Computers & Graphics (2026)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

GravoTet: A Fast Multigrid Hierarchy Construction for Tetrahedral Meshes*

Marcel Padilla[a,*], Ruben Wiersma[a], Tim Huisman[b], Jackson Campolattaro[b], Olga Sorkine-Hornung[a], Klaus Hildebrandt[b]

[a]Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
[b]Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

ARTICLE INFO

Article history:
Received June 30, 2026

Keywords: geometry processing, multigrid solver, volumetric variational problems, poisson problem

ABSTRACT

Geometric multigrid (GMG) methods are a fundamental tool for efficiently solving large sparse linear systems. A requirement for GMG is a hierarchy of grids; however, many practical volumetric domains are available only as single, irregular tetrahedral meshes, making the construction of a multigrid hierarchy necessary. Existing approaches often trade off speed against hierarchy quality: remeshing- or coarsening-based methods can be expensive to construct, whereas graph-based techniques are fast but often yield weaker multigrid performance. We introduce GravoTet, which bridges this gap by combining geometric structure with graph-based efficiency to construct fast and effective multigrid hierarchies. GravoTet builds a vertex hierarchy and then generates graph-Voronoi diagrams whose dual cells define coarse tetrahedra, enabling rapid construction of multigrid levels. Boundary elements are explicitly prioritized during both sampling and tet generation to preserve boundary. In our evaluation, we solve Poisson and biharmonic problems on irregular tetrahedral meshes and compare GravoTet against state-of-the-art geometric multigrid, algebraic multigrid and direct solvers, demonstrating superior performance, particularly on large meshes.

1. Introduction

Geometric multigrid (GMG) methods play a central role in efficiently solving large, sparse linear systems arising from the discretization of partial differential equations, such as Poisson and biharmonic problems. These systems frequently appear in geometry processing applications, including volumetric deformation, smooth interpolation and geodesic distance computation, as well as in broader domains such as structural mechanics, fluid simulation and biomedical modeling. GMG methods are particularly attractive because they provide linear computational and memory complexity for a broad class of problems [2].

A fundamental requirement of GMG methods is the availability of a geometric hierarchy of grids. In some settings, such hierarchies arise naturally, for example, when a domain is initially described by a coarse grid that can be uniformly or adaptively refined (e.g., subdivided tetrahedra or cubic grids). In many practical applications, however, complex geometries are represented by a single irregular and unstructured grid on which a hierarchy must first be constructed. In this work, we focus on volumetric problems defined on domains represented by irregular tetrahedral meshes. Our goals are twofold: we aim to build a hierarchy that, first, enables fast multigrid convergence and, second, is fast to construct. Existing approaches struggle to balance these two objectives. Hierarchy constructions based on remeshing or mesh coarsening are expensive to compute [3], whereas graph-based methods enable faster hierarchy construction [4] but typically yield hierarchies that are less effective for solvers, resulting in slower convergence.

Motivated by this gap, we introduce GravoTet (Fig. 1), a fast method for constructing geometric multigrid hierarchies on

*This is the authors' preprint of an article published in *Computers & Graphics* (Special Section on SMI 2026). The version of record is available at https://doi.org/10.1016/j.cag.2026.104662.
*Corresponding author
e-mail: marcel.padilla@inf.ethz.ch (Marcel Padilla)

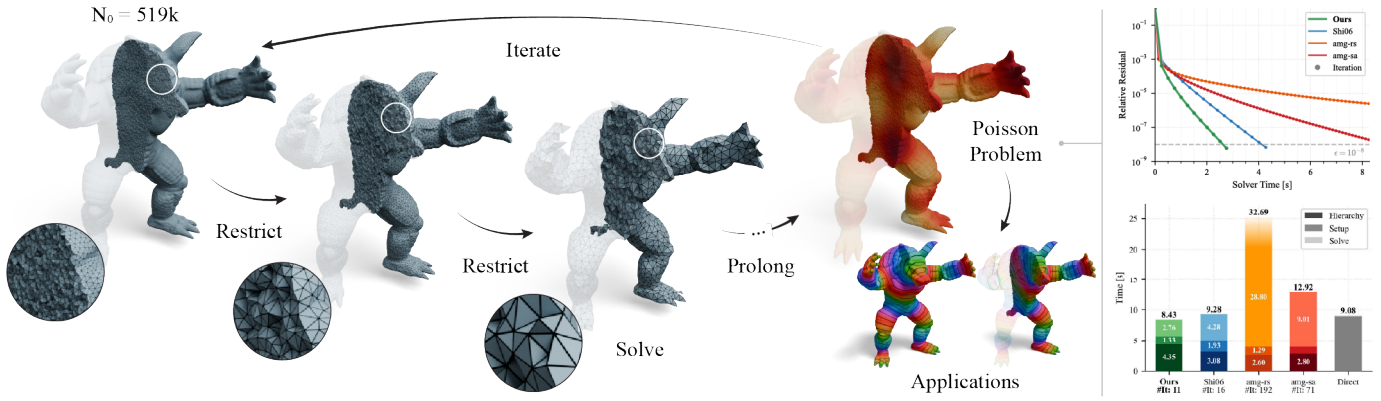

Fig. 1: We introduce GravoTet, a fast boundary-aware mesh hierarchy construction for tetrahedral meshes for geometric multigrid methods. The hierarchy is used for the efficient computation of barycentric prolongation weights. For volumetric Poisson and biharmonic problems, we obtain faster convergence rates in V-cycle iterations and have lower total computation times than CHOLMOD [1]. Restriction describes the mapping to a coarser mesh, while prolongation describes the mapping to a finer mesh.

tetrahedral meshes. This fast hierarchy construction is our first contribution. The intuition behind our approach mirrors Gravo MG [5], a geometric multigrid method for triangular meshes and surface point clouds; we construct a hierarchy of tetrahedral meshes, resembling Delaunay tetrahedralizations on an evenly spaced sampling of the domain. The tetrahedral meshes in the hierarchy are only used to define prolongation and restriction operators, rather than to assemble new coarse-level matrices. Therefore, we use a fast graph-based approximation of the Delaunay tetrahedralization: tetrahedra are formed from edges between neighboring cells in a graph-Voronoi diagram (hence, Gravo), mimicking the duality of Delaunay tetrahedra and 3D Voronoi cells. While this approach does not provide any guarantees on the validity of the tetrahedral meshes in the hierarchy, we empirically find that the resulting prolongation and restriction operators lead to fast convergence.

An important consideration for tetrahedral meshes is boundary preservation. If the prolongation- and restriction operators do not preserve the boundary well, multigrid iterations are spent, needlessly, on fixing the residual errors on the boundary. Therefore, as our second contribution, we propose two improvements for better boundary preservation. First, we augment the sampling step to prioritize sharp features on the boundary. Second, when computing the graph-Voronoi diagram and corresponding edges, we start with the graph on the boundary and then connect the interior. These changes have minor impact on runtime and improve fidelity to the boundary (Fig. 3), resulting in faster convergence. We display an overview of our method in Fig. 2.

We evaluate GravoTet through ablation studies, scaling experiments and comparisons against the closest competing tetrahedral GMG method [4], algebraic multigrid methods (AMG-RS, AMG-SA) [6, 7, 8] and the sparse direct solver CHOLMOD [1]. Our evaluation focuses on Poisson and biharmonic problems, demonstrating improved efficiency and faster convergence at higher resolutions.

2. Related work

Geometric multigrid (GMG) methods are well-established fast solvers that achieve linear computational and memory complexity for a broad class of partial differential equations [9, 10, 2]. They are widely used in graphics applications, including the simulation of elastic objects [11, 12], cutting [13] and fluid dynamics [14, 15], as well as in surface reconstruction [16] and gradient-domain processing [17]. A requirement for GMG methods is the availability of a geometric hierarchy of grids. For problems posed on complex domains represented by irregular tet meshes, constructing such hierarchies efficiently remains challenging. Mesh decimation approaches, such as edge-collapse coarsening and Delaunay-based remeshing, have been explored to address this difficulty [18, 19, 3, 20]. Although these approaches can produce high-quality hierarchies, the associated computational cost is substantial, limiting their practicality to scenarios where the hierarchy can be reused or precomputed. An alternative direction are graph-based multigrid techniques [4, 21], which avoid explicit grid construction by deriving inter-level operators directly from mesh connectivity. These methods generate hierarchies quickly, though in our experiments the resulting solvers tend to be less efficient and require more iterations to converge. GravoTet seeks to bridge this gap by combining the rapid hierarchy generation of graph-based approaches with geometric structure, enabling both fast and effective multigrid hierarchy construction.

