DOI: 10.1111/cgf.70404
EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Hierarchical Optimization of the As-Rigid-As-Possible Energy

Hendrik Meyer[1] Bernd Bickel[2,3] Marc Alexa[1]

[1]TU Berlin, Computer Graphics Group, Germany [2]ETH Zurich, Computational Design Lab, Switzerland [3]Google, Switzerland

Abstract

The As-Rigid-As-Possible (ARAP) energy [SA07] has become a versatile ingredient in various geometry processing and machine learning methods. The classic method for its minimization is a block coordinate descent, alternating between local rotation estimation and a global linear solve, which converges slowly for large problem instances. We develop and evaluate a multi-level scheme targeted specifically at the optimization of the ARAP energy on large meshes. The main points of our approach are (1) a mesh hierarchy that provides the necessary control over topology while being fast, (2) methods for upsampling the rotations from coarser to finer levels of the hierarchy, and (3) using direct solvers for the linear system. The resulting optimization, remarkably, yields smaller energy while typically being faster on a large number of test cases. The hierarchical approach generalizes to related energies and compares favorably to acceleration schemes such as ADMM, which, in turn, also profit from the hierarchical approach.

1. Introduction

As-rigid-as-possible (ARAP) surface modeling [SA07] has become an important tool for interactive mesh deformation. The deformation results from minimizing an energy – often referred to as *ARAP* energy – are subject to constraining a subset of the vertices of the mesh (see Section 2 for details). This energy is now used also in the context of deformable matching, i.e. as a regularizer in many geometry processing pipelines, in machine learning, and computational models for physics in games and animation.

Minimizing the energy requires simultaneously optimizing local rotations and vertex positions. This is commonly done using *block-coordinate descent*: In a local step, the vertices are fixed, and rotations are computed for each vertex. In the global step, all vertices are updated for fixed rotations by solving a linear system of equations. While this *local-global* scheme is guaranteed to never increase the energy, it is known to converge slowly, particularly for large deformations and many unknowns (i.e. large meshes). This has always been accepted, noting that convergence *could probably* be expedited using hierarchical or multigrid approaches. While there are several related works in this direction (Section 3), a concrete implementation showing consistently faster optimization or better energies is still lacking to our knowledge.

For a hierarchical/multi-level optimization, we need three ingredients:

1. A hierarchical mesh representation. For large meshes, storage requirements for the representation should be small. Also, in some applications one wants to minimize the energy only once, and the time required for constructing the hierarchy becomes

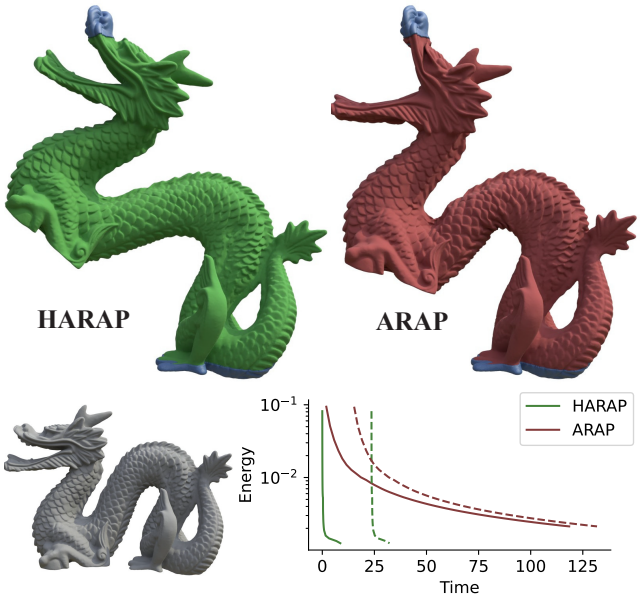


Figure 1: *Solving ARAP in a hierarchical fashion (green) typically leads to solutions with smaller energy than classic ARAP (red), often visually apparent as a smoother resulting shape. The solution is also more efficient: the graph shows ARAP energy over time without (solid) and with (dashed) necessary precomputation (mostly factoring the system matrices).*

part of the cost for optimization. This calls for very efficient simplification methods.
2. Optimizing the energy on each level of the hierarchy, exploiting the existing solution from the coarser level(s).
3. Initializing the next finer level based on the current solution.

The last two steps are intertwined and in the context of ARAP, this may involve both the local rotations and the vertex positions.

Our contributions are concrete solutions for all three steps. We use simplification based on an octree (similar to the idea of z-order curves or Morton codes) to generate the hierarchy, taking care not to connect elements of the mesh that are intrinsically far from each other. Despite the overhead for this topology control, the method remains very fast – details are presented in Section 4.

We exploit the fact that the local-global optimization exposes the local rotation as a variable and use this for our up-sampling strategy. Directly passing rotations between layers allows reconstructing the vertex positions on each level. This approach favors a direct solver for the linear system, which we found beneficial compared to iterative solvers, despite the cost of factoring the system matrix on each level. The specific upsampling strategy is described in Section 5.

Together, the resulting hierarchical optimization approach typically leads to solutions with lower energy, visually appearing smoother and closer to desired deformations, and it is faster. We demonstrate this quantitatively on the subset of meshes in Thingi10K [ZJ16] with more than 100K vertices. For several well-known meshes, we analyze the behavior in detail, including the effect of topology preservation in the simplification and the number of levels of the hierarchy. All our results show that using the hierarchy is generally beneficial compared to the standard local-global approach (see Section 7 for details), also compared to recent more advanced optimization approaches.

Lastly, using the standard ARAP minimization on each level makes the approach easy to adapt to related energies. We discuss these possibilities, including a demonstration for a very recent modification in Section 8.

