A Fast Geometric Multigrid Method for Curved Surfaces

Ruben Wiersma*
Delft University of Technology
The Netherlands
r.t.wiersma@tudelft.nl

Ahmad Nasikun*
Universitas Gadjah Mada
Delft University of Technology
Indonesia, The Netherlands

Elmar Eisemann
Delft University of Technology
The Netherlands

Klaus Hildebrandt
Delft University of Technology
The Netherlands

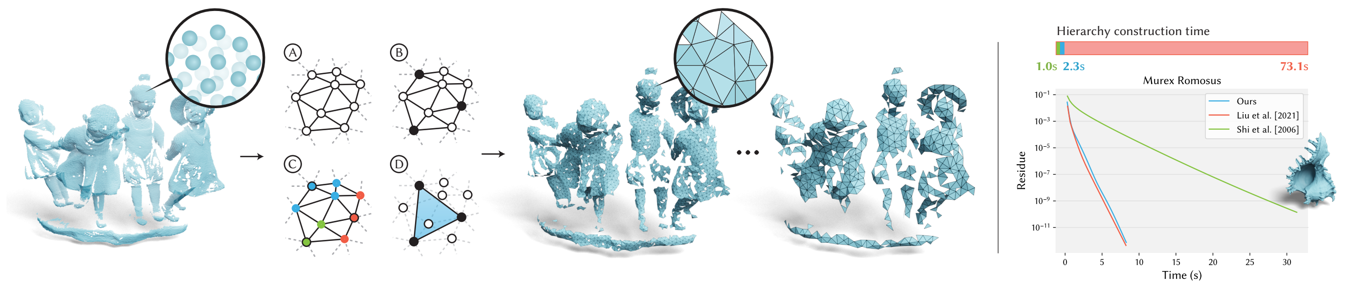

Figure 1: Illustration of our hierarchy construction for one level. Starting from a mesh or point cloud, A) we construct a neighbor graph on the surface. B) We then sample a spatially uniform set of nodes, C) compute the graph Voronoi diagram of the samples, and D) project unsampled points onto triangles formed by edges between Voronoi neighbors. This is repeated for every level. Right: Comparison of run time for solving a Laplace system on a triangle mesh. Our hierarchy construction is fast, while achieving similar solver performance to the state-of-the-art.

ABSTRACT

We introduce a geometric multigrid method for solving linear systems arising from variational problems on surfaces in geometry processing, Gravo MG. Our scheme uses point clouds as a reduced representation of the levels of the multigrid hierarchy to achieve a fast hierarchy construction and to extend the applicability of the method from triangle meshes to other surface representations like point clouds, nonmanifold meshes, and polygonal meshes. To build the prolongation operators, we associate each point of the hierarchy to a triangle constructed from points in the next coarser level. We obtain well-shaped candidate triangles by computing graph Voronoi diagrams centered around the coarse points and determining neighboring Voronoi cells. Our selection of triangles ensures that the connections of each point to points at adjacent coarser and finer levels are balanced in the tangential directions. As a result, we obtain sparse prolongation matrices with three entries per row and fast convergence of the solver. Code is available at https://graphics.tudelft.nl/gravo_mg.

*Both authors contributed equally to this research.

CCS CONCEPTS

• **Computing methodologies → Shape analysis**.

KEYWORDS

geometric multigrid, multigrid methods, Laplace matrix, geometry processing, Poisson problems

ACM Reference Format:
Ruben Wiersma, Ahmad Nasikun, Elmar Eisemann, and Klaus Hildebrandt. 2023. A Fast Geometric Multigrid Method for Curved Surfaces. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 6–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3588432.3591502

1 INTRODUCTION

Many geometry processing methods are based on variational problems and partial differential equations on curved surfaces. The discretization of these problems leads to sparse linear systems to be solved. One class of efficient solvers are Geometric Multigrid (GMG) methods, which use iterative solvers on a hierarchy of grids. They are more efficient than alternatives, such as sparse direct solvers, in many application scenarios [Liu et al. 2021]. While geometric multigrid solvers are well-studied for regular grids in Euclidean domains, the construction of effective geometric multigrid hierarchies remains challenging for irregular meshes on curved domains.

We distinguish two approaches to the design of GMG methods on curved surfaces. The first approach is to construct a hierarchy of meshes by mesh coarsening and then mapping between the meshes.

This approach obtains efficient prolongation operators that lead to fast convergence. A recent example is the intrinsic multigrid scheme by Liu et al. [2021]. The downside of this approach is a costly hierarchy construction. The second approach is to represent levels by graphs constructed by coarsening the edge graph of the input mesh [Shi et al. 2006]. This approach results in a fast construction but slower convergence.

We propose a new GMG method combining the strengths of both approaches. On the one hand, we use point clouds and neighbor graphs to represent levels, enabling a fast hierarchy construction. On the other hand, we use geometric operations to create local triangulations when constructing the prolongation operators for fast convergence. Our method solves linear systems as fast as the scheme of Liu et al. [2021], while reducing hierarchy-construction time by more than an order of magnitude. Moreover, our method is more generally applicable as it can be used not only for manifold triangular meshes but also for other discrete surface representations such as point clouds, non-manifold meshes, and polygonal meshes. Thus, we can solve systems set up with discrete differential operators for these representations, which were developed in recent years [Alexa and Wardetzky 2011; Liang and Zhao 2013; Sharp and Crane 2020]. Our hierarchy construction is more expensive compared to Shi et al. [2006]. Yet, the solving time is most often reduced more than the increase in hierarchy construction. This benefit increases for applications where multiple systems need to be solved.

The technical novelty of our method lies in a geometric multigrid method that is point-based, while still incorporating the geometry of the underlying surface. Our guiding idea is to construct intrinsic Delaunay triangulations on points sampled from the surface. Every other point can then be mapped from- and to the sampled points using barycentric coordinates in the intrinsic triangles. To get a fast and practical approach, we transfer this idea to a point-cloud setting. For every level in the hierarchy, we start by sampling points from the previous level using a fast uniform sampling strategy. Next, we compute graph Voronoi diagrams on the finer level using the sampled points as seeds and construct a neighborhood graph based on Voronoi cell adjacencies. Mimicking Delaunay triangulations, we construct triangles from the edges of the Voronoi adjacency graph. Each point of the finer level is projected to its closest triangle and barycentric coordinates are used for prolongation. This construction leads to sparse prolongation matrices with at most three entries per row and hence to fast prolongations and restrictions. The use of graph Voronoi cells ensures that the prolongation matrix and its transpose (the restriction matrix) contain entries corresponding to neighbors that are well-distributed over the tangential directions. We name our hierarchy construction Gravo MG, for graph Voronoi multigrid.

We evaluate Gravo MG in ablations and comparisons to [Liu et al. 2021], [Shi et al. 2006], and algebraic multigrid methods. Furthermore, we demonstrate the benefits of our scheme over sparse direct solvers in application scenarios.

2 RELATED WORK

Geometric multigrid. Multigrid methods [Bramble 1993] are among the most efficient iterative methods for solving linear systems. We call them *geometric* multigrid methods if the hierarchy construction is exclusively on the domain and no information is used about the system to be solved. GMG methods on regular grids are well studied [Hackbusch 1985] and used in graphics, *e.g.*, for fluid simulation [Dick et al. 2016; McAdams et al. 2010], image processing [Krishnan and Szeliski 2011; Pérez et al. 2003], and surface reconstruction [Kazhdan and Hoppe 2019; Kazhdan et al. 2006]. GMG methods for irregular grids on Euclidean domains are used for the simulation of cloth (2D) [Jeon et al. 2013; Wang et al. 2018] and elastic objects (3D) [Georgii and Westermann 2006; Otaduy et al. 2007].