A comparable situation arises in the context of constructing multilevel hierarchies for triangular meshes representing curved surfaces. Approaches based on mesh refinement [22] or intrinsic surface simplification [23] can yield high-quality hierarchies, but their computational cost is high. In contrast, the graph-based approach [4, 21] enables rapid hierarchy generation, though it typically leads to less effective multigrid solvers. A graph Voronoi–based hierarchy construction for triangle meshes was proposed in [5], serving as an inspiration for our work. GravoTet extends these ideas to tetrahedral meshes and incorporates boundary handling, which is absent in the

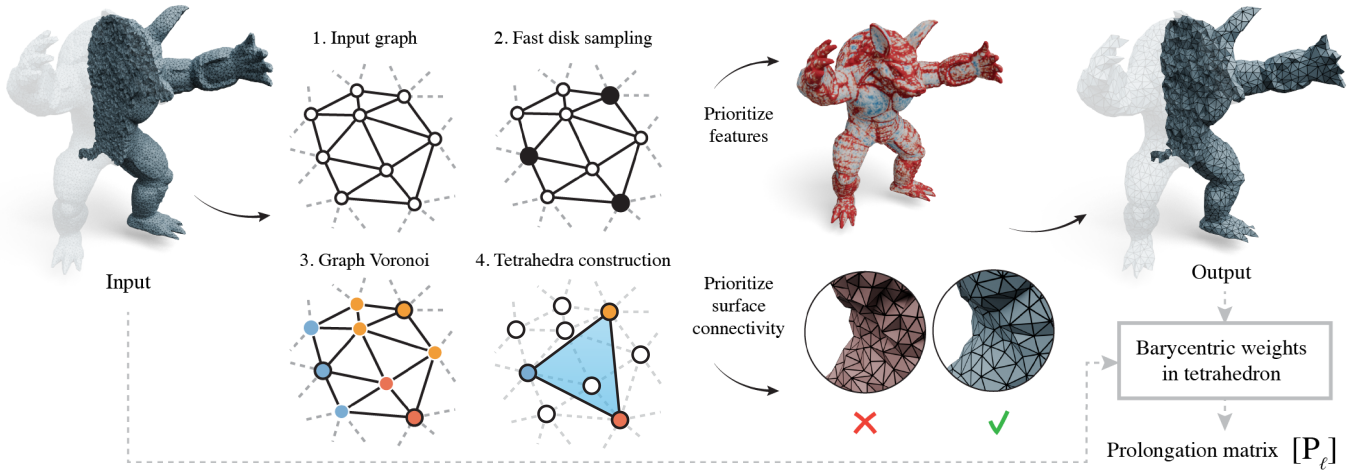

Fig. 2: An overview of our method. The input tetrahedral mesh is sampled and a tetrahedron is constructed using the graph-Voronoi approach (shown in 2D for illustration purposes). The sampling is directed by features and the simplex constructions focuses on the surface. The resulting coarser mesh is then used to compute barycentric weights.

closed-surface setting considered in [5].

Algebraic multigrid (AMG) methods provide an alternative that is independent of the underlying geometry of the domain, constructing multilevel hierarchies solely from the discretized operator and its sparse matrix structure [24, 25]. Widely used strategies such as Ruge–Stüben coarsening [6] and smoothed aggregation [7] are readily accessible through libraries like PyAMG [8]. AMG methods retain many core advantages of multigrid techniques, including rapid hierarchy construction and favorable scaling compared with traditional iterative solvers. However, their lack of geometric awareness often leads to slower convergence when compared to GMG approaches, which can exploit mesh and domain structure more effectively [26, 27], and means that the hierarchy needs to be recomputed when the linear system changes.

Sparse direct solvers constitute another widely used class of methods for solving sparse linear systems [28]. Guided by the benchmarking recommendations in [29], we use CHOLMOD [1], one of the most efficient and robust solvers in this category, in our experimental comparisons. While highly effective for moderate-sized problems, the factorization cost of sparse direct solvers grows rapidly with increasing problem size and becomes substantially higher when the sparsity of the system deteriorates. As a result, its computational expense can far exceed that of geometric multigrid methods, which maintain linear-scaling efficiency and remain more suitable for large problems.

3. Background

We consider the linear system arising from the discretisation of an elliptic problem, such as a Poisson or biharmonic problem, on a domain in $\mathbb{R}^3$. The discretisation leads to a system

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{N\times N}$ is a large, sparse, symmetric, positive definite matrix and $x, b \in \mathbb{R}^N$. The discrete domain is represented by a tetrahedral mesh $\mathcal{M}$ with $N \in \mathbb{N}$ vertices, and the space $\mathbb{R}^N$ represents functions on the domain, for example, the nodal vectors resulting from a finite-element discretization.

Multigrid solvers operate on a hierarchy of linear spaces $\mathbb{R}^{N_1}, \mathbb{R}^{N_2}, \ldots, \mathbb{R}^{N_L}$ with $N_1 > N_2 > \cdots > N_L$, where $L \in \mathbb{N}$ denotes the number of levels and $N_1 = N$. Prolongation and restriction between adjacent levels are performed by the prolongation matrices $P_\ell \in \mathbb{R}^{N_\ell \times N_{\ell+1}}$ and their transposes P_ℓ^T, respectively. Using compositions of these operators, each linear space in the hierarchy can be mapped to a subspace of the finest-level space $\mathbb{R}^{N_1}$ via

$$P_1 P_2 \ldots P_{\ell-1}\, \mathbb{R}^{N_\ell} \subset \mathbb{R}^{N_1}, \tag{2}$$

so the hierarchy forms a sequence of nested function spaces.

In a Galerkin multigrid framework, the system matrix on each level is defined recursively by

$$A^{\ell+1} = P_\ell^T A^\ell P_\ell \tag{3}$$

with $A^1 = A$. The prolongation matrices thereby determine both the quality of the coarse-level representation and the fidelity of the inter-level transfer. Consequently, the construction of P_ℓ is a central ingredient of any multigrid method and directly influences its convergence behavior [30].

Multigrid methods solve $Ax = b$ by combining two complementary processes across a hierarchy of L levels: *relaxation* (smoothing), which damps high-frequency error components using a classical iterative method such as Gauss–Seidel, and *coarse-grid correction*, which transfers the remaining low-frequency residual to a coarser level where it can be resolved more efficiently. This process can be applied recursively, progressing from the finest level ($\ell = 1$) to the coarsest ($\ell = L$), where the system is small enough for direct solves to be viable, and back up, forming a V-cycle.

4. GravoTet

Our goal is to design a method that constructs hierarchies quickly and produces effective hierarchies for multigrid solves.

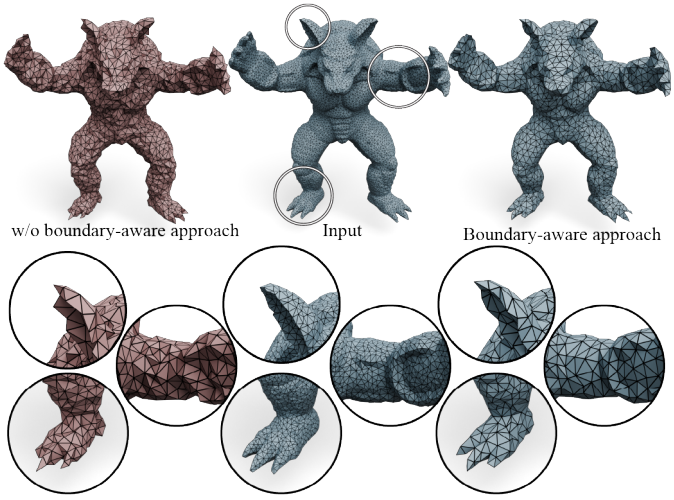

Fig. 3: The Armadillo input tetrahedral mesh (middle) is coarsened using our graph-Voronoi-based approach. We visually compare the results of the boundary-aware approach (right) with those without.