2. Background: As-rigid-as-possible Shape Modeling

Given a mesh with n vertices, we denote the vertex positions in the "rest" state as $\bar{\mathbf{v}}_i \in \mathbb{R}^3, i \in \{0,\dots,n-1\}$. The goal is to find new vertex positions $\mathbf{v}_i \in \mathbb{R}^3$ to satisfy vertex constraints of the form $\mathbf{v}_j = \mathbf{v}_j^*, j \in \mathcal{C}$, with $\mathcal{C}$ the subset of constrained vertices. The positions of the unconstrained vertices are determined by minimizing the ARAP energy.

The idea of ARAP is to penalize local deviation from rigid transformations. For surface meshes, locality is usually identified with the star of a vertex. While the original version suggested to consider the edges incident on a vertex [SA07], it has later been shown that it may be beneficial to consider the faces, leading to an expression that involves all edges of the incident faces [CPSS10]. Let $\mathcal{F}$ be the set of index triples identifying the triangles of the mesh, the ARAP energy can be expressed as

$$E_{\text{ARAP}} = \sum_{i=0}^{n-1} \sum_{i \in f \in \mathcal{F}} \sum_{j,k \in f} w_{f_{jk}} \left\| (\mathbf{v}_j - \mathbf{v}_k) - \mathbf{R}_i(\bar{\mathbf{v}}_j - \bar{\mathbf{v}}_k) \right\|_2^2, \quad (1)$$

where $w_{f_{jk}}$ is the *cotan* weight, i.e. the cotangent of the angle opposite of the edge jk in triangle f [PP93, CPSS10].

A central point in this formulation is that not only the vertex positions are variables, but also the per-vertex rotations $\mathbf{R}_i$. While these additional variables may appear to be a disadvantage, they lead to a very natural optimization scheme: (1) For fixed vertex positions compute energy minimizing local rotations; (2) for fixed rotations compute optimal vertex positions. This is repeated until convergence.

There are several benefits to the local-global approach: The non-linear part becomes a local problem of best fitting a rotation to the changing vertex stars. This can be conveniently solved using SVD of a 3×3 matrix [SHR17], or an even faster Newton-like approach [ZJA21] exploiting coherence between successive optimization steps. The global problem is a standard linear least squares optimization, so the solution can be computed by solving a positive semi-definite linear system. Moreover, by applying the per-vertex rotations to the rest state vertices, the change in the linear system of equations due to changing rotations appears on the right hand side and the system matrix remains constant. This allows factoring the *symmetric* system matrix once, and solving against varying vectors by forward- and backward-substitution. In addition, the non-zero structure of the system matrix for triangle meshes aligns well with sparsity-preserving Cholesky factorization [BBK05], meaning the number of additional non-zero elements in the Cholesky factors is small.

For large deformations, alas, this optimization has been observed to progress slowly, spending most of the effort on moving through regions of the energy landscape that are far from the eventual solution. Next, we discuss several options for tackling this problem.

3. Related Work

There are numerous variations on the ARAP energy, on the one hand, and several general approaches for optimizing energies defined over the positions of a triangle mesh, not specifically focused on ARAP, on the other hand. We focus here on the goal of computing better global minimizers of the ARAP energy in shorter time. We are not claiming that deformations computed by minimizing ARAP are necessarily superior to alternative methods, such as approximate ARAP deformations, and leave a comparison of the resulting deformations of fundamentally *different* methods for possible future work.

3.1. Approximations to ARAP

The ARAP energy is defined over the positions of the vertex set, resulting in a large number variables for meshes with many vertices. Several methods for the approximation of the ARAP energy are based on reducing or limiting the degrees of freedom of the possible deformations.

Early methods introduce an embedding space that used to deform surface meshes, by choosing a low number of handles on the mesh that drive the full deformation. While some methods try to incorporate the original structure into the energy by keeping nearby deformations similar [SSP07], other methods try to incorporate the

surface geometry into the energy formulation [HSL*06, SZT*07] itself, using Laplacian coordinates [SCOL*04].

Based on these methods, more recent techniques attempt to minimize the ARAP energy on a reduced set of degrees of freedoms. While ARAP is defined as a surface deformation energy, some methods were proposed based on handles defined in ambient space. These methods include space deformation, e.g., lattice deformation [ZSGS12], or skinning methods [JBK*12]. Alternatively, the degrees of freedom can be reduced by applying a mesh simplification algorithm [GH97]. The simplification is typically iterated until the number of vertices is reduced below a desired threshold. Applying the ARAP energy to these meshes is computationally efficient. The solution is then up-sampled to the original mesh resolution [MS11] by undoing the simplification steps while keeping the constraints satisfied. Some of these methods also allow the finer mesh to be updated without fully recomputing the simplification [DGG14]. Linear subspace design [WJBK15] allows representing the mesh by a subset of the original vertices. The full resolution mesh can be reconstructed from this subset. Only the subset's vertices need to be optimized, reducing the degrees of freedom to the size of the subset, although it takes the original geometry into account.

Although all of these methods can produce results similar to ARAP in a fraction of the time, they only approximate the original energy. In general, reducing the degrees of freedom will prevent small scale deformations, which may or may not be desirable in practical applications.

3.2. Multi-resolution mesh modeling

A hierarchical representation of a surface mesh is useful in many contexts. The most prominent and perhaps earliest example are *progressive meshes* [Hop96], generating a sequence of finer meshes by edge collapses. It has later been observed that for many tasks it is more convenient to create a small set of global levels of details [EDD*95, GSS99], similar to other multi-resolution representations in signal processing, such as wavelets.

The general strategy for generating multiple levels of detail is to perform repeated mesh simplification. The two main methods for simplification are local removal operations, performed concurrently on a set of vertices that can be treated independently [EDD*95, GSS99]; or vertex clustering [RB93], performed on some subdivision of either the ambient space or the surface itself [BA09]. A particular simple and efficient approach is vertex clustering on an octree [Lin03, SW03, SG05, LJBA13], which may also come in the guise of Morton codes (or z-order curves) [LB15]. This is the basis for our approach (see Section 4).