In this work, we consider GMG methods for curved surfaces. Since the domain is no longer a Euclidean space but a curved manifold, methods from the Euclidean setting do not transfer directly and new methods are needed. Existing GMG methods on surfaces focus on triangle mesh representations of discrete surfaces. If the mesh is already equipped with a hierarchy, for example from a subdivision method, it can be used directly for a multigrid method [Green et al. 2002]. However, usually, only a fine-scale mesh is given and a hierarchy must be built. Based on earlier work on multiresolution representations of triangle meshes, [Hoppe 1996; Kobbelt et al. 1998], edge collapses are used to create multigrid hierarchies in [Aksoylu et al. 2005; Ni et al. 2004; Ray and Lévy 2003]. The prolongation operators are defined by weighted averaging with the one-ring neighbors. There are two approaches to guaranteeing that each vertex in a finer level has at least one neighbour in the coarser level: either the coarsening process is restricted to only collapsing edges, so that a maximal independent set of vertices (MIS) is removed [Aksoylu et al. 2005; Ni et al. 2004], or the edge collapses are restricted so that a MIS is preserved [Aksoylu et al. 2005]. Liu et al. [2021] introduce an intrinsic multigrid scheme that uses edge coarsening to create the meshes for the different levels and maintains bijective mappings between the meshes on consecutive levels. The map between two meshes is used to define the prolongations. The map assigns to each vertex of the finer mesh a point in a triangle of the coarser mesh and linear interpolation in the triangle is used for prolongation. The resulting prolongation matrix has at most three entries per row. An alternative to mesh coarsening is to use graph coarsening for hierarchy construction [Shi et al. 2006, 2009]. A multigrid scheme for the computation of Laplace–Beltrami eigenpairs on surfaces is introduced in [Nasikun and Hildebrandt 2022]. The hierarchy used for the eigenproblem, however, is much coarser than the hierarchies used for solving linear systems: only two or three levels are used. A multigrid solver for the computation of harmonic foliations on surfaces is introduced in [Wang et al. 2022].

Algebraic multigrid. Algebraic multigrid (AMG) methods [Brandt 1986; Stüben 2001] are an alternative to GMG. They use the matrix of the linear system to be solved to build the hierarchy instead of using the domain. This has the advantage that AMG can be used for problems coming from arbitrary domains. Nevertheless, AMG methods need to rebuild the hierarchy when the system matrix changes, whereas GMG methods only need to rebuild the hierarchy when the domain changes. An efficient multigrid preconditioner specifically for Laplace systems on images and meshes was introduced in [Krishnan et al. 2013]. Although fast, it has the disadvantage of requiring the Laplace matrices to have only non-positive off-diagonal

ALGORITHM 1: Multigrid solver

```
Input: Matrix $A \in \mathbb{R}^{n \times n}$, initial vector $x \in \mathbb{R}^n$, right-hand side
       $b \in \mathbb{R}^n$, error tolerance $\varepsilon$, number of levels $\lambda$, numbers of
       pre/post-relaxations steps $\nu_{pre}$, $\nu_{post}$
Output: Solution $x \in \mathbb{R}^n$ to the linear system $Ax = b$
Function Multigrid($A, x, b, \varepsilon, \lambda, \nu_{pre}, \nu_{post}$):
    Build prolongation matrices $P_1, P_2, P_3 \ldots, P_\lambda$
    $A_1 \leftarrow A$
    for $l \leftarrow 2$ to $\lambda$ do
        $A_l \leftarrow (P_{l-1})^\top A_{l-1} P_{l-1}$        // Build level $l$ matrix
    end
    repeat
        $x \leftarrow$ MGI($x, b, 1$)
    until $\|Ax - b\| \leq \varepsilon \|b\|$        //Convergence test
    return $x$
End Function
```

ALGORITHM 2: Multigrid Iteration (V-cycle)

```
Input: Current iterate $x \in \mathbb{R}^{n_l}$, right-hand side $b \in \mathbb{R}^{n_l}$, level $l$
Output: New iterate $x \in \mathbb{R}^{n_l}$
Function MGI($x, b, l$):
    if $l < \lambda$ then
        $y \leftarrow$ Relax($A_l, x, b, \nu_{pre}$)        //Pre-relaxation
        $u \leftarrow$ MGI($0, P_l^\top (b - A_l y), l + 1$)        //Recursive call
        $x \leftarrow$ Relax($A_l, x + P_l u, b, \nu_{post}$)        //Post-relaxation
    else
        Solve $A_\lambda x = b$ using a direct solver
    end
    return $x$
End Function
```

entries, which is often not satisfied by mesh Laplacians, such as the cotangent-Laplacian [Pinkall and Polthier 1993].

Direct solvers. Sparse direct solvers [Davis et al. 2016] are reliable, accurate, and commonly used for Laplace systems in geometry processing. Once a factorization of a matrix is computed, these solvers can solve multiple systems with the same matrix but different right-hand sides. In special cases, such as low-rank changes of the matrix, the factorization can be updated efficiently [Chen et al. 2008; Herholz and Alexa 2018; Herholz and Sorkine-Hornung 2020]. However, substantial changes require a new factorization. A disadvantage of these solvers is that they do not scale well neither in terms of memory requirements nor computation time. In Section 5, we compare the performance of our method to direct solvers in different scenarios.

3 BACKGROUND: MULTIGRID SOLVER

Multigrid solvers use a hierarchy of grids to solve systems of equations. Iterative solvers converge at different speeds for different scales, depending on the resolution of the grid on which they operate. Thus, by performing iterations on different grids, a multigrid scheme extends the range in which the solver converges particularly fast. Here, we describe a multigrid solver, which will later be used to evaluate our proposed hierarchy and prolongation operators.

We consider the multigrid solver in Algorithm 1. To solve an n-dimensional linear system $Ax = b$ for x, it operates on a multigrid hierarchy with λ levels, where level 1 is the finest and level λ is the coarsest level. A function on the l^{th} grid is represented by a vector in $\mathbb{R}^{n_l}$, where the grid has n_l degrees of freedom. The mappings between the grids are realized by prolongation and restriction matrices. The prolongation matrices $P_l \in \mathbb{R}^{n_l \times n_{l+1}}$ map from level $l + 1$ to level l. We use the transposed matrices of the prolongation matrices $P_l^\top$ as restriction matrices. The advantage is that a symmetric matrix A implies that the linear systems in the coarse grid correction, which involve the restricted matrices $P_l^\top A_l P_l$, are also symmetric.

The multigrid solver first builds the prolongation matrices. We keep this step abstract at this point but discuss it in detail in the following section. In the next step, lines 3-6, the restricted matrices for all levels are constructed. After the precomputation, multigrid iterations are executed until convergence of the solution. The multigrid iterations traverse the hierarchy from fine to coarse and back. This process is called a V-cycle and is simple but effective. Alternatively, instead of directly going up to the coarsest grid, one could first go back to finer grids. Such strategies can help to counteract error accumulation when several levels are traversed and thereby reduce the required number of multigrid iterations. On the other hand, the V-cycle is fast. The multigrid iterations, Algorithm 2, apply relaxation steps before and after the coarse grid correction. We use Gauss–Seidel iterations for this. Alternatives are schemes such as Jacobi iterations or conjugate gradient iterations. The number of Gauss–Seidel iterations applied in the pre and post relaxations is specified by the parameters ν_{pre} and ν_{post}. For V-cycles, one sets $\nu_{pre} = \nu_{post}$.