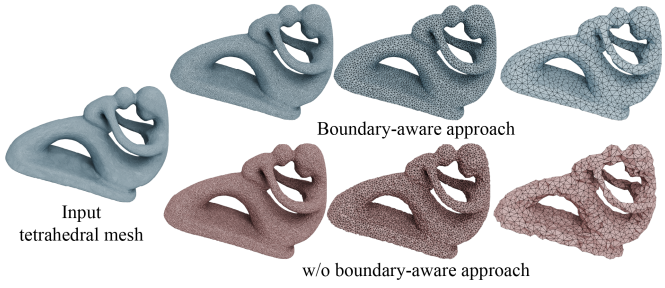

Fig. 4: A comparison of the mesh hierarchy produced by applying our boundary-aware approach vs. omitting it for the first three levels of the fidelity mesh.

As outlined in the background section, the multigrid hierarchy is fully described by the prolongation matrices. Our construction of the prolongation matrices proceeds in several stages, as outlined in Algorithm 1. GravoTet constructs a tetrahedral mesh on an evenly distributed point sampling for each coarse level. We mimic a Delaunay tetrahedralization on these point samples by computing tetrahedra from the edges between neighboring cells in a graph-Voronoi diagram. We pay special attention to boundary preservation, to improve convergence. In the following, we discuss the stages of our algorithm one by one.

Prioritized fast disk sampling. The first step in constructing each level of the hierarchy is vertex sampling. To this end, we employ a variant of *fast disk sampling* (FDS) [5]. FDS executes a single sweep over the vertices in the fine level. Whenever a vertex is selected, all neighboring vertices within a radius $r = \sqrt[3]{2}\,\bar{e}$, are excluded, where $\bar{e}$ denotes the average edge length of the finer level. The exclusion check is restricted to the 2-ring of the selected vertex. This produces a uniformly distributed set of coarse vertices in linear time (see Sec. S1 of the supplementary material). To preserve boundary fidelity, we order the boundary vertices of the fine mesh $\mathcal{M}$ according to a sharpness-score. The sharpness $f \in \mathbb{R}$ is defined as the sum of the dihedral angles between the normals of all incident boundary triangles. Vertices in locally flat regions have $f = 0$, whereas, for example, the corners of a cube yield $f = \frac{3}{2}\pi$. The interior vertices are appended to the sorted list in random order. Since FDS preserves the order of its input, the resulting coarse vertex set naturally inherits the prioritization of the fine mesh. Therefore, we only need to compute the sharpness-score and sorting as a pre-processing step. The samples following the pre-computed ordering retains sharp features down to the coarsest level. This behavior can be observed in Figs. 3 and 4, where both the surface and fine geometric details are visibly better preserved.

Algorithm 1 Construction of levels and prolongation matrices

Input: Tetrahedral mesh $\mathcal{M}$, number of vertices of lowest level $N_{\min}$
Output: Prolongation matrices $P_1, \ldots, P_{L-1}$
Preprocessing:
Compute sharpness for boundary vertices of $\mathcal{M}$
Create vertex priority list V_1 of vertices of $\mathcal{M}$
$\mathcal{N}_1 \leftarrow$ ExtractNeighborGraph($\mathcal{M}$)
Build Hierarchy:
$\ell \leftarrow 1$; $N_1 \leftarrow$ number of vertices in $\mathcal{M}$
while $N_\ell > N_{\min}$ **do**
    $V_{\ell+1} \leftarrow$ PrioritizedFastDiskSampling($V_\ell, \mathcal{N}_\ell$)
    $N_{\ell+1} \leftarrow$ number of points in $V_{\ell+1}$
    $C_{\ell+1} \leftarrow$ BoundaryPrioritizedGraphVoronoi($V_{\ell+1}, \mathcal{N}_\ell$)
    $\mathcal{N}_{\ell+1} \leftarrow$ ComputeNeighbors($V_{\ell+1}, C_{\ell+1}, \mathcal{N}_\ell$)
    $\mathcal{M}_{\ell+1} \leftarrow$ Tetrahedralize($V_{\ell+1}, \mathcal{N}_{\ell+1}$)
    $P_\ell \leftarrow$ BuildProlongationMatrix($V_\ell, \mathcal{M}_{\ell+1}$)
    $\ell \leftarrow \ell + 1$
end while

A vertex v of $\mathcal{M}_1$ is classified as a boundary vertex if the sum of the solid angles of its incident tetrahedra is strictly less than 4π. Similarly, an edge connecting two boundary vertices is classified as a boundary edge if the sum of the dihedral angles of its adjacent tetrahedra is strictly less than 2π. These classification rules are chosen because they are local and easily parallelizable. The boundary extraction is only applied to the input tetrahedral mesh $\mathcal{M}_1$, which is always manifold.

Boundary-first Voronoi connectivity. The next step is to establish the connectivity among the newly sampled coarse points. This is achieved by first computing a boundary-prioritized graph Voronoi diagram, followed by the construction of the neighborhood graph. The graph Voronoi diagram [31] uses the points of the finer level $V_{\ell+1}$ as seeds. Each seed i receives all points in V_ℓ that are closest to it in graph distance. These cells are obtained efficiently using a multisource Dijkstra algorithm.

To facilitate boundary preservation, we compute the Voronoi cells on the boundary first and then on the interior. We initially restrict the Dijkstra propagation to the boundary mesh $\mathcal{B}_\ell$, using only the boundary samples as sources. Once all boundary vertices have been assigned to their surface clusters, the distance propagation is extended to the full volumetric graph $\mathcal{M}_\ell$. This stratified approach maintains geometric fidelity at the boundary while producing coherent volumetric connectivity.

After the graph Voronoi cells are computed, the neighbor graph for the points $V_{\ell+1}$ is computed. For points $i, j \in V_{\ell+1}$, we add an edge $\{i, j\}$ to $N_{\ell+1}$, if there is an edge in N_ℓ that connects a point of the Voronoi cell of i with a point of the Voronoi cell of j.

Tetrahedralization. We proceed to generate the tetrahedra at level $\ell + 1$, based on the neighbor graph $N_{\ell+1}$. To this end, we employ a dualization of the graph-Voronoi diagram, analogous to how Delaunay tetrahedralizations arise as duals of classical Voronoi diagrams. Specifically, we create a tetrahedron for every quadruple of vertices $\{i, j, k, l\} \subset V_{\ell+1}$ for which the neighbor graph $N_{\ell+1}$ contains all six pairwise edges: $\{i, j\}$, $\{j, k\}$, $\{k, i\}$, $\{i, l\}$, $\{j, l\}$, and $\{k, l\}$.

Prolongation matrices. After the tetrahedron list $\mathcal{M}_{\ell+1}$ for the new level has been constructed, we assemble the prolongation matrix $P_\ell \in \mathbb{R}^{N_\ell \times N_{\ell+1}}$. For each vertex i on level ℓ, we search for a tetrahedron on level $\ell+1$ that contains it. This search is fast as it can be restricted to those tetrahedra that contain the coarse-level point that is the seed of the graph-Voronoi cell containing i. If such a tetrahedron is found, the prolongation weights are given by the barycentric coordinates of i with respect to that tetrahedron; these weights are the non-zero entries of the i^{th} row of P_ℓ. If a fine-level vertex is not in any tetrahedron, its coordinates are projected onto the nearest valid geometric entity: either a face, an edge, or a single coarse vertex. Lower-dimensional barycentric coordinates on these structures naturally define the corresponding interpolation weights. Since each fine vertex is associated with at most one simplex (of dimension up to three), the resulting prolongation matrix remains highly sparse, containing at most four nonzero entries per row. Because the graph-Voronoi tetrahedralization is approximate, the resulting structure is in general a simplicial complex rather than a strict tetrahedral mesh: thin or isolated features may coarsen into triangles, edges, or even isolated vertices, and overlapping tetrahedra can occasionally appear. When a fine vertex projects onto a lower-dimensional simplex, its row of P_ℓ contains fewer than four nonzero entries.