Multi-resolution representations are also common for mesh deformation. The main issue is that linear techniques may cause unnatural behavior in the local details of the geometry. This can be avoided by applying the linear mesh deformation to a coarse level representation and then adding the detail of the finer levels relative to the deformed coarse mesh [ZSS97, KCVS98, SYBF06, BK04]. This is also useful for non-linear mesh deformation [BPGK06], where the solution from coarse levels is up-sampled and then locally refined using non-linear optimization.

Multi-resolution algorithms have also been used for position-based dynamics [MAK24, Mü08] and simulation of deformable objects [XTL19]. For the simulation the state of the object is expressed as skinning transformations [JBK*12] on the coarse levels instead of as vertex positions. We follow a similar idea by using rotations to express the solution on any level. Furthermore, the simulation requires dynamic modification of the system matrix, favoring a classic multi-grid solver, which is not necessarily the case for ARAP.

A variety of problems can be tackled with such a *cascading multi-grid* technique, common in numerical methods on regular grids [BHM00]. Multi-grid techniques are often advantageous compared to global methods (direct solvers for linear systems or iterative methods such as gradient descent or Newton) when the solution is far from the initial guess; and they offer an inherent space/time trade-off. Apart from the specialized instances mentioned above, they have also been explored as a general purpose tool for meshes [LZBCJ21, WNEH23].

We are aware of only one instance of a multi-grid method applied specifically to the problem of minimizing the ARAP energy [BCDD22]. The construction for the hierarchy uses QSlim, which is often too slow for very large meshes in practice. For the solution of the linear system on each of the hierarchy levels, they follow common practice and use an iterative approach, namely conjugate gradients. One of our central observations is that factoring the system on each level pays off and leads to better convergence.

3.3. Efficient optimization of non-linear deformation energies

The local-global minimization strategy used in As-Rigid-As-Possible (ARAP) deformation was initially seen as an efficient optimization scheme for non-linear deformation energies. However, it was demonstrated that it is essentially a form of Sobolev gradient descent [KGL16]. This insight led to the development of a number of first-order optimization techniques for deformation energy minimization [KGL16, RPPSH17, SS15], acting on the vertex positions directly without intermediate rotation variables.

To improve convergence properties near energy minima composite majorization [SPSH*17] was introduced as second order method, This advancement enhanced convergence rates and stability. Building upon this, a hybrid approach was proposed that blends first-order and second-order methods, achieving even faster convergence in practice [ZBK18].

A more general framework for simulating deformation energies was introduced with Projective Dynamics [BDS*12, BML*14, Wan15]. While this framework efficiently supports a combination of multiple energy terms, it is limited linearizable energies. Subsequent work extended the approach by interpreting Projective Dynamics as a quasi-Newton method [LBK17], or generalized it using the Alternating Direction Method of Multipliers (ADMM) to support non-linear energies [OBLN17].

Further improvements were achieved through Anderson Acceleration [And65, WN11], which was applied to both Projective Dynamics [PDZ*18] and ADMM-based solvers [ZPOD19].

More recent work utilizes the underlying structure of many energies, namely the presence of a rotational component. This has

enabled decoupling strategies that isolate the rotational part of the energies [BN21, SLS22].

4. Mesh Hierarchy

The first step for a hierarchical minimization approach is creating a hierarchy of successively coarser approximations of the mesh. Let the original mesh be represented by n vertices $\bar{\mathbf{v}}_i \in \mathcal{V}$ in rest state, and triangles represented as a set of index triples $\mathcal{T} = \mathcal{T}_0$. Our basic idea is to generate a nested sequence of subsets of triangles $\mathcal{T}_0 \supset \mathcal{T}_1 \supset \mathcal{T}_2 \supset \dots$ as well as updated vertex positions $\mathcal{V}_i$ so that the subsets of triangles still form a connected triangle mesh (see Figure 2).

We create this hierarchy conceptually by clustering on nested regular grids. This choice is motivated by several reasons:

- Clustering on nested regular grids is extremely fast, with little additional cost to the overall optimization procedure.
- Coarse levels have a spatially uniform vertex distribution, so that large deformations in space can be propagated quickly.
- Vertex clustering is insensitive to defects in the mesh, i.e. it works also for meshes that are not manifold, not watertight, or self-intersecting.
- It is unlikely that the preservation of features that may be relevant visually is necessary for the multi-grid optimization.

To aid efficient computation of the clustering, we use a simple and well-known trick [LJBA13]: if the coordinates are stored as integers, we can get their positions on successively coarser grids (i.e. an octreee) simply by removing the least significant digits from the binary representation. We start with a q bit integer representation, computed as

$$\mathbf{w}_{i,d} = \left\lfloor (2^{\mathbf{q}_d} - 1) \frac{\bar{\mathbf{v}}_{i,d} - \mathbf{b}_{\min,d}}{\mathbf{b}_{\max,d} - \mathbf{b}_{\min,d}} + \frac{1}{2} \right\rfloor \in \mathbb{Z}^3, \tag{2}$$

where $d \in \{0,1,2\}$ specifies the x,y and z-components and $\mathbf{b}_{\min}, \mathbf{b}_{\max} \in \mathbb{R}^3$ represent the bounding box of the mesh. We choose $\mathbf{q}$, such that the average edge length $\bar{E}$ is smaller or equal to the cell length for component d,

$$\mathbf{q}_d = \left\lfloor \log_2 \left(\frac{\mathbf{b}_{\max,d} - \mathbf{b}_{\min,d}}{\bar{E}} \right) \right\rfloor. \tag{3}$$

Vertex hierarchy The integer coordinates provide an embedding into a regular grid with $\prod_d (2^{\mathbf{q}_d} - 1)$ cells. Vertex positions within the same cell of this grid are mapped to the same position. Rather than repeating the process for smaller values of q to get coarser grids, we generate vertex positions on coarser grids by only considering the first $q - l$ bits in the integer representation, with $l \in \{0, \dots, q-1\}$ identifying the *level* in the hierarchy. This defines the vertex sets $\mathcal{V}_l$. Depending on the level l, different vertices collapse to a single one. This collapsing of vertices defines an equivalence relation. Concretely, vertices are equivalent on level l, if the first $q - l$ bits in the representation $\mathbf{w}_i$ are the same:

$$i \sim_l j : \lfloor 2^{-l} \mathbf{w}_i \rfloor = \lfloor 2^{-l} \mathbf{w}_j \rfloor. \tag{4}$$

Note that the equivalence relation defines a nested regular grid, i.e. one can intuitively think of this as clustering on an octree. Equivalently, the equivalence classes are nested.

