyeon-Seok Ko. 2013. Constraining Multigrid for Cloth. *Comput. Graph. Forum* 32, 7 (2013), 31–39.
- Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective projection in a shell. *ACM Trans. Graph.* 39, 6 (2020), 247:1–247:18.
- Alexandr Katrutsa, Talgat Daulbaev, and Ivan V. Oseledets. 2020. Black-box learning of multigrid parameters. *J. Comput. Appl. Math.* 368 (2020).
- Misha Kazhdan and Hugues Hoppe. 2019. An Adaptive Multi-Grid Solver for Applications in Computer Graphics. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 138–150.
- Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. 2012. Can Mean-Curvature Flow be Modified to be Non-singular? *Comput. Graph. Forum* 31, 5 (2012), 1745–1754.
- Michael M. Kazhdan and Hugues Hoppe. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph.* 27, 3 (2008), 21.
- Andrei Khodakovskiy, Nathan Litke, and Peter Schröder. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.* 22, 3 (2003), 350–357.
- Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. 1998. A General Framework for Mesh Decimation. In *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, Wayne A. Davis, Kellogg S. Booth, and Alain Fournier (Eds.). Canadian Human-Computer Communications Society, 43–50.
- Dilip Krishnan, Raanan Fattal, and Richard Szeliski. 2013. Efficient preconditioning of laplacian matrices for computer graphics. *ACM Trans. Graph.* 32, 4 (2013), 142:1–142:15.
- Dilip Krishnan and Richard Szeliski. 2011. Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph.* 30, 6 (2011), 177.
- Junyu Lai, Yangang Chen, Yu Gu, Christopher Batty, and Justin W. L. Wan. 2020. Fast and Scalable Solvers for the Fluid Pressure Equations with Separating Solid Boundary Conditions. *Comput. Graph. Forum* 39, 2 (2020), 23–33. <https://doi.org/10.1111/cgf.13909>
- Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence C. Cowsar, and David P. Dobkin. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, Steve Cunningham, Walt Bransford, and Michael F. Cohen (Eds.). ACM, 95–104.
- Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.* 21, 3 (2002), 362–371.
- Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. 2020. Neural subdivision. *ACM Trans. Graph.* 39, 4 (2020), 124.
- Hsueh-Ti Derek Liu and Alec Jacobson. 2019. Cubic Stylization. *ACM Transactions on Graphics* (2019).
- Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. 2008. A Local/Global Approach to Mesh Parameterization. *Comput. Graph. Forum* 27, 5 (2008), 1495–1504.
- Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam Gingold. 2017. Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Transactions on Graphics (TOG)* 36, 6, Article 216 (Nov. 2017), 15 pages. <https://doi.org/10.1145/3130800.3130897>
- Josiah Manson and Scott Schaefer. 2011. Hierarchical Deformation of Locally Rigid Meshes. *Comput. Graph. Forum* 30, 8 (2011), 2387–2396. <https://doi.org/10.1111/j.1467-8659.2011.02074.x>
- Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. In *Proceedings of the 2010 Eurographics/ACM SIGGRAPH Symposium on Computer Animation, SCA 2010, Madrid, Spain, 2010*, Zoran Popovic and Miguel A. Otaduy (Eds.). Eurographics Association, 65–73.
- Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.* 30, 4 (2011), 37.
- Xinlai Ni, Michael Garland, and John C. Hart. 2004. Fair morse functions for extracting the topological structure of a surface mesh. *ACM Trans. Graph.* 23, 3 (2004), 613–622.
- Seungwoo Oh, Jun-yong Noh, and KwangYun Wohn. 2008. A physically faithful multigrid method for fast cloth simulation. *Comput. Animat. Virtual Worlds* 19, 3-4 (2008), 479–492.
- L. N. Olson and J. B. Schroder. 2018. PyAMG: Algebraic Multigrid Solvers in Python v4.0. <https://github.com/pyamg/pyamg> Release 4.0.
- Miguel A. Otaduy, Daniel Germann, Stephane Redon, and Markus H. Gross. 2007. Adaptive deformations with fast tight bounds. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2007, San Diego, California, USA, August 2-4, 2007*, Michael Gleicher and Daniel Thalmann (Eds.). Eurographics Association, 181–190.
- Nicolas Ray and Bruno Lévy. 2003. Hierarchical Least Squares Conformal Map. In *11th Pacific Conference on Computer Graphics and Applications, PG 2003, Canmore, Canada, October 8-10, 2003*. IEEE Computer Society, 263–270.
- John W Ruge and Klaus Stüben. 1987. Algebraic multigrid. In *Multigrid methods*. SIAM, 73–130.
- Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested cages. *ACM Trans. Graph.* 34, 6 (2015), 170:1–170:14.
- Ryan Schmidt and Karan Singh. 2010. Meshmiser: An Interface for Rapid Mesh Composition. In *ACM SIGGRAPH 2010 Talks* (Los Angeles, California) (*SIGGRAPH '10*). Association for Computing Machinery, New York, NY, USA, Article 6, 1 pages.
- Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: a sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph.* 33, 6 (2014), 205:1–205:12.
- Lin Shi, Yizhou Yu, Nathan Bell, and Wei-Wen Feng. 2006. A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph.* 25, 3 (2006), 1108–1117.
- Xiaohan Shi, Huijun Bao, and Kun Zhou. 2009. Out-of-core multigrid solver for streaming meshes. *ACM Trans. Graph.* 28, 5 (2009), 173. <https://doi.org/10.1145/1618452.1618519>
- Mélina Skouras, Bernhard Thomaszewski, Bernd Bickel, and Markus H. Gross. 2012. Computational Design of Rubber Balloons. *Comput. Graph. Forum* 31, 2 (2012),