The norm used for the convergence test in line 9 of Algorithm 2 depends on the context. A common choice is the standard norm of $\mathbb{R}^n$. For the Poisson and smoothing problems, we use a mass-weighted 2-norm [Wardetzky et al. 2007].

4 HIERARCHY CONSTRUCTION

Our goal is to design a hierarchy construction that is faster than the intrinsic multigrid method by Liu et al. [2021]. It should be compatible with point clouds and general surface representations, while maintaining fast convergence during solving. Before giving an overview of our method, we revisit the idea that guided our design.

In the scheme of Liu et al. [2021], each level is represented by a mesh and mappings to adjacent levels. An intrinsic multigrid approach can alternatively represent levels by multiple *intrinsic* triangulations of the same surface. For example, each level can be a point sampling with corresponding intrinsic Delaunay triangulation. This idea, however, does not reflect a fast and more general construction. To achieve this, we transfer the idea into a point-cloud setting.

4.1 Overview

Our approach takes as input a set of point locations V sampled from a surface and a set of edges E between these points denoting local neighborhoods. For a mesh $\{V, E, F\}$, we use the vertices V and edges E. For a point cloud, edges could be taken from a radius graph or the 1-ring in a local Delaunay triangulation [Sharp and Crane 2020].

The algorithm outputs a sequence of sparse prolongation matrices P_l, mapping signals from n_{l+1} points to n_l points, where $n_{l+1} < n_l$. Relating n_{l+1} and n_l, we refer to points in level n_{l+1} as *coarse points* and points in n_l as *fine points*. Level 1 are the input points.

Algorithm overview. The hierarchy is constructed one level at a time. For each level, the algorithm takes the input graph from level l, $\{V_l, E_l\}$, and outputs the graph in level $l+1$, $\{V_{l+1}, E_{l+1}\}$. The algorithm also produces the prolongation matrix P_l. This is repeated until level λ is reached. Here, we describe one such step. Figure 1 provides a corresponding visual overview.

First, the point cloud is subsampled using a fast greedy algorithm that aims to enforce a minimum edge length in the next graph (subsection 4.2). Next, we create a graph Voronoi diagram, where the coarse points (sampled points) act as Voronoi centers and the fine points are the loci of the Voronoi cells. We then seek a mapping from the coarse points to the fine points. Mimicking the construction of a Delaunay triangulation as the dual of a Voronoi diagram, we construct a neighbor relation of the graph Voronoi cells (subsection 4.3) and compute all the triangles formed by the edges between Voronoi cell centers (subsection 4.4). Finally, the fine points are projected onto these triangles to find the triangle closest to the fine point. The neighbor relations of the graph Voronoi diagram are then used as edges for the next level E_{l+1}.

4.2 Sampling

Each new level contains fewer points than the previous and the samplings should be spatially uniform [Liu et al. 2021; Shi et al. 2006]. In other words, we seek a dense sample set S, in which no pair of points is closer than a prescribed distance r. To find such a set, we use an algorithm based on the maximum independent set: we sweep once over V, keeping track if points are eligible for addition to S. Initially, all points are eligible. If a point p is eligible, we add it to S and mark the points within geodesic distance r of p as ineligible. The radius r is based on the fraction $\phi < 1$ of points we wish to keep and the average edge length $\hat{e}$

$$r = \phi^{-\frac{1}{3}} \hat{e}. \quad (1)$$

In our experiments, we set $\phi = 1/8$ and stop coarsening at 1000 points, yielding roughly $\log_8(N/1000)$ levels. We also limit the search for nearby points to the 2-ring, as this strikes a good balance between construction speed and sampling quality. In section 5.3 we experimentally validate that we indeed approximately reach the desired fraction of samples and Figure 10 demonstrates the uniformity of the resulting sampling and corresponding triangles.

4.3 Neighbor graph

We use graph Voronoi diagrams [Erwig 2000] to define neighborhoods for the sampled points. Since we build the levels successively from fine to coarse, the neighbor graph $\{V_l, E_l\}$ is already built on level l when level $l+1$ is visited. The points V_{l+1} are the seeds of the graph Voronoi diagram in $\{V_l, E_l\}$. For each seed $i \in V_{l+1}$, the Voronoi cell consists of the points in V_l that are closer in graph distance to i than to all other points of V_{l+1}. The graph Voronoi diagram can be efficiently computed by a multisource Dijkstra algorithm. For points $i, j \in V_{l+1}$, we add an edge $\{i, j\}$ to E_{l+1}, if there is an edge in E_l that connects a point of the Voronoi cell of i with a point of the Voronoi cell of j.

4.4 Prolongation

Prolongation operators map functions on level $l+1$ to functions on level l, by matrices $P_l \in \mathbb{R}^{n_l \times n_{l+1}}$. The restrictions, mapping from level l to level $l+1$, are given as the transpose matrices $P_l^\top$. Important for the design of the prolongation matrices is their sparsity. The sparser the prolongation matrices, the sparser the restricted matrices $P_l^\top A P_l$, and the faster the mappings between the levels. To construct the prolongation, we use linear interpolation in triangles. Hereby, we get very sparse prolongations matrices. Other interpolation methods, such as radial basis functions or spline interpolations, would result in much denser prolongation matrices.

First, a set of candidate triangles on the coarse points is constructed. Every coarse point has a Voronoi cell on the finer level. Two coarse points i, j are connected by an edge $\{i, j\} \in E_{l+1}$ in the coarser level if their corresponding Voronoi cells are neighbors. We consider all triangles that can be constructed from these edges: all triplets $\{i, j, k\}$ such that $\{i, j\}, \{j, k\}, \{k, i\} \in E_{l+1}$.

The motivation to use these edges is the duality between (intrinsic) Voronoi diagrams and Delaunay triangulations (two points in a Delaunay triangulation are connected by an edge iff their Voronoi cells are adjacent). Since graph Voronoi cells are not continuous, but approximations computed from a sampling, the triangles we obtain are not necessarily Delaunay triangles, and they do not necessarily form a manifold. However, as illustrated in Figure 10 and 11, we mostly get well-shaped triangles and a good coverage of the surface, even for point clouds.

To get the prolongation weights for a fine point p, we search for the closest candidate triangle. For efficiency, we restrict this search to the triangles that include the coarse point closest to p. The weights are the barycentric coordinates of the closest point to p in the selected triangle (this can be on an edge or a vertex). The barycentric coordinates of the projected point are then entered into the prolongation matrix.

Edge-cases. In some cases, a suitable triangle cannot be found within the Voronoi neighborhood of the closest point. This might happen, for example, if all points in the neighborhood are (nearly) co-linear, or if the fine point falls outside of the triangles formed in the neighborhood. In these cases, we resort to finding the closest three points within the neighborhood and use inverse-distance weights. This is preferable over projecting to a single vertex, as it helps the spread of information during prolongation. In practice, this only happens in a fraction of cases (roughly 0.25%).