For each level l of the hierarchy, the set of different vertex positions in $\mathcal{V}_l$ becomes smaller. We never explicitly need to store the vertex position, as they can be generated from the integer representation on the fly. All we need for the later step of generating the nested sets of triangles is the equivalence relation.

Triangle hierarchy The process of collapsing vertices into the same position causes triangles to degenerate. This directly defines the nested sequence of triangles: a triangle is part of $\mathcal{T}_l$ iff it is non-degenerate for $\mathcal{V}_l$. Notice that triangles that are degenerate on l cannot become non-degenerate on a level $l' > l$. Or conversely, a non-degenerate triangle on level l is non-degenrate on level $l' < l$. Concretely, we define the triangle sets based on the equivalence classes as

$$\mathcal{T}_{l+1} = \{(i,j,k) \in \mathcal{T}_l : i \not\sim_l j \not\sim_l k\}. \tag{5}$$

For computation, we start at level $l = 0$ and identify triangles that have at least two vertices that are equivalent and remove them from the initial set $\mathcal{T} = \mathcal{T}_0$. This defines $\mathcal{T}_1$. We continue with $l = 1$, defining $\mathcal{T}_2$, and so on until a minimum of vertices remains (see Figure 3).

Topology Throughout the hierarchy, the topology of the mesh will change – and this is generally desirable, as preserving the original topology would severely limit simplification. While many such topology changes are not hurting the ARAP minimization, one instance deserves more attention: ARAP is based on an intrinsic energy (up to a bending term), meaning the dependence among vertices is determined by the length of the paths in the (weighted) vertex-edge graph. In the construction of the hierarchy, however, we identify vertices based on distance in ambient space. This may cause vertices to be identified that are far from each other in the vertex-edge graph, for example when different parts of a shape are close to each other or are self-intersecting. We prevent the merging of different parts of the surface by revisiting the definition of an equivalence class: Equivalence should not only reflect proximity in ambient space, but also connectivity among the vertices *within* one cell. We continue with the details of this process and defer a comparison based on real results to Section 7.

Previously, two vertices $i, j \in \mathcal{V}_l$ were collapsed if they were part of the same equivalence class, $i \sim_l j$. We extend this idea by saying that i and j are collapsed iff there is a path $\mathcal{P} = (i, \dots, k, \dots, j)$ on the edges of $\mathcal{T}_l$ for which $i \sim_l k \sim_l j \; \forall \, k \in \mathcal{P}$. This prevents cases, as shown in Figure 3, where vertices share an equivalence class yet are not intrinsically close.

We implement this identification by a breadth first search within $\mathcal{V}_l$ (Algorithm 1). Notice that in practice there are only very few edges that need to be traversed for each equivalence class. Once a path is found between two vertices they are marked as connected. Every breadth first search reveals one equivalence class and is started with an arbitrary vertex not yet assigned. Each vertex has to be visited only once because the edges are undirected.

Note that a vertex in the hierarchy may well be unused, because none of the triangles connected to it is non-degenerate. These vertices are removed in the current level. That being said, for the initialization of the next finer level from the optimization on a coarse

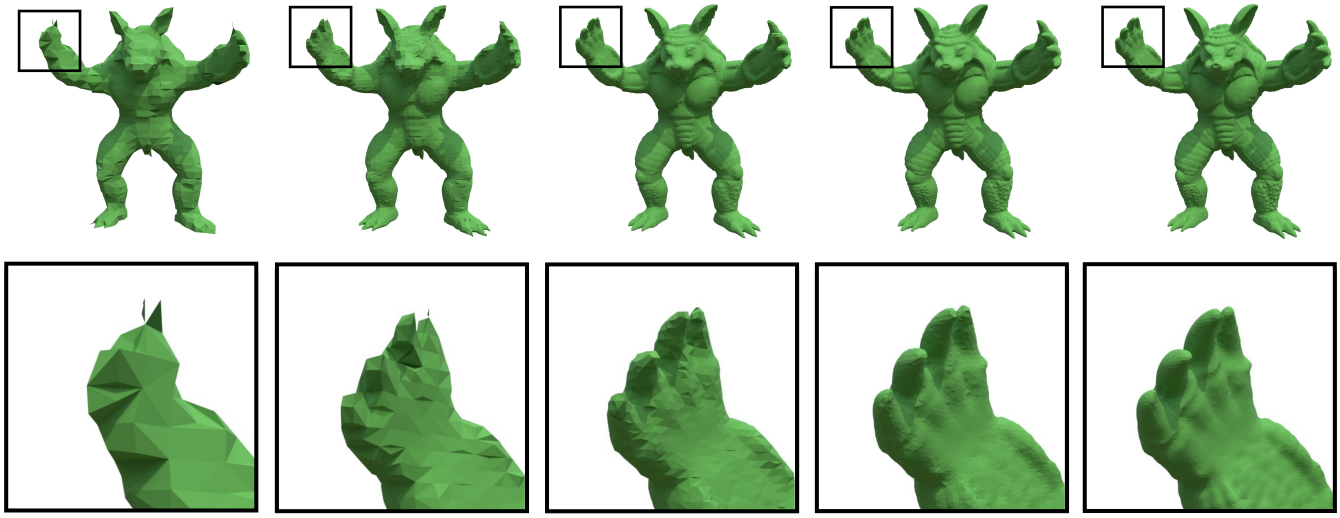


Figure 2: *Hierarchy levels of the armadillo mesh. The vertex clustering approach does not maintain manifoldness and may create flat surfaces.*