Two robustness questions arise from the approximate coarse complex. First, a fine vertex may lie inside several overlapping tetrahedra. We assign it to one by a fixed local rule and compute the barycentric coordinates with respect to that tetrahedron, which uniquely determines valid weights in the rows of P_ℓ. Second, coarse tetrahedra may be inverted. The barycentric containment test relies on signed ratios that cancel under orientation reversal, so inverted tetrahedra are treated the same as correctly oriented ones. We discuss both points in detail in Sec. S4 of the supplementary material.

5. Results

We evaluate the computational performance, convergence properties and geometric robustness of our method with a series of experiments. Our evaluation is primarily focused on classical volumetric Poisson and biharmonic problems, which are archetypal linear systems relevant to physical simulation and geometry processing and are costly to solve, especially for high-resolution tetrahedral meshes. We evaluate these problems on varying tetrahedral meshes and resolutions and benchmark the results against the most relevant solvers. We plan to release the code.

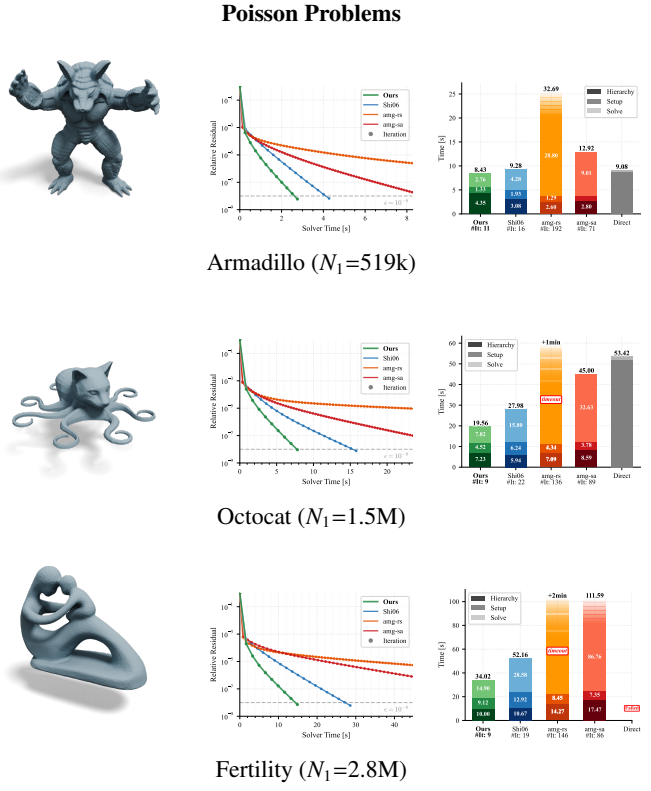

Fig. 5: Poisson problem performance comparison. Left: rendered input mesh. Center: relative residual vs. wall-clock time for the V-cycle iterations. Right: execution time breakdown. We provide more graphs in the supplementary material.

5.1. Comparisons & metrics

We benchmark GravoTet against the following representative methods motivated by our discussion in Section 2:

- **Shi06** [4]: A fast, graph-based multigrid method that can be used on tetrahedral meshes.
- **AMG-RS** [6] and **AMG-S** [7]: Standard algebraic multigrid methods, specifically Ruge-Stüben and smoothed aggregation, which are easily accessible through the PyAMG library [8].
- **CHOLMOD** [1]: A state-of-the-art sparse direct solver, serving as a baseline metric for total execution time. Selected based on the benchmarking recommendations in [29].

To isolate the effects of our core contributions, all multigrid methods are evaluated using the same generic V-cycle implementation, differing only in the prolongation operators P_ℓ used, isolating P_ℓ from each method's own solver optimizations. All methods use two pre- and two post-smoothing Gauss-Seidel sweeps per level. We keep each method's default level count.

Only our method could solve all systems within a set amount of time. For other methods, we report failures to converge, and in the case of timeouts we report the relevant time budget (Table 1).

Metrics. We split the total computation time (t_{total}) into the build time of the prolongation matrices (t_{build}) and the solve time (t_{solve}), which includes the setup of the solver and the V-cycle iterations. The relevant timing depends on the application: if only a single solve is required, t_{total} should be considered; if multiple systems should be solved on the same domain (e.g., in physical simulations or user interaction), t_{solve} becomes more relevant. Note that, in the case of multiple *different* linear systems, the direct solver cannot reuse the factorization and should be evaluated on the total time.

5.2. Evaluation applications

The performance of our method is evaluated on solving two common linear systems of equations in geometry processing and physically based simulation.

Volumetric Poisson equations frequently arise when computing scalar fields over bounded 3D domains. Prominent examples include the pressure projection step in Eulerian fluid simulations [14, 32, 33], as well as volumetric mesh deformation techniques [4, 12]. In geometry processing, similar linear systems govern the computation of harmonic coordinates, smooth volumetric skinning weights, and geodesic distances [34]. We solve:

$$S\,x = M y, \tag{4}$$

where S is the discrete cotangent stiffness matrix, M is the barycentric diagonal mass matrix, x the unknown scalar field, and y is a given source field. We assemble S and M using *libigl* [35]. We use a Dirichlet boundary condition composed of cosine waves.

We use the relative residual to assess convergence. For systems of the form (4), we employ the M^{-1}-norm[1], leading to the stopping criterion

$$\|S\,x - M y\|_{M^{-1}} < \epsilon\,\|M y\|_{M^{-1}}, \tag{5}$$

where we set $\epsilon = 10^{-8}$.

Biharmonic equations arise for smoothing tasks or when modeling elastic deformation. We apply the Dirichlet boundary condition to two vertices in the domain, and solve:

$$S\,M^{-1}\,S\,x = M y. \tag{6}$$

Because the biharmonic problem is much worse conditioned compared to the Poisson problem, we relax our stopping criterion to use a threshold $\epsilon = 10^{-5}$.

5.3. Comparisons

We evaluate GravoTet across a diverse set of tetrahedral meshes that exhibit distinct geometric and topological challenges and with varying genera. We use commonly available watertight surface meshes and generate tetrahedral meshes via *tetGen* [37] (see Sec. S2 of the supplementary material for mesh generation parameters) with initial vertices N_1 ranging from 22k to 2.8M. Table 1 displays many examples, with vertex counts, timings, and required iterations for all methods. We also plot relative residual convergences over time for the different multigrid algorithms on representative Poisson problems (Fig. 5) and biharmonic problems (Fig. 6). Extended tables and similar plots for other meshes can be found in the supplementary materials of this work.

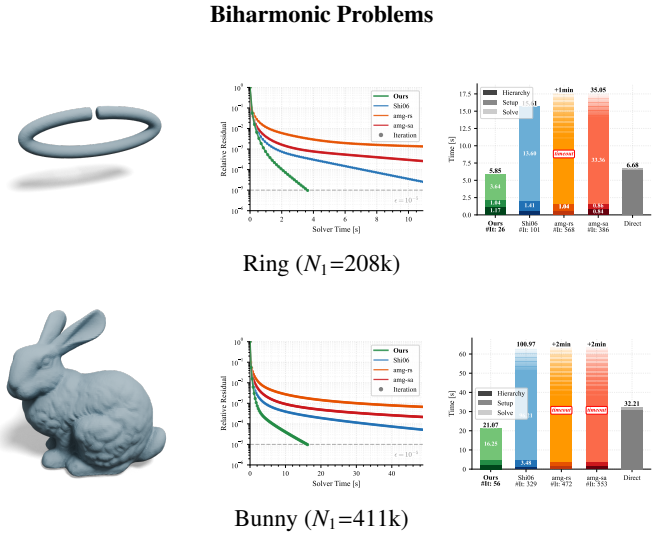

Fig. 6: Biharmonic problem performance comparison. Left: rendered input mesh. Center: relative residual vs. wall-clock time for the V-cycle iterations. Right: execution time breakdown. We provide more graphs in the supplementary material.

GravoTet converges on all 26 Poisson- and 26 biharmonic problems, where all other methods fail or timeout at least once, making GravoTet the most robust approach in our experiments. GravoTet achieves the fastest total time in 42 of 52 cases (80%). Our method requires the lowest number of iterations, substantiating our claim of higher-quality prolongation matrices. The closest competing method is Shi06, which has lower build times, but requires more iterations and longer solving time. GravoTet is most successful for high vertex counts ($> 100k$). For lower resolutions, AMG-SA's negligible build cost wins out.