Reducing single-entry rows. The resulting prolongation matrices are very sparse with maximally three non-zero entries per row. Since the coarse points are created by subsampling the fine points, the fine points that are sampled transfer their function value directly to the corresponding coarse point during prolongation. Therefore, there is only one entry in the corresponding rows. We obtain prolongation matrices with fewer single-entry rows by moving each sampled point to the mean of the points that form its graph Voronoi cell before we compute the closest point projections. In our experiments, we obtained a slight improvement of solving times with this strategy over not moving the coarse points.

5 EXPERIMENTS

We evaluate Gravo MG and compare to state-of-the-art GMG methods and AMG approaches. For reference, we provide results for direct solvers. We also provide insights into design choices via ablation studies.

5.1 Implementation

Our multigrid-solver implementation builds on Liu et al. [2021]'s code, where the prolongation matrix definition is exchanged. In every experiment, we set the number of pre/post-relaxation steps $v_{pre} = v_{post} = 2$. The hierarchy construction uses custom routines built around Eigen [Guennebaud et al. 2010] and only requires a matrix of points and an array of edges. The code for our solver is available as a C++ library and Python package, along with scripts to replicate the main tables and figures in this paper: https://graphics.tudelft.nl/gravo_mg.

We use an Intel®Core™i9-9900 CPU (3.10GHz, 32GB memory). The code does not employ multithreading but could be parallelized. None of the methods in our comparisons are parallelized, except Pardiso. A discussion on the potential for parallelization of the solver we used can be found in Section 7 of [Liu et al. 2021].

5.2 Problems

In our ablations and comparisons, we test our approach on two standard problems that can be written as linear systems: data smoothing and Poisson problems. For meshes, both problems involve the cotan Laplace matrix S and the lumped mass matrix M, see [Wardetzky et al. 2007]. In the case of point clouds and non-manifold meshes, we use the robust Laplacian by Sharp and Crane [2020]. Data can be smoothed by solving

$$(M + \alpha S)x = My, \tag{2}$$

where y is the noisy input function and α a parameter that determines how much the data is smoothed. The Poisson problem is

$$(S + \eta M)x = My, \tag{3}$$

where y is a random vector. The term ηM is added to obtain a positive-definite system matrix. The parameter η is chosen to be very small, for example $\eta = 1 \times 10^{-6}$. Our solver terminates when tolerance ε is reached (line 9 of algorithm 1) or after a maximum number of iterations.

5.3 Ablation Studies

We would like to understand the effects of our design choices on the hierarchy construction and subsequent solving steps. We structure these experiments along three themes: sampling, prolongation selection, and weighting. In each ablation, we compare variants of our approach on a fixed set of meshes and point clouds and run a data smoothing problem as detailed above. We smooth a random function with $\alpha = 1 \times 10^{-3}$ and tolerance 1×10^{-4}. Each variant is then evaluated in terms of time to construct the hierarchy, the number of iterations required to reach the target tolerance, and the total time.

Sampling. We seek a sampling method that balances run-time during hierarchy construction and sampling quality. To validate that our approach effectively balances these demands, we compared our approach to random sampling, Poisson-disk-sampling (PDS), geodesic farthest point sampling (FPS), and maximal independent set (MIS) selection. For every method, we set the target ratio between levels to 1/8th. In Table 1, we observe that our approach is faster than the others when solving. When considering the full hierarchy construction, our approach is faster than every other sampling approach, because we perform part of the graph Voronoi diagram construction during sampling and have fewer points per level to consider than MIS.

We also compare decay rate between the maximal δ-independent set, used in [Shi et al. 2006], and our sampling. In Figure 2, we see that it is possible to decay much faster with our approach than MδIS, because it must always be a superset of the MIS. This is an advantage of our approach; we can perform faster and fewer iterations, while having a fast sampling time.

Prolongation selection. Our approach uses triangles of coarse points for the prolongation operator. This results in sparse prolongation matrices that spread information in each tangential direction. In this ablation, we seek to support this choice. In Table 2, we compare our approach to the following variants that do not explicitly work with triangles: simply prolonging to the closest two, three, or four points from the graph Voronoi neighbors and picking three random points. We also test a variant that considers triangles without restricting to Voronoi edges, 'closest tri'. This results in less

Table 1: Data smoothing timings with variations of the sampling step (*Smp*). We compare our approach to *random* sampling, Poisson Disk Sampling (*PDS*), geodesic Farthest Point Sampling (*FPS*), and Maximum-Independent Set (*MIS*). All timings are in seconds, unless otherwise specified.

Model	#V	Ours		Random		PDS		FPS		MIS	
		Smp	Solve	Smp	Solve	Smp	Solve	Smp	Solve	Smp	Solve
Brd Man	691k	0.08	**0.62**	0.01	1.01	0.47	0.63	1m	0.79	0.02	0.87
Rd Circ. Box	701k	0.08	**0.81**	0.01	1.30	0.39	1.10	1m	1.38	0.02	1.01
Nefertiti	1m	0.12	**1.10**	0.01	2.05	1.02	1.29	2m	1.65	0.04	1.15
Murex	1.8m	0.42	**3.02**	0.03	4.84	1.47	3.41	6m	4.83	0.11	3.68
XYZ Dragon	3.6m	0.35	5.72	0.06	12.95	2.48	**4.92**	27m	-	0.11	7.38

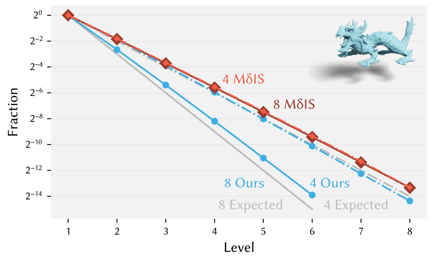

Figure 2: Comparison of the decay rate between MδIS used by Shi et al. [2006] and our approach on XYZ dragon. The y-axis is in $\log_2$ scale.

Table 2: Timings for hierarchy construction and solving on data smoothing with variations of the entries in the prolongation operator. *Ours* only considers triangles formed by Voronoi edges. The other variants either pick n *closest* points, pick n *random* points or pick the three Voronoi neighbors that form the *closest triangle*. All timings are in seconds.

Model	#Vert	3 points			2 points			3 points									4 points		
		Ours			Closest			Random			Closest vert			Closest tri			Closest		
		Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve
Beard Man	691k	0.64	4	**0.62**	0.46	9	0.91	0.42	26	2.91	0.50	5	0.72	0.87	5	0.77	0.51	5	0.83
Red Circular Box	701k	0.67	6	**0.79**	0.49	10	1.03	0.49	31	3.45	0.52	7	0.95	1.01	8	1.06	0.58	7	1.01
Nefertiti	1m	0.93	4	**1.05**	0.67	9	1.68	0.60	29	5.75	0.69	6	1.36	1.35	5	1.29	0.73	5	1.37
Murex Romosus	1.8m	2.64	5	**3.11**	1.80	11	4.15	1.70	40	15.07	2.07	6	3.60	3.82	6	3.41	1.97	6	3.91
XYZ Dragon	3.6m	3.46	9	**5.72**	2.41	15	7.55	2.28	45	25.10	2.61	14	8.35	4.45	16	8.76	2.68	11	7.32

consistent triangulations, as shown in Figure 3. For each of the non-triangle selection approaches, we use inverse distance weights. We observe that our approach solves faster than all other variants, while increasing the hierarchy construction time only a bit.