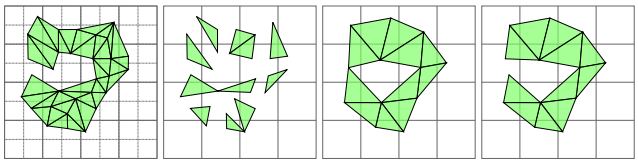


Figure 3: *We start with the triangles given at level l (left). We remove all triangles that will degenerate due to two vertices within the same cell, i.e. $\sim_{l+1}$ (center left). The vertices are joined, reconnecting the triangles (center-right). Considering the topology prevents previously disconnected vertices from being joined (right).*

Algorithm 1: Topology check

```
foreach i ∈ V_l do
    visited[i] ← false
foreach i ∈ V_l do
    if visited[i] then continue;
    initialize queue Q; enqueue i into Q
    while Q is not empty do
        j ← dequeue Q
        visited[j] ← true; class[j] ← i
        foreach k ∈ Adjacent(j) do
            if i ∼_l j then
                enqueue k into Q
```

level we have found it beneficial to gather values by identifying the closest optimized vertex in the vertex-edge graph, again by breadth first search. This information is computed once during the construction of the hierarchy, to be used for the up-sampling process, described in the next section.

Constraints Part of the input for ARAP is the set of constrained vertex positions. Since in the hierarchical approach, an ARAP min-

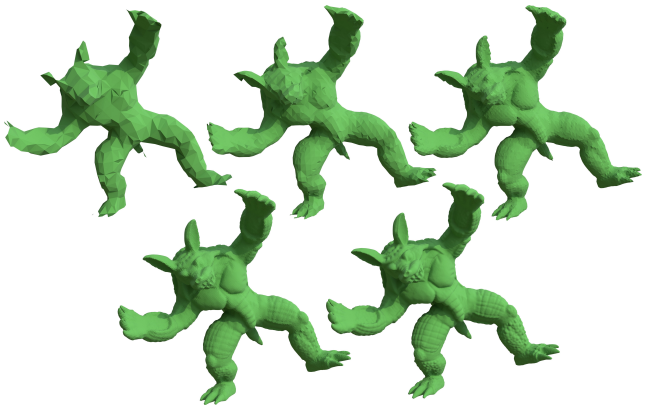


Figure 4: *Deformed state of the armadillo after optimization on each level.*

imization is performed on each level, we also need constraints on the coarser levels. This is done as follows: If any vertex within an equivalence class is constrained, the whole class becomes constrained.

Yet, for the position, we deviate from the simple assignment to the truncated integer representation. Rather, we displace the position based on the relative position of the constrained vertex and the position of the constrained class. Instead of directly assigning this position to the vertex class as a constraint, we store it as an offset from the vertex class position. This allows adjusting for the changing rotation throughout the optimization, see next section. If multiple vertices within one cell are constrained, we take the average of their target positions for the current level.

5. Optimization

For minimization of the ARAP energy we follow the idea of a cascadic multi-grid solve on the hierarchy introduced in the previous section. The strategy is to optimize concrete positions for the vertices in $\mathcal{V}_l$ on the subset of triangles $\mathcal{T}_l$, starting from the coarsest level, propagating the solutions as the initialization for the next finer level (see Figure 4). In addition to vertex positions, the optimization also generates sets of local rotations associated with the vertices, which we denote $\mathcal{R}_l$. We describe the necessary up-sampling procedure for using the information generated at the coarser level as an initial guess for the finer level below. Given the starting configuration, the optimization on each level follows the basic ARAP minimization approach: (1) update vertex positions by solving a linear system; (2) update local rotations. We discuss possible choices for the solution of the linear system in the global step in the supplementary material.

A full overview of the optimization problem can be seen in Algorithm 2.

5.1. Up-sampling

On each level, we generate two sets of variables: the vertex positions of the mesh and the rotations associated with each vertex. We notice that in the context of the iterative optimization of the energy on each level, it suffices to up-sample *only one* of the two sets, as rotations imply vertex position in the global solve and, conversely, vertex positions imply rotations in the local solve.

Perhaps the more common approach in the context of multi-grid solvers would be to up-sample the vertex positions. This can be done by exploiting the relation of the vertex $\bar{\mathbf{v}}_i^l$ to all vertices $\bar{\mathbf{v}}_j^{l-1}$ on the next level that are identified with it, i.e. $i \sim_l j$. We can then adjust the relative position based on the rotation $\mathbf{R}_i^l$ associated to $\mathbf{v}_i^l$ as follows:

$$\mathbf{v}_j^{l-1} = \mathbf{v}_i^l + \mathbf{R}_i^l \left(\bar{\mathbf{v}}_j^{l-1} - \bar{\mathbf{v}}_i^l \right), \quad i \sim_l j. \tag{6}$$

Intuitively, the vertices in one cell on level l are transformed together rigidly by the rotation computed for the representative vertex. This means the different vertex sets move independently, potentially causing discontinuities in the up-sampled mesh depending on the similarity of the rotations (compare Figure 5).