The gap widens for the more ill-conditioned biharmonic equation: Shi06 fails to converge within the set time limit on 10 of 26 problems. On the remaining problems, GravoTet requires an average of 4.3× fewer iterations. AMG-RS and AMG-SA each converge on only 8 of 26 biharmonic problems within the set time limit. The direct solver, CHOLMOD, is never the fastest method. Of 52 total cases, CHOLMOD fails on 15 cases due to high memory demand and outperforms GravoTet on only 6 smaller meshes.

Sparsity and iteration count. To assess the algebraic efficiency of our generated multigrid hierarchy, we consider the number

[1] The M^{-1}-norm is $\|S\,x\|_{M^{-1}} = \sqrt{(S\,x)^T M^{-1} S\,x}$. We use the M^{-1}-norm as $S\,x$ is an integrated quantity and $M^{-1}S\,x$ is the corresponding pointwise quantity. The M-norm of $S\,x$ equals the M^{-1}-norm of $M^{-1}S\,x$, $\|S\,x\|_{M^{-1}} = \sqrt{(S\,x)^T M^{-1} S\,x} = \sqrt{(M^{-1}S\,x)^T M M^{-1} S\,x} = \left\|M^{-1}S\,x\right\|_M$. For more background, we refer to [36].

Table 1: Detailed performance metrics for comparison models of Poisson and biharmonic problems. Our method consistently archives the lowest number of iterations while also achieving the lowest total time in most cases. Especially for the ill-conditioned biharmonic problem our method's faster convergence rates result in a strong advantage.

Model	**#Vert N_1**	**Ours**				**Shi06**				**AMG-RS**				**AMG-SA**				**Direct (CHOLMOD)**		
		Build	**Solve**	**Total**	**#It**	**Build**	**Solve**	**Total**	**#It**	**Build**	**Solve**	**Total**	**#It**	**Build**	**Solve**	**Total**	**#It**	**Fact.**	**Subst.**	**Total**
POISSON PROBLEMS																				
Fandisk	22k	0.09	0.12	0.21	**7**	0.06	0.19	0.25	10	0.05	0.22	0.27	49	0.07	0.11	**0.18**	39	0.17	0.00	0.18
Homer	33k	0.18	0.18	0.36	**8**	0.10	0.34	0.43	13	0.08	0.31	0.39	46	0.09	0.17	**0.26**	41	0.27	0.01	0.27
Teapot	52k	0.29	0.48	0.76	**7**	0.16	0.46	0.62	11	0.14	0.65	0.79	60	0.13	0.38	**0.51**	46	0.61	0.02	0.63
Horse	74k	0.46	0.39	0.84	**9**	0.22	0.64	0.86	14	0.20	1.01	1.20	62	0.19	0.60	**0.80**	48	0.78	0.02	0.80
Spike	102k	0.57	0.47	**1.04**	**6**	0.32	1.07	1.39	13	0.31	1.45	1.76	66	0.28	1.03	1.31	61	1.41	0.05	1.46
Max Planck	157k	0.76	0.80	**1.56**	**8**	0.56	1.89	2.44	14	0.48	4.15	4.63	116	0.64	1.71	2.35	58	2.26	0.07	2.32
Ring	208k	1.08	0.99	**2.07**	**6**	0.64	1.87	2.51	16	0.64	2.71	3.35	49	0.96	1.82	2.78	43	2.46	0.06	2.52
Rocker Arm	262k	0.99	1.41	**2.40**	**8**	0.83	2.45	3.28	16	0.84	4.60	5.44	73	1.12	3.04	4.16	58	3.35	0.09	3.44
Bob	309k	1.44	1.81	**3.24**	**7**	1.22	3.12	4.34	15	1.05	8.03	9.08	106	1.37	4.25	5.61	67	4.70	0.13	4.83
Cylinder	310k	2.45	1.87	**4.32**	**6**	1.09	3.38	4.47	17	1.27	7.04	8.31	87	1.65	4.73	6.38	64	4.75	0.14	4.88
Bunny	411k	2.12	2.71	**4.83**	**7**	1.28	4.43	5.72	14	1.48	14.63	16.11	142	1.95	5.69	7.64	60	7.74	0.20	7.94
Cube 80^3	512k	5.13	4.83	9.96	**8**	3.36	6.94	10.30	17	5.07	8.70	13.78	17	2.44	6.96	**9.40**	57	11.04	0.29	11.33
Armadillo	519k	4.35	4.08	**8.43**	**11**	3.08	6.20	9.28	16	2.60	30.09	32.69	192	2.80	10.12	12.92	71	8.87	0.21	9.08
Nefertiti	617k	4.32	5.10	**9.42**	**10**	2.54	7.59	10.13	15	2.41	41.30	43.70	233	3.16	11.48	14.64	67	13.78	0.35	14.13
Spot	709k	6.02	5.52	11.54	**8**	2.47	8.95	**11.42**	16	2.87	41.72	44.59	205	3.67	13.40	17.07	69	18.57	0.48	19.05
Spike Ball	829k	7.66	7.01	**14.67**	**9**	3.47	11.35	14.81	17	3.08	*timeout*	*> 1min*	285	6.30	16.09	22.39	63	35.79	1.22	37.01
Beast	881k	7.06	7.61	**14.68**	**10**	5.11	10.83	15.95	17	6.47	45.96	52.43	161	6.44	19.04	25.49	73	*failed*	*failed*	*failed*
Cube 100^3	1.0M	3.39	14.06	**17.45**	**8**	4.57	15.18	19.75	19	16.34	22.00	38.34	18	8.48	17.85	26.33	50	115.79	0.96	116.76
Dragon	1.0M	9.58	9.26	**18.84**	**12**	6.55	12.68	19.22	17	7.96	51.89	59.85	160	10.73	21.33	32.06	72	*failed*	*failed*	*failed*
Happy Buddha	1.2M	6.88	10.83	**17.70**	**13**	5.11	17.29	22.41	19	5.76	*timeout*	*> 1min*	184	6.90	27.29	34.19	87	45.51	0.65	46.16
Octocat	1.5M	7.23	12.34	**19.56**	**9**	5.94	22.04	27.98	22	7.09	*timeout*	*> 1min*	136	8.59	36.41	45.00	89	51.82	1.60	53.42
Kitten	1.5M	8.53	12.73	**21.26**	**8**	6.26	22.17	28.43	17	7.53	*timeout*	*> 2min*	287	9.04	30.98	40.02	72	60.04	5.81	65.85
Lucy	1.9M	21.93	18.41	**40.34**	**16**	12.43	28.30	40.73	20	13.45	185.55	199.00	260	19.98	38.41	58.38	89	52.14	1.38	53.51
Cow	2.0M	23.58	19.83	**43.41**	**10**	14.51	53.14	67.64	19	15.38	*timeout*	*> 4min*	237	23.12	87.46	110.58	87	*failed*	*failed*	*failed*
XYZ Dragon	2.4M	14.19	25.75	**39.94**	**14**	10.50	39.65	50.15	21	12.96	*timeout*	*> 2min*	153	15.68	69.47	85.15	98	*failed*	*failed*	*failed*
Fertility	2.8M	10.00	24.02	**34.02**	**9**	10.67	41.49	52.16	19	14.27	*timeout*	*> 2min*	146	17.47	94.11	111.59	86	*failed*	*failed*	*failed*
BIHARMONIC PROBLEMS																				
Fandisk	22k	0.12	0.32	**0.44**	**24**	0.08	0.67	0.76	52	0.06	2.43	2.48	290	0.06	1.18	1.24	156	0.52	0.01	0.53
Homer	33k	0.23	0.63	**0.87**	**20**	0.14	1.91	2.05	94	0.08	19.27	19.35	911	0.09	6.48	6.57	369	1.39	0.03	1.42
Teapot	52k	0.33	1.30	**1.63**	**39**	0.16	3.30	3.46	99	0.13	14.99	15.13	613	0.13	6.33	6.46	312	1.63	0.05	1.68
Horse	74k	0.45	1.49	**1.94**	**26**	0.25	5.70	5.95	133	0.20	58.91	59.11	1803	0.19	13.99	14.18	518	2.05	0.05	2.11
Spike	102k	0.66	3.34	**4.00**	**47**	0.31	15.36	15.67	236	0.29	*timeout*	*> 1min*	1156	0.27	33.29	33.56	778	4.29	0.11	4.40
Max Planck	157k	1.00	4.91	**5.91**	**44**	0.51	26.74	27.25	264	0.47	*timeout*	*> 1min*	721	0.61	*timeout*	*> 1min*	900	7.24	0.17	7.42
Ring	208k	1.17	4.68	**5.85**	**26**	0.60	15.01	15.61	101	0.63	*timeout*	*> 1min*	568	0.84	34.22	35.05	386	6.53	0.15	6.68
Rocker Arm	262k	1.47	5.76	**7.23**	**24**	1.64	27.48	29.12	153	0.83	*timeout*	*> 1min*	442	1.13	*timeout*	*> 1min*	539	9.41	0.22	9.63
Bob	309k	2.51	10.30	**12.81**	**42**	1.01	51.16	52.17	241	1.03	*timeout*	*> 1min*	344	1.42	*timeout*	*> 1min*	415	14.37	0.33	14.69
Cylinder	310k	1.96	9.81	**11.77**	**38**	1.08	41.47	42.56	193	1.07	*timeout*	*> 1min*	346	1.38	*timeout*	*> 1min*	422	12.81	0.30	13.11
Bunny	411k	2.22	18.85	**21.07**	**56**	1.28	99.69	100.97	329	1.47	*timeout*	*> 2min*	472	1.94	*timeout*	*> 2min*	553	30.82	1.39	32.21
Cube 80^3	512k	2.64	26.93	**29.57**	80	1.65	78.27	79.92	227	4.63	35.38	40.01	**51**	2.25	*timeout*	*> 2min*	507	40.13	0.97	41.10
Armadillo	519k	2.81	19.38	**22.20**	**39**	1.75	118.48	120.23	302	1.88	*timeout*	*> 2min*	358	2.53	*timeout*	*> 2min*	410	31.23	0.88	32.11
Nefertiti	617k	2.81	25.82	**28.62**	**43**	2.22	*timeout*	*> 2min*	266	2.39	*timeout*	*> 2min*	290	3.12	*timeout*	*> 2min*	326	56.01	7.24	63.25
Spot	709k	3.14	32.48	**35.61**	**48**	2.61	*timeout*	*> 2min*	230	2.87	*timeout*	*> 2min*	250	3.67	*timeout*	*> 2min*	283	*failed*	*failed*	*failed*
Spike Ball	829k	8.93	60.10	**69.03**	**61**	4.45	*timeout*	*> 4min*	353	4.93	*timeout*	*> 4min*	456	6.80	*timeout*	*> 4min*	303	*failed*	*failed*	*failed*
Beast	881k	6.30	42.16	**48.46**	**51**	4.94	*timeout*	*> 4min*	216	3.50	*timeout*	*> 4min*	236	3.23	*timeout*	*> 4min*	267	*failed*	*failed*	*failed*
Cube 100^3	1.0M	6.07	63.66	**69.72**	90	3.42	242.62	246.03	335	10.75	96.51	107.26	**69**	4.86	*timeout*	*> 4min*	484	*failed*	*failed*	*failed*
Dragon	1.0M	10.06	45.13	**55.19**	**31**	5.93	*timeout*	*> 4min*	193	6.60	*timeout*	*> 4min*	239	8.84	*timeout*	*> 4min*	313	*failed*	*failed*	*failed*
Happy Buddha	1.2M	11.39	72.70	**84.09**	**44**	8.09	*timeout*	*> 6min*	239	8.56	*timeout*	*> 6min*	264	11.52	*timeout*	*> 6min*	298	*failed*	*failed*	*failed*
Octocat	1.5M	7.98	53.31	**61.29**	**27**	5.97	*timeout*	*> 4min*	207	7.05	*timeout*	*> 4min*	224	8.74	*timeout*	*> 4min*	257	*failed*	*failed*	*failed*
Kitten	1.5M	8.47	75.77	**84.23**	**46**	6.18	*timeout*	*> 6min*	288	7.36	*timeout*	*> 6min*	322	8.96	*timeout*	*> 6min*	369	*failed*	*failed*	*failed*
Lucy	1.9M	11.84	89.17	**101.01**	**43**	8.02	*timeout*	*> 6min*	228	9.45	*timeout*	*> 6min*	255	11.29	*timeout*	*> 6min*	294	*failed*	*failed*	*failed*
Cow	2.0M	15.58	144.04	**159.62**	**46**	8.41	*timeout*	*> 8min*	140	16.38	*timeout*	*> 8min*	220	25.84	*timeout*	*> 8min*	342	*failed*	*failed*	*failed*
XYZ Dragon	2.4M	13.86	121.74	**135.61**	**34**	12.45	*timeout*	*> 8min*	200	12.96	*timeout*	*> 8min*	214	15.75	*timeout*	*> 8min*	246	*failed*	*failed*	*failed*
Fertility	2.8M	14.14	150.77	**164.91**	**36**	13.15	*timeout*	*> 10min*	200	15.48	*timeout*	*> 10min*	217	19.17	*timeout*	*> 10min*	246	*failed*	*failed*	*failed*