Weighting. We project the fine points onto the triangles formed by coarse points and use barycentric weights as predictors for the value of the fine point. Previous works suggest that the choice of weighting schemes has little effect on convergence times [Aksoylu et al. 2005; Shi et al. 2006], while Liu et al. [2021] argue that the weighting scheme is crucial for some shapes. In Table 3, we compare our approach with uniform weights and inverse distance weights alongside a variant where we do not shift the coarse points to the barycenters. We observe that our approach works best with barycentric coordinates (*Ours*). Inverse-distance weights are not far behind. Shifting coarse points has benefits for some, but not all shapes. This is not the core contribution of our work and could be left out in some cases. A benefit of not shifting coarse points is that each iteration is faster because the prolongation matrix contains more single-entry rows.

5.4 Comparisons

We compare our approach on a wide range of meshes and point clouds for a Poisson problem with $\eta = 1 \times 10^{-6}$ and target tolerance of 1×10^{-4}. The input function y is a random vector sampled from $\mathcal{N}(0, 1)$.

The shapes were selected to have at least 100k vertices and exhibit a wide variety: uniform meshes (*e.g.*, Nefertiti), non-uniform meshes (*e.g.*, Alfred Jacquemart, Indonesian statue), broken and non-manifold meshes. The meshes also exhibit detailed features (*e.g.*, XYZ dragon) and complex curvature (*e.g.*, Murex Romosus). All the shapes are shown in Figure 12. We make no use of additional pre-processing steps, such as remeshing or fixing non-manifold edges: every mesh is used as-is in the highest resolution available from the respective sources. For the point clouds, we opted for high-resolution scanned data. The point clouds come from the Tanks and Temples benchmark dataset [Knapitsch et al. 2017] and from range scans in the AIM@Shape repository [Falcidieno 2007].

Gravo MG is compared to the GMG solvers by Liu et al. [2021] and Shi et al. [2006], and the AMG methods Ruge–Stuben and Smoothed Aggregation. For reference, we list the timings of direct solvers. For Liu et al., we use their provided implementation. We reimplemented Shi et al. based on their paper. The latter mentions multiple weighting schemes, including uniform weights and inverse distance weights. We tested both and report the best-performing approach: inverse distance weights. For the AMG approaches, we use the implementation provided in PyAMG [Bell et al. 2022] with default settings provided by the package. We set the maximum number of iterations for all iterative solvers to 100, since more iterations would not change the overall picture regarding which solver is faster. The direct solver references are the Cholesky LLT factorization provided in Eigen and Intel®MKL's PARDISO solver, which is highly optimized and parallelized [Intel Corporation 2023].

Our approach yields faster solving times for the majority of input meshes (Table 4). More results for manifold meshes are listed in the supplement in Table 1. On average, our construction is 36x faster than Liu et al. and only 1.8x slower than Shi et al.'s method. With regards to solving time, Liu et al. takes 3% more time on average for the Poisson problem and 7% for data smoothing and Shi et al.

Table 3: Timings for solving on data smoothing with variations of the weighting scheme. We compare barycentric coordinates (*Ours*) to *uniform* weights, *inverse distance* weights, and barycentric coordinates without chaging the positions of the coarse points before projection (*No shift*). All timings are in seconds.

Model	#Vert	Ours		Uniform		Inv. Dist.		No shift	
		#It	Solve	#It	Solve	#It	Solve	#It	Solve
Beard Man	691k	4	0.63	12	1.30	5	0.71	4	**0.56**
Red Circular Box	701k	6	**0.80**	23	2.36	6	**0.80**	10	1.12
Nefertiti	1m	4	1.04	14	2.68	5	1.21	4	**0.97**
Murex Romosus	1.8m	5	3.09	16	6.54	6	3.37	5	**2.71**
XYZ Dragon	3.6m	9	**5.71**	23	12.23	9	5.74	18	9.28

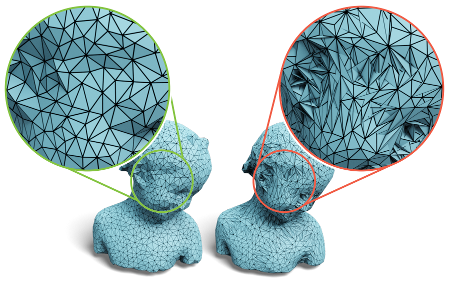
Figure 3: Using dual Voronoi triangles results in a more consistent set of candidate triangles than using all triangles in the coarse point's 1-ring.

Table 4: Comparison of our hierarchy construction and solver for a Poisson problem with $\eta = 1 \times 10^{-6}$ mass matrix coefficient and tolerance of 1×10^{-4}. Missing entries are not available for the given method. The maximum number of iterations for iterative solvers is set to 100.

Model	#Vert	Gravo MG (Ours)			Liu et al. [2021]			Shi et al. [2006]			AMG-RS			AMG-SA			Eigen		Pardiso	
		Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve	Hier	#It	Solve	Fact.	Subst.	Fact.	Subst.
MANIFOLD TRIANGULAR MESHES																				
Aim Dragon	152k	0.14	7	0.19	5.14	8	0.27	0.06	27	0.62	0.15	26	0.46	0.31	29	0.48	0.81	0.02	0.58	0.04
Blade Smooth	195k	0.15	4	0.17	5.87	3	0.17	0.09	18	0.60	0.18	100	2.42	0.40	42	0.94	0.76	0.02	0.69	0.04
Moses	258k	0.42	12	0.55	8.72	5	0.35	0.30	100	4.30	0.27	100	3.64	0.55	100	3.33	0.90	0.02	1.04	0.05
Julius Caesar	387k	0.30	11	0.58	12.10	17	1.09	0.16	28	1.54	0.39	100	4.89	0.77	70	2.79	5.07	0.06	1.29	0.09
Bimba	502k	0.43	7	0.65	15.58	6	0.74	0.24	48	4.16	0.51	100	6.97	1.15	69	4.45	3.54	0.05	2.13	0.10
Antique Head	651k	0.53	4	0.51	20.07	4	0.60	0.28	14	1.22	0.64	100	8.48	1.37	67	4.33	15.02	0.11	2.43	0.17
Beard Man	691k	0.59	4	0.56	22.15	3	0.58	0.26	14	1.43	0.52	100	7.07	1.46	14	0.96	24.57	0.14	2.72	0.19
Red Circular Box	701k	0.64	6	0.74	22.97	6	0.97	0.34	66	6.67	0.72	100	8.98	1.51	66	5.01	17.76	0.11	2.84	0.17
Dancing Children	724k	0.69	9	1.10	23.21	8	1.25	0.41	39	4.54	0.68	100	9.32	1.23	100	8.70	6.18	0.09	2.78	0.17
Ramses	826k	0.79	7	1.21	28.90	5	1.18	0.47	40	6.03	0.91	100	11.82	2.08	49	5.40	6.24	0.09	3.60	0.18
Nefertiti	1m	0.89	4	0.94	34.22	4	1.18	0.56	65	11.69	1.09	100	14.74	2.58	46	6.14	9.15	0.10	4.54	0.22
Isidore Horse	1.1m	1.12	11	1.90	35.60	5	1.27	0.50	70	11.03	1.11	100	14.60	2.50	88	10.72	24.01	0.17	4.56	0.28
Ram	1.3m	2.40	3	1.95	56.18	-	-	1.14	49	13.39	2.45	100	25.95	4.72	100	21.71	17.07	0.15	6.99	0.33
Murex Romosus	1.8m	2.32	6	2.85	73.05	5	3.32	0.99	63	20.79	2.38	100	29.83	5.08	58	15.28	40.06	0.26	9.13	0.44
XYZ Dragon	3.6m	3.24	9	5.32	121.97	7	5.32	1.57	55	28.95	3.16	100	43.75	9.14	75	30.54	77.62	0.69	15.88	0.94
NON-MANIFOLD TRIANGULAR MESHES																				
Lakoon	188k	0.16	8	0.29	-	-	-	0.09	41	1.33	0.21	100	2.77	0.47	48	1.12	0.42	0.01	0.71	0.04
Indonesian Statue	294k	0.26	11	0.58	-	-	-	0.16	64	3.15	0.30	100	3.92	0.63	100	3.55	0.92	0.03	1.18	0.06
Beethoven	383k	0.45	4	0.49	-	-	-	0.23	60	4.00	0.51	20	1.29	0.96	100	5.30	2.39	0.04	1.65	0.09
Bayon Lion	749k	1.42	6	1.55	-	-	-	0.70	26	4.31	1.30	100	15.36	2.49	43	5.57	5.99	0.08	3.79	0.18
Helmet Moustache	941k	2.04	9	2.89	-	-	-	0.74	57	11.15	2.03	100	19.66	3.31	38	6.14	24.66	0.14	5.56	0.25
Zeus	1.3m	2.47	11	3.86	-	-	-	1.17	58	15.91	2.34	100	27.21	4.11	100	22.68	30.40	0.20	7.19	0.35
Alfred Jacquemart	1.4m	3.33	5	3.79	-	-	-	1.67	43	16.21	3.03	100	30.33	5.44	51	12.78	8.88	0.14	8.07	0.35
POINT CLOUDS																				
Oil Pump	103k	0.07	9	0.12	-	-	-	0.04	15	0.22	0.10	100	1.27	0.19	55	0.56	0.17	0.01	0.31	0.02
Caesar Merged	388k	0.29	6	0.40	-	-	-	0.17	18	1.14	0.41	100	5.52	0.83	87	3.98	4.90	0.06	1.50	0.10
Truck	1.2m	0.99	17	3.20	-	-	-	0.65	26	5.53	1.27	100	18.87	3.67	72	10.77	5.63	0.14	5.09	0.29
Ignatius	1.4m	1.26	7	1.87	-	-	-	0.78	33	8.41	1.59	100	21.61	4.42	100	17.78	8.92	0.18	6.13	0.36