Algorithm 2: Hierarchical ARAP Optimization

```
Input: Template mesh T_0, vertex set V_0, constraints C
Output: Deformed mesh at finest level
Compute mesh hierarchy T_0 ⊃ T_1 ⊃ ··· ⊃ T_L with
  corresponding vertex sets V_0, V_1, ..., V_L (Section 4)
for l ← 0 to L do
  | Precompute ARAP operators and data for level T_l, V_l
Initialize deformation at coarsest level l ← L
for l ← L to 0 do
  | repeat
  |   | Local step: estimate per-element rotations
  |   | Global step: solve linear ARAP system
  |   |   (Section 5.2)
  | until convergence;
  | if l > 0 then
  |   | Upsample deformation to level l − 1 (Section 5.1)
return Deformation at level T_0
```

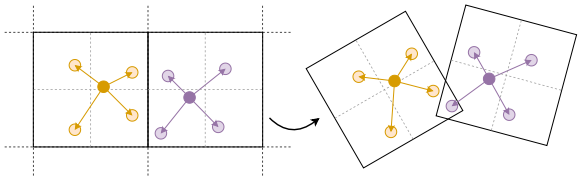


Figure 5: *Two neighbouring cells are transformed rigidly during up-sampling. The transformation of the cells can introduce discontinuities.*

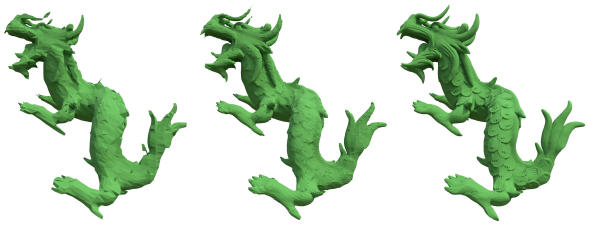


Figure 6: *Upsampling the rotations allows reconstructing the current solution on all levels. The deformed mesh (left) can be upsampled to the next level (middle) and also to the highest resolution directly (right).*

While up-sampling the vertex positions may be necessary if one wants to use an iterative solver for the global setup (see below), we rather suggest to up-sample the rotations. Quite simply, we set the rotations

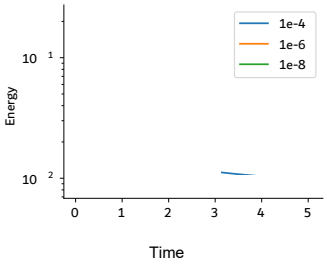
$$\mathbf{R}_j^{l-1} = \mathbf{R}_i^l, \quad i \sim_l j \tag{7}$$

propagating the rotation computed for one vertex to up to 8 vertices on the next finer level. This requires that the next step on the finer level is a global solve for vertex positions. For building the system matrix, we do need vertex positions, yet we can simply take the ones implicitly constructed for the hierarchy.

The benefit of up-sampling rotations compared to vertex positions is that a single global solve immediately establishes a smooth solution (see Figure 6), because rotations are only matched in a least squares sense. In contrast, the first set of rotations computed from the up-sampled vertex positions will rather capture the introduced discontinuities than being close to the desired solution.

The smoothness of the up-sampling process is useful not only for optimization but also for interactive visualization. In fact, one could up-sample the rotations through all levels to the finest solution. This may be seen as an alternative approach to approximate solutions on a reduced set of variables, as discussed in Section 3.1.

A crucial parameter for the multi-grid approach is the convergence criterion used on each level. Aborting the optimization too early may lead to higher costs on the finer levels, where



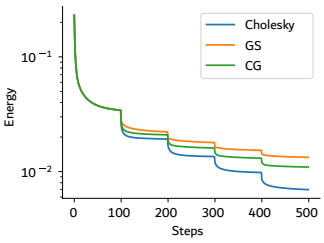
optimization is more expensive. On the other hand, a very accurate solution is not necessary, as it is used only as an initial guess on the finer levels (compare inset). We use the squared max-norm on the displacements resulting from the global solve. Using the squared max-norms makes the condition independent of the vertex count and can be used consistently across all hierarchy levels. This is not a strict convergence criterion but we found it to be a simple yet effective heuristic.

5.2. Global solve

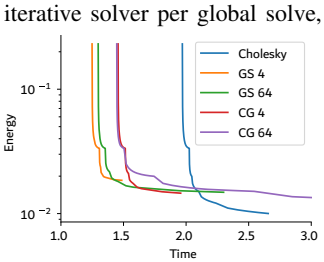
Given the initial state for either the vertices or the rotations, we want to minimize the energy by alternating local and global steps. We omit a discussion of the local rotation estimation – here we follow standard practice. For solving the linear system, we see the following options:

- Standard practice in the context of ARAP is a direct solver. The system matrix is constant throughout the iterations (for the constant mesh on each level) and can be factored using Cholesky, so that each global solve only amounts to forward and back substitution. In addition, using a direct solver requires no initial guess for the vertices, and it is sufficient to up-sample only the rotations.
- On the other hand, in the context of multi-grid solvers, it is standard to use iterative solvers such as Gauss-Seidel (GS) or Conjugate Gradients (CG). This avoids the high setup cost of direct solvers and may be sufficient because one can start from a good initial guess generated on the coarser levels. Incidentally, all related work we are aware of follows this route.

We analyzed the situation in a range of experiments and found that iterative solvers are not well-suited for minimizing the ARAP energy – despite good initial configurations.



The GS steps usually converge to a higher energy on each level than a direct solve, as can be seen in the inset. Although the solution is already close to the minimum, GS converges slowly. Either many iterations are necessary to reach optimal vertex positions, or one continues with an approximate solution, recomputing the rotations. In our experiments, neither of the two options performed convincingly, with GS iterations consistently underestimating the step necessary to reach the best solution on each level. This diminishes the computational advantage of GS compared to Cholesky, especially as the main cost of Cholesky lies in the factorization and may be amortized over computing more than one deformation.



This effect can also be seen by measuring the required runtime. Using only few iterations of an iterative solver per global solve, the optimization converges earlier than using the Cholesky solver but the energy is higher. Increasing the number of iterations improves the final energy, yet it also increases the runtime.

We conclude that the Cholesky solver is to be preferred, at least as long as memory permits. An additional advantage of the multi-grid setting is that one may switch to iterative solvers on the way to the finer levels if factoring becomes impractical for very large meshes. In this way, the advantage of direct solvers and quick visualization can be used while still being able to compute a converged solution on the finest level.