- 835–844.
- Oded Stein, Alec Jacobson, Max Wardetzky, and Eitan Grinspun. 2020. A Smoothness Energy without Boundary Distortion for Curved Surfaces. *ACM Trans. Graph.* 39, 3 (2020), 18:1–18:17.
- Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.* 34, 6 (2015), 245:1–245:13.
- Philip Trettner and Leif Kobbelt. 2020. Fast and Robust QEF Minimization using Probabilistic Quadrics. *Comput. Graph. Forum* 39, 2 (2020), 325–334. <https://doi.org/10.1111/cgf.13933>
- Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. 2000. *Multigrid*. Elsevier.
- Petr Vanek, Jan Mandel, and Marian Brezina. 1996. Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. *Computing* 56, 3 (1996), 179–196.
- Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel Multigrid for Nonlinear Cloth Simulation. *Comput. Graph. Forum* 37, 7 (2018), 131–141.
- D. J. A. Welsh and M. B. Powell. 1967. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Comput. J.* 10, 1 (1967), 85–86.
- Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.
- Hui Zhao, Na Lei, Xuan Li, Peng Zeng, Ke Xu, and Xianfeng Gu. 2017. Robust Edge-Preserved Surface Mesh Polycube Deformation. In *25th Pacific Conference on Computer Graphics and Applications, PG 2017 - Short Papers, Taipei, Taiwan, October 16-19, 2017*, Jernej Barbic, Wen-Chieh Lin, and Olga Sorkine-Hornung (Eds.). Eurographics Association, 17–22.
- Yongning Zhu, Efthychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.* 29, 2 (2010), 16:1–16:18.

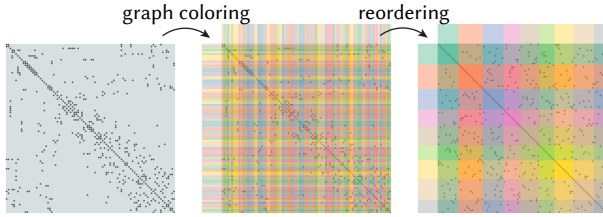


Fig. 26. Given a system matrix with the sparsity pattern showing on the left, we first use a greedy graph coloring approach detailed in Alg. 3 to “paint” the variables that are independent of each other with the same color (middle). Then we perform reordering to group the variables with the same color together (right) to parallelize our Gauss-Seidel relaxation.

Algorithm 3: Graph Coloring

1. sort nodes $\{n_i\}$ by degree
 2. $palette \leftarrow \{\}$
 3. **for each node** n_i **do**
 4. $N \leftarrow$ gather colors from painted neighbors of n_i
 5. $c \leftarrow$ find first entry in $palette$ not occurring in N
 6. **if** c **is not found then**
 7. $c \leftarrow$ new color
 8. append c to $palette$
 9. **else**
 10. move c to back of $palette$
 11. paint n_i with color c
-