takes 274% more time for the Poisson problem and 81% for data smoothing. Note that we require less time for one iteration than Liu et al., because we use a higher decay rate (1/8 vs. 1/4). This is balanced out in most cases by a higher iteration count and the overall solving times are similar when we use a decay rate of 1/4.

GMG methods are most beneficial in settings where a user would iterate on the system matrix, but the benefit of using Gravo MG is already noticeable starting with the first solve. For all meshes larger than 100k vertices, our approach is faster than Liu et al. for both the Poisson problem and data smoothing. The same holds for Shi et al. for the Poisson problem. For data smoothing, we are faster for one solve in 83% of cases and for three solves in 93% of cases. Compared to the Pardiso solver, we are faster for one solve of the Poisson problem in 92% of cases and for three solves in 95% of cases (data smoothing, 1x: 95%, 3x: 98%). Note, however, that our solver stops at a higher residual error than direct solvers. The strength of multigrid approaches is in settings where one needs a quick and relatively accurate solution. A direct solver is often preferable in settings where high accuracy is required.

To provide insight into the convergence of our approach compared to the other GMG schemes, we plot convergence for a data smoothing problem with $\alpha = 1 \times 10^{-3}$ for the Murex Romosus shape in Figure 1 and the same plot against number of iterations in Figure 9. Again we see that our approach is on par with Liu et al. [2021] and beats Shi et al. [2006] with a high margin. More convergence plots for data smoothing, including plots over the number of iterations, can be found in the supplement. These confirm our results. There are some outliers: for Red Circular Box, Shi et al. [2006] converges faster than the other GMG approaches and for Moses, Gravo MG slows down around a residual of 1×10^{-6}.

5.5 Applications

We evaluate our solver in three scenarios: data smoothing, a geometric flow, and physical simulation. We compare solving times to a sparse Cholesky solver, commonly used for these problems.

Data smoothing. For data smoothing, we consider an input function y on a surface and compute a smoother function x by minimizing a quadratic objective

$$(x - y)^\top M(x - y) + \alpha x^\top Sx + \beta x^\top SM^{-1}Sx. \tag{4}$$

The first term is a data term that penalizes deviation from the input function, the second and third terms are Laplace and bi-Laplace smoothing energies and $\alpha, \beta \in \mathbb{R}^{\geq 0}$ are parameters. Results are shown in Figures 4 and 5. The figures list timings for solving the linear systems with our method and Eigen's sparse Cholesky solver. When changing parameter α to adjust the amount of smoothing, the direct solver needs to compute a new matrix factorization resulting in significant solving-time differences compared to our solver, in particular, when the bi-Laplacian energy is included.

Conformal flow. As an example of a nonlinear geometric flow, we consider the conformal flow [Kazhdan et al. 2012]. For robustness, we use an implicit time-integration that requires solving a linear Laplace system for every time step. We show results in Figures 6 and 7 and compare our solving times to those of Eigen's sparse Cholesky solver. Since the system matrix changes every time step, the direct solver constantly needs to compute new factorizations, resulting in substantial differences when performing multiple steps.

Balloon inflation. As an example of a physical simulation, we consider the balloon inflation from [Skouras et al. 2012]. A surface mesh represents a thin-layered rubber balloon that undergoes membrane deformation subject to air pressure. For time-integration an implicit Euler scheme is used and the resulting nonlinear equations are solved using a Newton scheme. To find the descent direction a sparse linear system is solved. As for the geometric flow, due the simulation's nonlinearity, the system matrix changes with every time step, forcing the direct solver to compute a new factorization in every time step. Results and timings are shown in Fig 8.

6 CONCLUSION

We introduce Gravo MG, a surface multigrid method that features fast hierarchy construction, applicability to general surface representations, and fast convergence. Our experiments demonstrate excellent performance compared to other GMG and AMG methods and direct solvers.

Conceptually, our method deviates from the common paradigm of GMG to represent levels via watertight meshes obtained by edge collapse. We use the geometry of the surface, while AMG ignores it for hierarchy construction. This opens a new direction for GMG on manifolds, which are generally applicable and fast to build, hereby improving the scalability of geometry processing methods.

In future work, graph Voronoi diagrams could be used for point cloud processing. We are excited about the quality of the triangles we generate from the graph Voronoi diagrams and see a potential use, when fast triangulations or uniform samples on point clouds are needed. Regarding theory, it would be interesting to explore under which conditions this approach can provide guarantees regarding the triangulation. Further acceleration is still possible. An important aspect is parallelization of both the hierarchy construction and the solver. Our solver could also become a preconditioner for a Krylov method like GMRES or CG to accelerate convergence.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. This project is partly supported by the Indonesia Endowment Fund for Education (LPDP) via a doctoral scholarship for Ahmad Nasikun and by the Directorate of Research (Direktorat Penelitian) of the Universitas Gadjah Mada (UGM). We thank the authors of the meshes used in our experiments. A full list of meshes and attributions is listed at https://graphics.tudelft.nl/gravo_mg.