6. Extension to per simplex ARAP deformations

Previously we only considered surface meshes in 3D where the rigid cells are defined as the overlapping one ring neighborhoods per vertex. For most of the approaches mentioned in Section 3.3 a simplicial mesh of the embedding space is assumed, i.e., triangles in 2D and tetrahedra in 3D. In these cases the ARAP energy is typically defined per simplex [CPSS10, LZX*08], requiring additional work to be compatible with our approach. In the following we describe how our approach extends to these meshes.

ARAP The ARAP definition varies from Equation 1 by considering a rotation per simplex [CPSS10, LZX*08]. For a set of triangles/tetrahedra $\mathcal{T}$ it is defined as

$$E_{\text{ARAP}} = \frac{1}{2} \sum_{t \in \mathcal{T}} \sum_{j,k \in t} w_{t_{jk}} \left\| (\mathbf{v}_j - \mathbf{v}_k) - \mathbf{R}_t (\bar{\mathbf{v}}_j - \bar{\mathbf{v}}_k) \right\|_2^2 \tag{8}$$

for triangles in 2D, where $w_{t_{jk}}$ is the *cotan* weight, i.e. the cotangent of the angle opposite of the edge jk in triangle t, and as

$$E_{\text{ARAP}} = \frac{1}{6} \sum_{t \in \mathcal{T}} \sum_{j,k \in t} w_{t_{jk}} \left\| (\mathbf{v}_j - \mathbf{v}_k) - \mathbf{R}_t (\bar{\mathbf{v}}_j - \bar{\mathbf{v}}_k) \right\|_2^2 \tag{9}$$

for tetrahedra in 3D, where $w_{t_{jk}}$ is the *cotan* weight, i.e. the cotangent of the dihedral angle opposite of the edge jk in tetrahedron t, scaled by its length. Although this slightly changes the energy, the classic optimization scheme remains consistent with the surface case. The following descriptions apply to both triangle and tetrahedral meshes.

Hierarchy The hierarchy computations for triangle meshes described in Section 4 works in any dimension. It extends to to tetrahedral meshes in a straightforward way. Instead of considering sets of triplets, the tetrahedral elements are represented as a set of quadruplets. The only required change is modifying Equation 5 to consider quadruplets

$$\mathcal{T}_{l+1} = \{(a,b,c,d) \in \mathcal{T}_l : a \not\sim_l b \not\sim_l c \not\sim_l d\}. \tag{10}$$

All other parts of the construction remain unchanged.

Upsampling This method of up-sampling is only indirectly applicable to tetrahedral meshes, since the rotations are not defined per vertex, but per simplex. This can be resolved by using the average of all incident simplex rotations $\mathbf{R}_t^l$ as a vertex rotation, i.e.

$$\tilde{\mathbf{R}}_v^l = \frac{1}{|\mathcal{N}_v|} \sum_{t \in \mathcal{N}_v} \mathbf{R}_t^l, \tag{11}$$

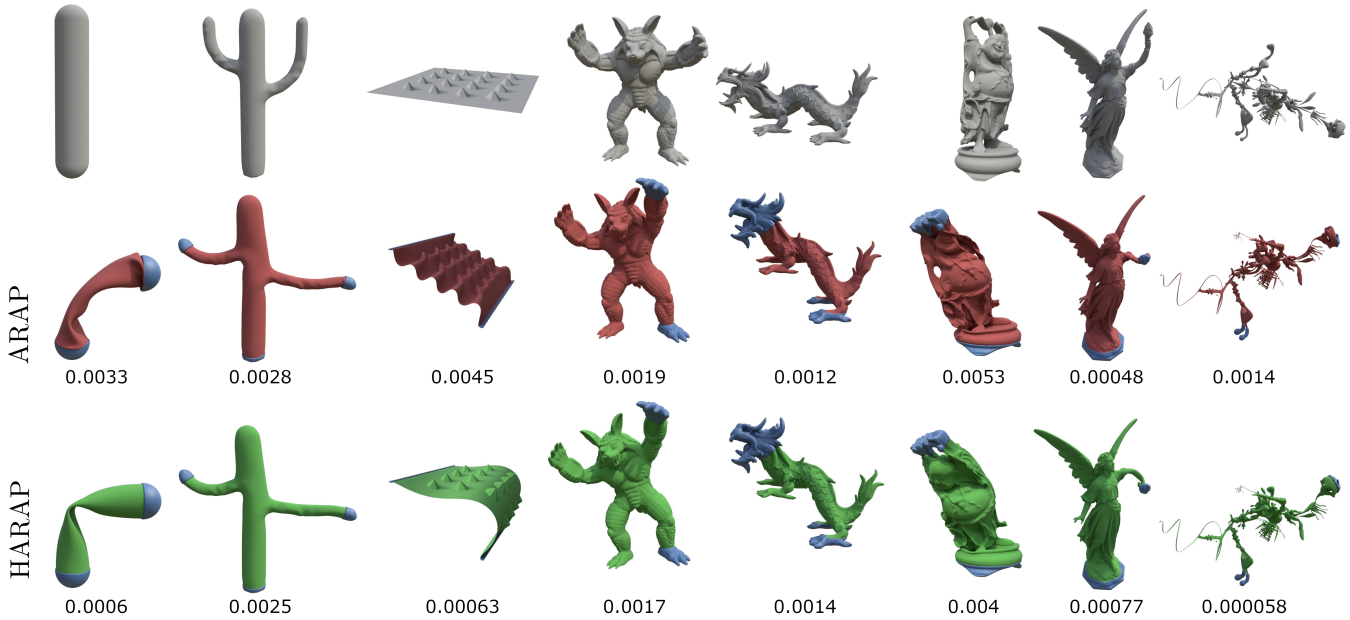


Figure 7: *From top to bottom: Input, ARAP [SA07] (red) and hierarchical ARAP (green). Constraints are sets of blue points. Energy values are shown beneath each mesh (same as Table 1).*

where $\mathcal{N}_v$ is the set of incident simplices to vertex v. The vertex rotations can now be used to either upsample the vertex positions, Equation 6, or by assigning them as new rotations to the finer level, Equation 7. Given the vertex rotations on the finer level, we again need to distribute them to the incident simplices by averaging,

$$\tilde{\mathbf{R}}_t^{l-1} = \frac{1}{4} \sum_{v \in t} \tilde{\mathbf{R}}_v^{l-1}. \tag{12}$$

As the average of rotations is not necessarily a rotation itself, $\tilde{\mathbf{R}}_t^{l-1}$ needs to be projected back to the closest rotation, which closely resembles the local step of our local-global approach.