of iterations, the number of vertices in each level of the hierarchy and the sparsity for different multigrid methods in Table 2. Because GravoTet defines prolongation via tetrahedral elements, each fine vertex receives information from no more than four coarse vertices. The other approaches have significantly higher iteration counts, especially for the biharmonic problem, and tend to have a higher maximum number of nonzeroes per row (Table 2).

Mesh quality. Finally, we study the quality of the tetrahedral meshes in the hierarchy. We observe that our coarsening approach slightly decreases the quality of the tetrahedral mesh relative to the input tetGen baseline. Fig. 7 displays the minimum dihedral angle and aspect ratio distribution of all coarsening levels (more examples in the supplemental material). Coarsening occasionally introduces very slim tetrahedra, but we find that this does not result in problems for convergence (see the corresponding convergence graphs in Fig. 5). The coarse complex is used only to define prolongation operators, so slim or low-quality tetrahedra do not compromise solver stability. For thin or concave regions, graph-based geodesic distances during the Voronoi propagation prevent interior shortcuts that would otherwise group geometrically distant boundary samples.

5.4. Ablation study

Our approach introduces two major augmentations to the algorithm adapted from Gravo MG: a boundary-aware coarsening

Table 2: Comparative analysis of matrix sparsity showing the average and maximal number of non-zero entries in the prolongation matrices. Our approach results in lower iteration counts for the Poisson and biharmonic problem while maintaining a high level of sparsity. The graph-Voronoi based approach guarantees a maximum of 4 non-zero entries per row.

Mesh	Method	#Vert N_l	#It Poisson	#It Biharm.	$\frac{\text{nnz}}{\text{row}}$ (avg/max)
Cylinder	Ours	310k; 38k; 3.9k; 428	**6**	**38**	3.50 / 4
	Shi06	310k; 82k; 17k; 3.5k; 738; 173	17	193	3.17 / 10
	AMG-RS	310k; 71k; 13k; 2.1k; 342	87	+346	2.06 / 8
	AMG-SA	310k; 11k; 143	64	+422	4.31 / 10
Spike Ball	Ours	829k; 101k; 10k; 1.2k; 310	**9**	**61**	3.44 / 4
	Shi06	829k; 216k; 45k; 9.2k; 2.0k; 523; 167	17	+407	3.11 / 11
	AMG-RS	829k; 190k; 34k; 5.5k; 968; 205	+285	+447	1.98 / 10
	AMG-SA	829k; 31k; 469	63	+504	4.65 / 12
Lucy	Ours	1.9M; 225k; 23k; 2.5k; 307	**16**	**43**	3.49 / 4
	Shi06	1.9M; 488k; 101k; 20k; 4.2k; 943; 224	20	+228	3.10 / 18
	AMG-RS	1.9M; 425k; 77k; 12k; 2.0k; 428	260	+255	2.02 / 10
	AMG-SA	1.9M; 68k; 820; 18	89	+294	4.18 / 13
Fertility	Ours	2.8M; 333k; 33k; 3.4k; 392	**9**	**36**	3.53 / 4
	Shi06	2.8M; 726k; 145k; 28k; 5.7k; 1.2k; 260	19	+200	3.11 / 17
	AMG-RS	2.8M; 633k; 113k; 18k; 2.8k; 536; 152	+146	+217	1.95 / 9
	AMG-SA	2.8M; 99k; 1.0k; 14	86	+246	4.01 / 12