REFERENCES

Burak Aksoylu, Andrei Khodakovsky, and Peter Schröder. 2005. Multilevel Solvers for Unstructured Surface Meshes. *SIAM J. Sci. Comput.* 26, 4 (2005), 1146–1165. https://doi.org/10.1137/S1064827503430138

Marc Alexa and Max Wardetzky. 2011. Discrete Laplacians on General Polygonal Meshes. *ACM Trans. Graph.* 30, 4 (2011), 102:1–102:10.

Nathan Bell, Luke N. Olson, and Jacob Schroder. 2022. PyAMG: Algebraic Multigrid Solvers in Python. *Journal of Open Source Software* 7, 72 (2022), 4142. https://doi.org/10.21105/joss.04142

James H Bramble. 1993. *Multigrid Methods.* Chapman and Hall/CRC.

Achi Brandt. 1986. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comput.* 19, 1 (1986), 23–56. https://doi.org/10.1016/0096-3003(86)90095-0

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 (2008), 22:1–22:14. https://doi.org/10.1145/1391989.1391995

Timothy A. Davis, Sivasankaran Rajamanickam, and Wissam M. Sid-Lakhdar. 2016. A survey of direct methods for sparse linear systems. *Acta Numer.* 25 (2016), 383–566. https://doi.org/10.1017/S0962492916000076

Christian Dick, Marcus Rogowsky, and Rüdiger Westermann. 2016. Solving the Fluid Pressure Poisson Equation Using Multigrid - Evaluation and Improvements. *IEEE Trans. Vis. Comput. Graph.* 22, 11 (2016), 2480–2492. https://doi.org/10.1109/TVCG.2015.2511734

Martin Erwig. 2000. The graph Voronoi diagram with applications. *Networks* 36, 3 (2000), 156–163.

Bianca Falcidieno. 2007. Bringing the Semantics into Digital Shapes: the AIM@SHAPE Approach. In *Eurographics Italian Chapter Conference*, Raffaele De Amicis and Giuseppe Conti (Eds.). The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2007/103-106

Joachim Georgii and Rüdiger Westermann. 2006. A multigrid framework for real-time simulation of deformable bodies. *Comput. Graph.* 30, 3 (2006), 408–415. https://doi.org/10.1016/j.cag.2006.02.016

Seth Green, George Turkiyyah, and Duane W. Storti. 2002. Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *ACM Symposium on Solid Modeling and Applications.* ACM, 265–272. https://doi.org/10.1145/566282.566321

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Wolfgang Hackbusch. 1985. *Multi-grid methods and applications.* Springer series in computational mathematics, Vol. 4. Springer.

Philipp Herholz and Marc Alexa. 2018. Factor once: reusing cholesky factorizations on sub-meshes. *ACM Trans. Graph.* 37, 6 (2018), 230. 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. https://doi.org/10.1145/3414685.3417828

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. https://dl.acm.org/citation.cfm?id=237216

Intel Corporation. 2023. oneMKL PARDISO - Parallel Direct Sparse Solver Interface. https://www.intel.com/content/www/us/en/docs/onemkl/developer-reference-c/2023-1/onemkl-pardiso-parallel-direct-sparse-solver-iface.html. Accessed: April 17, 2023.

In-Yong Jeon, Kwang-Jin Choi, Tae-Yong Kim, Bong-Ouk Choi, and Hyeong-Seok Ko. 2013. Constrainable Multigrid for Cloth. *Comput. Graph. Forum* 32, 7 (2013), 31–39. https://doi.org/10.1111/cgf.12209

Misha Kazhdan and Hugues Hoppe. 2019. An Adaptive Multi-Grid Solver for Applications in Computer Graphics. *Comput. Graph. Forum* 38, 1 (2019), 138–150. https://doi.org/10.1111/cgf.13449

Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. 2012. Can mean-curvature flow be modified to be non-singular?. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1745–1754.

Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Symposium on Geometry Processing (ACM International Conference Proceeding Series, Vol. 256).* Eurographics Association, 61–70.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Transactions on Graphics* 36, 4 (2017).

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. https://doi.org/10.1145/2461912.2461992

Dilip Krishnan and Richard Szeliski. 2011. Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph.* 30, 6 (2011), 177. https://doi.org/10.1145/2070781.2024211

Jian Liang and Hongkai Zhao. 2013. Solving Partial Differential Equations on Point Clouds. *SIAM J. Sci. Comput.* 35 (2013), A1461–A1486.

Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface Multigrid via Intrinsic Prolongation. *ACM Trans. Graph.* 40, 4 (2021).

Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. In *Symposium on Computer*

Animation. Eurographics Association, 65–73.

Ahmad Nasikun and Klaus Hildebrandt. 2022. The Hierarchical Subspace Iteration Method for Laplace–Beltrami Eigenproblems. *ACM Trans. Graph.* 41, 2 (2022), 17:1–17:14. https://doi.org/10.1145/3495208

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. https://doi.org/10.1145/1015706.1015769

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. https://doi.org/10.2312/SCA/SCA07/181-190

Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. *ACM Trans. Graph.* 22, 3 (2003), 313–318. https://doi.org/10.1145/882262.882269

Ulrich Pinkall and Konrad Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experim. Math.* 2 (1993), 15–36.

Nicolas Ray and Bruno Lévy. 2003. Hierarchical Least Squares Conformal Map. In *Pacific Conference on Graphics and Applications*. IEEE, 263–270. https://doi.org/10.1109/PCCGA.2003.1238268

Nicholas Sharp and Keenan Crane. 2020. A Laplacian for Nonmanifold Triangle Meshes. *Comput. Graph. Forum* 39, 5 (2020), 69–80. https://doi.org/10.1111/cgf.14069

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. https://doi.org/10.1145/1141911.1142001

Xiaohan Shi, Hujun 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 Gross. 2012. Computational design of rubber balloons. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 835–844.

K. Stüben. 2001. A review of algebraic multigrid. *J. Comput. Appl. Math.* 128, 1 (2001), 281–309. https://doi.org/10.1016/S0377-0427(00)00516-1 Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

Shaodong Wang, Shuai Ma, Hui Zhao, and Wencheng Wang. 2022. A multigrid approach for generating harmonic measured foliations. *Computers & Graphics* 102 (2022), 380–389. https://doi.org/10.1016/j.cag.2021.10.003

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. https://doi.org/10.1111/cgf.13554

Max Wardetzky, Miklós Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. 2007. Discrete quadratic curvature energies. *Computer Aided Geometric Design* 24, 8-9 (2007), 499–518.

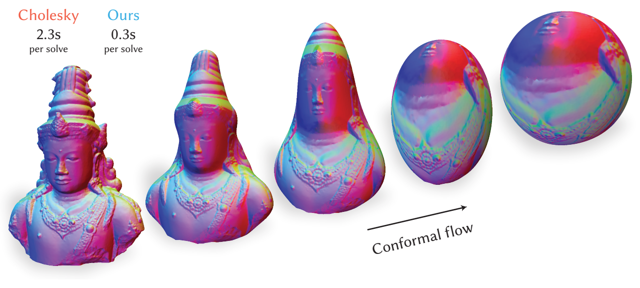

Figure 7: We compare the performance of our multigrid solver against a direct solver on conformal mean curvature flow on a nonmanifold mesh. (Indonesian statue model, 295k vertices).