A MULTI-COLOR GAUSS SEIDEL

Our multigrid method spends a lot of the runtime on the Gauss-Seidel relaxation. We further accelerate the Gauss-Seidel relaxation by exploiting graph coloring (see Fig. 26), a standard optimization. Specifically, we treat the non-zero off-diagonal entries of a given sparse matrix A as a graph. We color this graph so that each node has a different color from its neighbors using a simple modification of the method proposed by Welsh and Powell [1967], summarized in Alg. 3 and repeated here for completeness. We color each node in descending order of degree. When considering node i , we try each color from a list of k colors that have been previously used for nodes $(1, \dots, i-1)$. A color choice is valid if not matching any of the previously colored neighbors of node i . If valid, node i is colored and that color is moved to the back of the list. If no valid color is found in the list, a new color is used and added to the back of the list. This algorithm has $O(|V|\log|V| + |E|k)$ runtime and $O(|V| + |E|)$ memory complexity, respectively, where k is the number of output colors (for sparse matrices, $k \ll |V|$). Although suboptimal (finding the optimal coloring is NP-complete), it handily outperforms the method of [Fratarcangeli et al. 2016] in runtime, memory usage, and color parsimony. By moving selected colors dynamically to the back of the list, we achieve better color balance (see, e.g., Fig. 26) than considering the list in fixed order of insertion.

B DIRICHLET BOUNDARY CONDITIONS

Solving a linear system $Ax = b$ is equivalent to minimizing a quadratic energy

$$E(x) = \frac{1}{2}x^T Ax - x^T b \quad (7)$$

where one can derive the same linear system by setting $\partial E / \partial x = 0$. One way to handle Dirichlet boundary conditions $x(known) = c$ is to reformulate the quadratic energy using only the unknown variables. Here we use $known$ and $unknown$ to represent indices of knowns and unknowns. We further use $x_k = x(known)$ to denote known variables and $x_u = x(unknown)$ for unknown variables in x . For matrices, we follow the same notation. For example, we use $A_{uk} = A(unknown, known)$ to represent the corresponding sub-block in matrix A . We then rewrite the energy as (assuming A is symmetric)

$$E(x_u) = \frac{1}{2}x_u^T A_{uu}x_u + x_u^T A_{uk}x_k - x_u^T b_u + \text{constant}. \quad (8)$$

By setting the derivative to zero, we can derive a reduced linear system for only unknowns

$$\underbrace{A_{uu}}_{\text{LHS}} x_u = \underbrace{-A_{uk}x_k + b_u}_{\text{RHS}} \quad (9)$$

We can leverage the same trick to incorporate Dirichlet constraints in the multigrid system. We use x_c to denote the coarse variable such that $x = Px_c$ where the P is the Galerkin prolongation operator. We can then write the unknowns as

$$x_u = P_u x_c \quad (10)$$

where $P_u = P(unknown, :)$ (MATLAB notation) represents the rows of P that correspond to the *unknown* indices. Adding this to Eq. (11)

leads to

$$E(x_c) = \frac{1}{2} (P_{u:} x_c)^T A_{uu} (P_{u:} x_c) + (P_{u:} x_c)^T A_{uk} x_k \quad (11)$$

$$- (P_{u:} x_c)^T b_u + \text{constant}. \quad (12)$$

$$= \frac{1}{2} x_c^T P_{u:}^T A_{uu} P_{u:} x_c + x_c^T P_{u:}^T A_{uk} x_k \quad (13)$$

$$- x_c^T P_{u:}^T b_u + \text{constant}. \quad (14)$$

Similarly, setting the derivative with respect to x_c results in

$$\underbrace{P_{u:}^T A_{uu} P_{u:}}_{\text{reduced LHS}} x_c = \underbrace{P_{u:}^T (-A_{uk} x_k + b_u)}_{\text{reduced RHS}} \quad (15)$$

We can notice that, except at the second finest level where we need to extract the rows in prolongation that correspond to the *unknowns* $P_{u:}$, we can solve the linear system at coarser levels without worrying about the constraints.

Another special case may occur when there are too many *known* indices. If too many variables in x are given the reduced system $P_{u:}^T A_{uu} P_{u:}$ may have completely zero rows/columns. To handle this edge case, we further remove the columns on $P_{u:}$ where the maximum value is zero and the corresponding rows in the prolongation at the next level.

C SUCCESSIVE SELF-PARAMETERIZATION

Our pre-computation of multigrid hierarchy involves decimating the triangle mesh with successive self-parameterization and mapping vertices on the fine mesh to the coarse mesh to obtain their barycentric coordinates. We report the runtime of both pre-computation steps in Fig. 18.