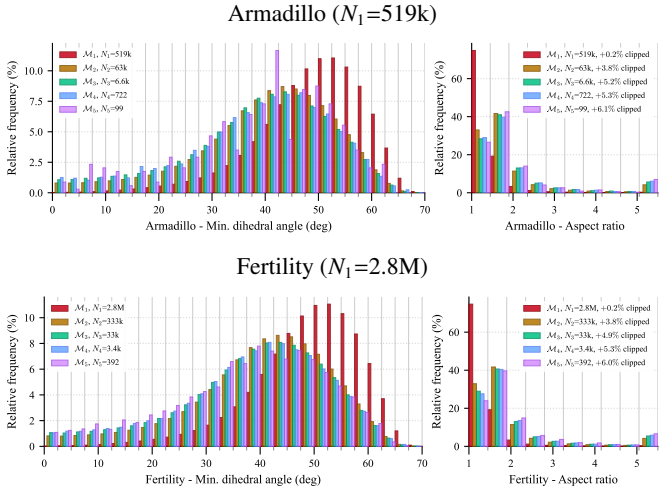

Fig. 7: Representative statistics on the quality of the tetrahedra from two distinct coarsenings.

strategy and the application of Voronoi-based barycentric prolongation weights to tetrahedral volumetric domains. To isolate and analyze the impact of each component, we evaluate our algorithm incrementally. First, we skip the boundary-first graph connectivity as well as the feature prioritization, this is labelled as *bdy.* vs. *no bdy*. Second, we replace our barycentric-based prolongation weight computation, which always involves at most 4 vertices, with the 4-nearest points combined with inverse-distance weights, which we label as *bary.* vs. *inv. dist.*. We observe in Fig. 8 that the boundary-aware approach and the use of barycentric weights in containing tetrahedra both improve the residual convergence rate. We find that the latter is essential for the biharmonic problem. The implementation of these approaches comes with only a small increase in build time. We observed this behavior with all meshes.

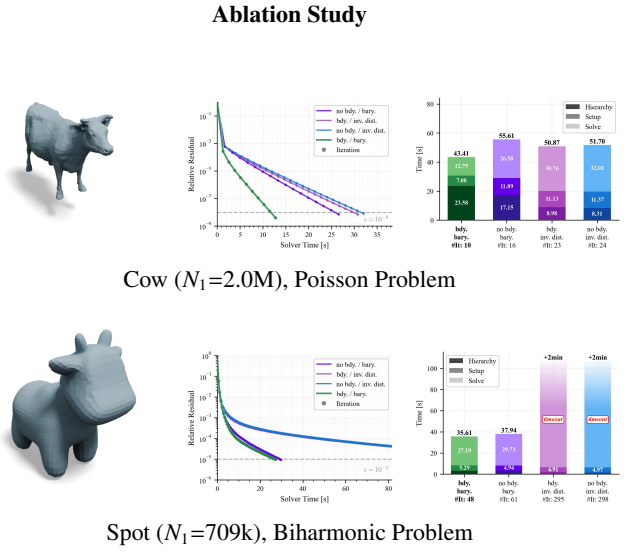

Fig. 8: An ablation variant of our method on the Poisson equation (top, Cow) and the biharmonic equation (bottom, Spot). Left: rendered input mesh. Center: relative residual vs. wall-clock time for the V-cycle iterations. Right: total execution time breakdown.

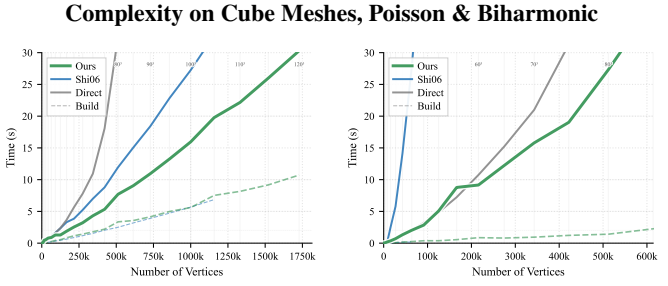

Fig. 9: Scalability benchmarks recorded across increasing resolutions of a regular tetrahedral cube for the Poisson problem (left) and the biharmonic problem (right).

5.5. *Performance*

To analyze scalability, we evaluate the solvers on a regularly tetrahedralized cube at varying resolutions, comparing the total execution times against the best competitors: Shi06 [4] and CHOLMOD [1]. All benchmarks were performed on a machine equipped with an Intel Core i9-14900KF CPU and 32 GB of RAM. In Fig. 9, we see that GravoTet exhibits highly favorable asymptotic scaling with respect to element density.

5.6. *Applications*

GravoTet is suitable for a variety of applications. Figure 10 shows examples of solutions found using our method on volumetric steady-state equations and geodesic distance equations, using the method from [38].

6. Conclusion

In this work, we introduce GravoTet, a geometric multigrid formulation for solving sparse linear systems on tetrahedral

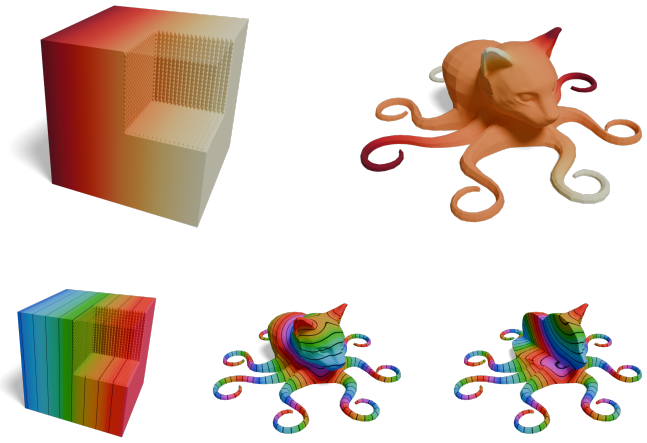

Fig. 10: Top row: steady-state Laplace solution. Bottom row: geodesic distance computed via the heat method.

meshes. At the core of our method is a fast construction of geometric multigrid hierarchies adapted from [5], extended to work on tetrahedral meshes and augmented to preserve boundary. We empirically demonstrate that the resulting hierarchies are higher-quality than those produced by other approaches, yielding robust and efficient solver convergence across a wide variety of applications as exemplified on Poisson and biharmonic problems. Because both the hierarchy construction and solver convergence are fast, our method outperforms other state-of-the-art approaches on the task of solving a single linear system, especially on large meshes (more than 100 k vertices).

Several opportunities remain for further improving our method. Our feature preservation technique does not guarantee the preservation of sharp edges, this could be accomplished by integrating edge-feature information into the graph construction phase. More generally, the current sampling priorities are heuristic. In particular, the dihedral-angle sharpness score can be sensitive to surface noise, and more robust choices based on local averaging or application-specific features are likely beneficial for specific problem classes or boundary geometries. Additionally, we would like to explore methods to improve the quality (Fig. 7) of the coarsened tetrahedral meshes and see potential for further parallelizing the construction. Finally, our approach does not strictly require tetrahedral meshes as input and could work on point clouds with an appropriate neighborhood graph. This could be relevant for solving linear systems on the centroids resulting from 3D Gaussian Splatting [39].

GravoTet bridges a gap between multigrid hierarchies that can be constructed efficiently and those with fast convergence behavior. It enables solving high-resolution volumetric geometry processing tasks that were computationally impractical before.

CRediT authorship contribution statement

Marcel Padilla: Writing – review & editing, Investigation. **Ruben Wiersma:** Writing – review & editing, Visualization. **Tim Huisman:** Writing – review & editing, Investigation. **Jackson Campolattaro:** Writing – review & editing. **Olga Sorkine-Hornung:** Supervision. **Klaus Hildebrandt:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Marcel Padilla was supported by the Alexander von Humboldt Feodor Lynen Fellowship and Jackson Campolattaro by the Dutch Research Council (NWO) under file number OCENW.M.21.346.

Data availability

Some of the basic surface meshes with various attributes used in this work cannot be redistributed.