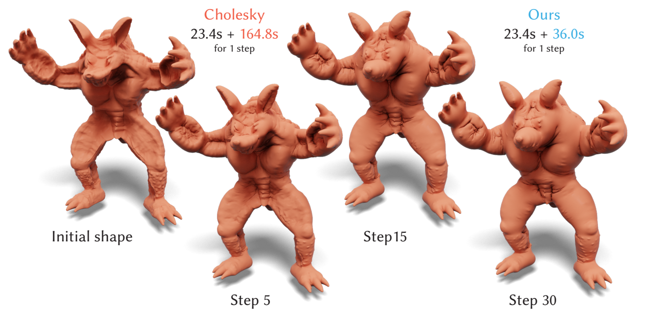

Figure 8: Simulation of balloon inflation (Model: Armadillo, 180k vertices).

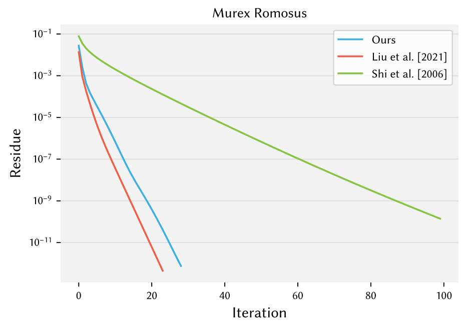

Figure 9: Residual over iteration count for Ours, Liu et al. [2021], and Shi et al. [2006] for data smoothing on Murex Romosus with α =1 $\times$ 10^{-3}. See Figure 1 for a plot over time.

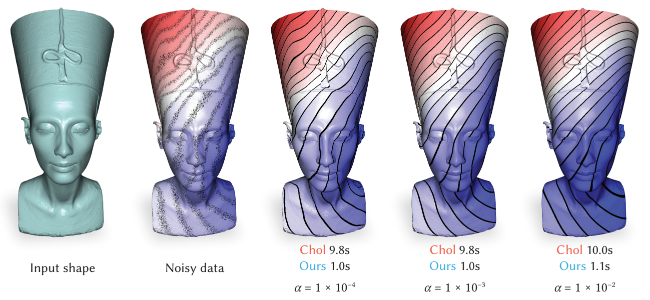

Figure 4: Smoothing of scalar data on a surface mesh with various parameter settings (Model: Nefertiti, 1m vertices) using the Dirichlet energy as smoothness energy.

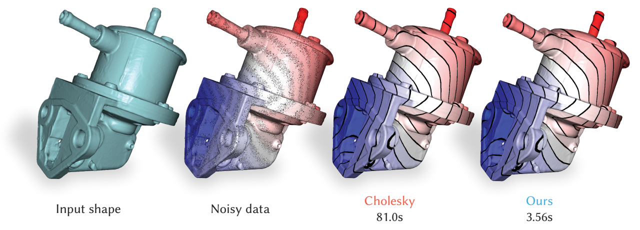

Figure 5: Smoothing of scalar data on a surface mesh (Model:Oilpump, 570k vertices) using a weighted sum of the Dirichlet energy and a bi-Laplacian energy as smoothness energy.

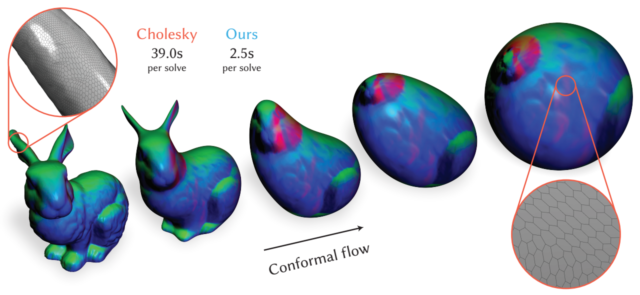

Figure 6: Conformal flow on polygon mesh. (Bunny model, 626k vertices).

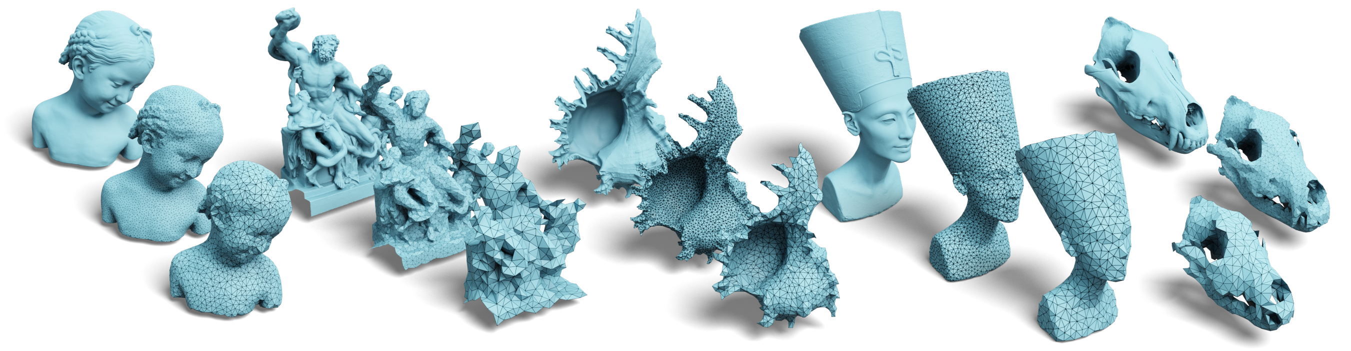

Figure 10: The input shapes and triangles in the last levels. Shapes: Bimba, Lakoon (non-manifold), Murex Romosus, Nefertiti, and Wolf Skull.

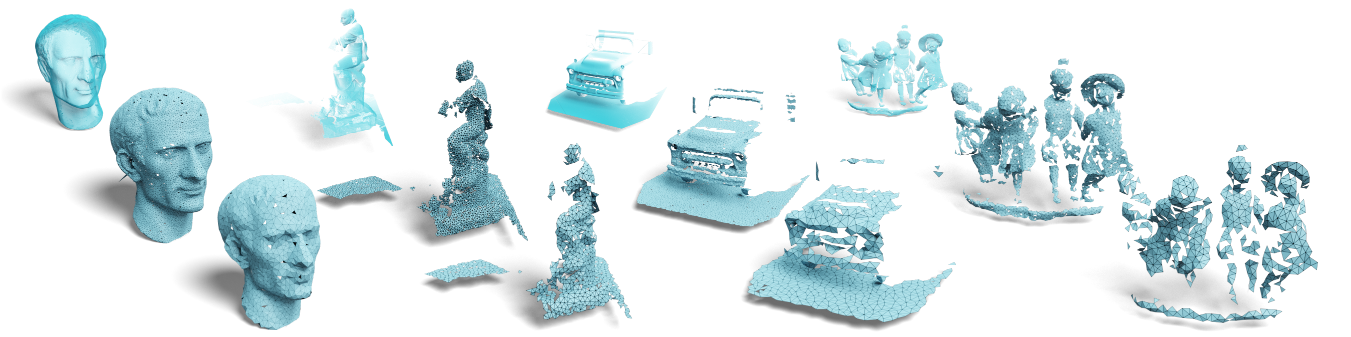

Figure 11: Input point clouds and considered triangles for the last two levels of the hierarchy. From left to right: Caesar, Ignatius, Truck, and Dancing Children.

Figure 12: All (non)manifold triangular meshes used in our experiments. Mosaic generated with code from Qingnan Zhou.