Implementing successive self-parameterization only requires a small change to an existing edge collapse algorithm. Specifically, right after collapsing a single edge, the only modification is to use the method described in Sec. 4.2 and Sec. 4.3 to formulate the joint variable and then flatten both patches to a consistent UV domain. To determine whether the collapse and the flattening is valid, we refer to the Appendix B in [Liu et al. 2020] for more details. During the querying stage, for a given query point represented as barycentric coordinates, we simply go through the list of local joint UV parameterization we stored from the decimation stage and update the barycentric coordinates successively using the method described in Fig. 8. We pre-store the face indices involved in each edge collapse so that for each query point, we can easily check whether this point is involved in the collapse via checking the face indices.

D ABLATION STUDY

In addition to the prolongation operator, the hyperparameters of a multigrid method also play a role in the convergence behavior. In terms of stopping criteria, the accuracy ε depends on the problem and the size of the mesh. We usually set $10^{-5} \leq \varepsilon \leq 10^{-7}$ in order to get visually indistinguishable results compared to the ground truth. Using a reasonable initialization, such as the vertex positions in the previous iteration in mesh deformation, would further reduce the number of iterations to get the desired accuracy. For other hyperparameters, we conduct ablation studies on the choice of relaxation methods, pre-/post-relaxation iterations μ_{pre} , μ_{post} , and the

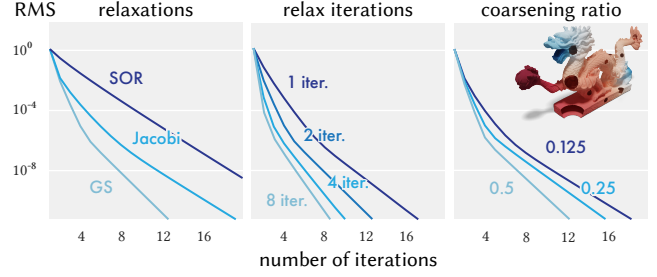


Fig. 27. We conduct an ablation study on the multigrid hyperparameters, including the relaxation method (left), the number of relaxation iterations (middle), and the coarsening ratio (right). ©model by Oliver Laric under CC BY-NC-SA.

coarsening ratio between consecutive levels (see Fig. 27). In terms of the relaxation methods, Gauss-Seidel is usually the go-to choice due to its effectiveness in smoothing out the high-frequency error. Practitioners may also prefer the (damped) Jacobi because it is faster and easier to parallelize, even though each iteration is less effective. In terms of the number of relaxation iterations, usually a couple of iterations (2 or 3) are sufficient to handle the high-frequency error. While we also notice that some multigrid methods (e.g., [Xian et al. 2019]) use lower-order prolongation with many more relaxation iterations to compensate for the inter-grid transfer error. In terms of coarsening ratio, using a less aggressive coarsening (e.g., 0.5) could reduce the error caused by the inter-grid transfer, but it often results in a bigger multigrid hierarchy and a longer runtime per cycle. On the other hand, using a more aggressive coarsening often leads to large inter-grid transfer error and slow convergence. Our default setup coarsens the geometry down to 0.25 of its previous resolution until we reach the coarsest mesh with no lesser than 500 vertices. In each V-cycle, we use the Gauss-Seidel relaxation with 2 pre- and post-relaxation iterations. Our experiments suggest that the optimal set of parameters that minimizes the wall-clock runtime depends on the geometry and the PDE of interest. But we use our default parameters for all our experiments in Sec. 6 for consistency.

E FAST DATA SMOOTHING

When we discretize the data smoothing energy Eq. (6), we often arrive the following linear system

$$\underbrace{(\alpha Q + (1 - \alpha)M)}_A x = (1 - \alpha)Mf \quad (16)$$

where Q is a matrix that depends on the choice of the smoothness energy, M is the mass matrix, f is the noisy function, and α is the smoothness parameter. In order to build the coarsened system matrix, a straightforward implementation would be doing $P^T A P$ directly, but we can actually split the computation via

$$P^T A P = \alpha(P^T Q P) + (1 - \alpha)(P^T M P). \quad (17)$$

Then we can pre-compute $P^T Q P$ and $P^T M P$ even before knowing the parameter α . As a results, during the online stage when a user adjusts α , we only require an efficient matrix addition to compute the system matrices for all the multigrid levels.