References

[1] Chen, Y, Davis, TA, Hager, WW, Rajamanickam, S. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. ACM Trans Math Softw 2008;35(3):22:1–22:14. doi:10.1145/1391989.1391995.

[2] Yserentant, H. Old and new convergence proofs for multigrid methods. Acta Numerica 1993;2:285–326. doi:10.1017/S0962492900002385.

[3] Brune, PR, Knepley, MG, Scott, LR. Unstructured geometric multigrid in two and three dimensions on complex and graded meshes. SIAM Journal on Scientific Computing 2013;35(1):A173–A191. doi:10.1137/110827077.

[4] Shi, L, Yu, Y, Bell, N, Feng, WW. A fast multigrid algorithm for mesh deformation. ACM Trans Graph 2006;25(3):1108–1117. doi:10.1145/1141911.1142001.

[5] Wiersma, R, Nasikun, A, Eisemann, E, Hildebrandt, K. A fast geometric multigrid method for curved surfaces. ACM Transactions on Graphics 2023;41(4). doi:10.1145/3588432.3591502.

[6] Ruge, JW, Stüben, K. 4. Algebraic Multigrid. In: Multigrid Methods. Frontiers in Applied Mathematics; Society for Industrial and Applied Mathematics. ISBN 978-1-61197-188-0; 1987, p. 73–130. doi:10.1137/1.9781611971057.ch4.

[7] Vaněk, P, Mandel, J, Brezina, M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing 1996;56(3):179–196. doi:10.1007/BF02238511.

[8] Bell, N, Olson, LN, Schroder, JB. Pyamg: Algebraic multigrid solvers in python. J Open Source Softw 2022;7(72):4142. doi:10.21105/JOSS.04142.

[9] Bank, RE, Dupont, T. An optimal order process for solving finite element equations. Mathematics of Computation 1981;36(153):35–51.

[10] Braess, D, Hackbusch, W. A new convergence proof for the multigrid method including the v-cycle. SIAM Journal on Numerical Analysis 1983;20(5):967–975.

[11] Georgii, J, Westermann, R. A multigrid framework for real-time simulation of deformable bodies. Comput Graph 2006;30(3):408–415. doi:10.1016/J.CAG.2006.02.016.

[12] Otaduy, MA, Germann, D, Redon, S, Gross, M. Adaptive deformations with fast tight bounds. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '07; Goslar, DEU: Eurographics Association. ISBN 9781595936240; 2007, p. 181–190.

[13] Dick, C, Georgii, J, Westermann, R. A hexahedral multigrid approach for simulating cuts in deformable objects. IEEE Transactions on Visualization and Computer Graphics 2011;17(11):1663–1675. doi:10.1109/TVCG.2010.268.

[14] Dick, C, Rogowsky, M, Westermann, R. Solving the fluid pressure poisson equation using multigrid - evaluation and improvements. IEEE Trans Vis Comput Graph 2016;22(11):2480–2492. doi:10.1109/TVCG.2015.2511734.

[15] McAdams, A, Sifakis, E, Teran, J. A parallel multigrid poisson solver for fluids simulation on large grids. In: Popovic, Z, Otaduy, MA, editors. Proceedings of the 2010 Eurographics/ACM SIGGRAPH Symposium on Computer Animation, SCA 2010, Madrid, Spain, 2010. Eurographics Association; 2010, p. 65–73. doi:10.2312/SCA/SCA10/065-073.

[16] Kazhdan, M, Bolitho, M, Hoppe, H. Poisson surface reconstruction. In: Proceedings of the Fourth Eurographics Symposium on Geometry Processing. SGP '06; Goslar, DEU: Eurographics Association. ISBN 3905673363; 2006, p. 61–70.

[17] Kazhdan, M, Hoppe, H. An adaptive multi-grid solver for applications in computer graphics. Computer Graphics Forum 2019;38(1):138–150. doi:https://doi.org/10.1111/cgf.13449.

[18] Ollivier-Gooch, C. Coarsening unstructured meshes by edge contraction. International Journal for Numerical Methods in Engineering 2003;57(3):391–414. doi:https://doi.org/10.1002/nme.682.

[19] Shi, X, Koehl, P. Adaptive skin meshes coarsening for biomolecular simulation. Computer Aided Geometric Design 2011;28(5):307–320. doi:https://doi.org/10.1016/j.cagd.2011.04.001.

[20] Liu, J, Shang, F, Song, T. A new method for coarsening tetrahedral meshes. International Journal for Numerical Methods in Engineering 2017;112(13):2048–2066. doi:https://doi.org/10.1002/nme.5594.

[21] Shi, X, Bao, H, Zhou, K. Out-of-core multigrid solver for streaming meshes. ACM Transactions on Graphics (TOG) 2009;28(5):1–7.

[22] Liu, HTD, Zhang, JE, Ben-Chen, M, Jacobson, A. Surface multigrid via intrinsic prolongation. ACM Trans Graph 2021;40(4). doi:10.1145/3450626.3459768.

[23] Liu, HTD, Gillespie, M, Chislett, B, Sharp, N, Jacobson, A, Crane, K. Surface simplification using intrinsic error metrics. ACM Transactions on Graphics 2023;42(4):1–17. doi:10.1145/3592403.

[24] Brandt, A. Algebraic multigrid theory: The symmetric case. Applied Mathematics and Computation 1986;19(1):23–56. doi:10.1016/0096-3003(86)90095-0.

[25] Stüben, K. A review of algebraic multigrid. Journal of Computational and Applied Mathematics 2001;128(1):281–309. doi:https://doi.org/10.1016/S0377-0427(00)00516-1; numerical Analysis 2000. Vol. VII: Partial Differential Equations.

[26] Hackbusch, W. Multi-grid methods and applications; vol. 4 of *Springer series in computational mathematics*. Springer; 1985. ISBN 978-3-540-12761-1.

[27] Brandt, A. Algebraic multigrid theory: The symmetric case. Appl Math Comput 1986;19(1–4):23–56. doi:10.1016/0096-3003(86)90095-0.

[28] Davis, TA, Rajamanickam, S, Sid-Lakhdar, WM. A survey of direct methods for sparse linear systems. Acta Numer 2016;25:383–566. doi:10.1017/S0962492916000076.

[29] Jacobson, A. Sparse positive definite linear system solver benchmark. https://github.com/alecjacobson/sparse-solver-benchmark; 2021. Accessed: 2026-03-28.

[30] Briggs, WL, Henson, VE, McCormick, SF. A multigrid tutorial, Second Edition. SIAM; 2000. ISBN 978-0-89871-462-3.

[31] Erwig, M. The graph voronoi diagram with applications. Networks 2000;36(3):156–163.

[32] Klingner, BM, Feldman, BE, Chentanez, N, O'Brien, JF. Fluid animation with dynamic meshes. ACM Trans Graph 2006;25(3):820–825. doi:10.1145/1141911.1141961.

[33] Ando, R, Thürey, N, Wojtan, C. Highly adaptive liquid simulations on tetrahedral meshes. ACM Trans Graph 2013;32(4). doi:10.1145/2461912.2461982.

[34] Krishnan, D, Fattal, R, Szeliski, R. Efficient preconditioning of laplacian matrices for computer graphics. ACM Trans Graph 2013;32(4). doi:10.1145/2461912.2461992.

[35] Jacobson, A, Panozzo, D, et al. libigl: A simple C++ geometry processing library. 2018. https://libigl.github.io.

[36] Wardetzky, M, Bergou, M, Harmon, D, Zorin, D, Grinspun, E. Discrete quadratic curvature energies. Computer Aided Geometric Design 2007;24(8-9):499–518.

[37] Si, H. Tetgen, a delaunay-based quality tetrahedral mesh generator. ACM Trans Math Softw 2015;41(2):11:1–11:36. doi:10.1145/2629697.

[38] Crane, K, Weischedel, C, Wardetzky, M. The heat method for distance computation. Commun ACM 2017;60(11):90–99. doi:10.1145/3131280.

[39] Zhou, H, Lähner, Z. Laplace-Beltrami operator for Gaussian splatting. In: International Conference on 3D Vision (3DV). 2026, p. 1–10. URL: https://arxiv.org/abs/2502.17531. arXiv:2502.17531.