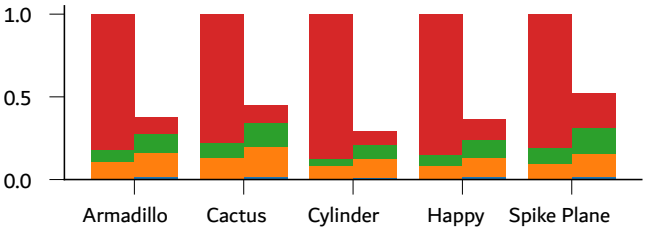


Figure 8: *Stacked runtime for 5 meshes. Hierarchy (blue), precompute (orange), constrain (green), solve (red).*

7. Results

In the following section we report on experimental results. All data were collected on an Intel i7-7700K CPU with 4 cores (8 threads) at 4.5Hz each and 32 GB RAM. For numerical linear algebra we use the Pardiso Cholesky solver provided by IntelMKL. Computation of rotations is optimized with parallelization and AVX instructions, included in libigl [JP*18].

The classic ARAP optimization can be split into 3 distinct phases:

Precomputation: Computing the Laplace matrix and setting up all additional data structures required during the solve.
Constraints: Updating the Laplace matrix given a set of constrained vertices and factorizing the resulting matrix.
Solving: Minimizing the ARAP energy given a specific set of target positions for the constraints.

For the hierarchical algorithm this is preceded by the computation of the mesh hierarchy. The phases allow us to distinguish between different types of interaction scenarios. If the user interacts with a fixed set of constrained vertices, we only have to perform the *solving* phase. Updating the set of constraints requires additionally re-doing the *constraining* phase. We performed comparisons on a set of meshes with a varying number of vertices. The coarsest level is chosen to have at least 750 vertices and we used a convergence criterion of 10^{-6} on all levels. The topology awareness is enabled in the hierarchy computation. The runtime for each phase of the algorithm is shown in Table 1 (some meshes can be seen in Figures 7 and 1).

Notice that the hierarchical approach converges earlier and has in most cases a lower energy. Figure 8 visualizes the required time per phase. For ARAP, most of the time is spent in the solve phase, unlike the hierarchical approach. Moving the constraints, therefore, requires only a fraction of the total time. The cost of updating the set of constrained vertices is also only slightly higher for the hierarchical approach but can become more pronounced for larger meshes (compare Table 1, Lucy).

Table 1: *Comparison of ARAP with hierarchical ARAP. The columns correspond to the time for the hierarchy computation (H), precomputation (P), constraining (C), solving (S), as well as the total time (T), the resulting energy Energy (E) and max. memory usage (M, not optimized). Top: ARAP, Bottom: HARAP.*

Mesh	$\|\mathcal{V}\|$	$H_{(s)}$	$P_{(s)}$	$C_{(s)}$	$S_{(s)}$	$T_{(s)}$	E	$M_{(GB)}$
Cactus	120K	-	0.5	0.4	2.9	3.8	0.0028	0.6
		0.08	0.7	0.6	0.4	1.7	0.0025	0.7
Spike Plane	130K	-	0.4	0.4	3.6	4.5	0.0045	0.7
		0.08	0.6	0.7	0.9	2.3	0.00063	0.8
Armadillo	170K	-	0.9	0.6	7.1	8.6	0.0019	0.9
		0.15	1.2	1.0	0.8	3.2	0.0017	1.0
Yeah Right	380K	-	1.5	1.4	6.7	9.7	0.0014	2
		0.26	1.8	2.0	1.9	6.0	8.5e-05	2.2
Cylinder	480K	-	2.2	1.2	23.1	26.5	0.0033	2.6
		0.34	3.1	2.1	2.1	7.7	0.0006	2.8
Happy	490K	-	2.2	1.7	22.6	26.5	0.0053	2.7
		0.42	3.0	2.8	3.4	9.6	0.004	2.9
Dragon	1.2M	-	5.9	6.1	115.4	127.4	0.0021	6.3
		1.35	10.4	10.8	4.4	27.0	0.0014	7
XYZRGB Dragon	3.6M	-	16.1	11.1	68.6	95.9	0.0029	19.3
		3.27	26.3	20.6	10.6	60.8	0.003	21
Lucy	5M	-	23.8	25.5	131.4	180.6	0.00048	26.6
		6.66	45.1	47.1	16.3	115.1	0.00077	29.8

Figure 9: *Deformed Lucy. The hierarchical approach diffuses the error on the lower levels. ARAP is not always able to sufficiently diffuse the error. (Left: HARAP, Right: ARAP)*

For extremely large meshes we found that the hierarchical approach terminated a lot earlier than the classic ARAP approach but this sometimes came at the cost of a slightly higher higher energy. In Figure 9 we can see that the hierarchical approach favors globally smooth deformations which are usually preferable. The single level ARAP doesn't manage to diffuse the error as well as our hierarchy, although its termiantes in a lower energy state. And even for the very narrow Yeah Right mesh the hierarchical approach proved beneficial.

The hierarchical approach improves the global smoothness, yet there is strong local bending in some meshes, e.g. cylinder. We have implemented a recent smoother version of the energy [OHSH25] and found similar benefits in runtime as reported for the classic ARAP energy (see Figure 10).

We also tested our implementation on a set of meshes from the

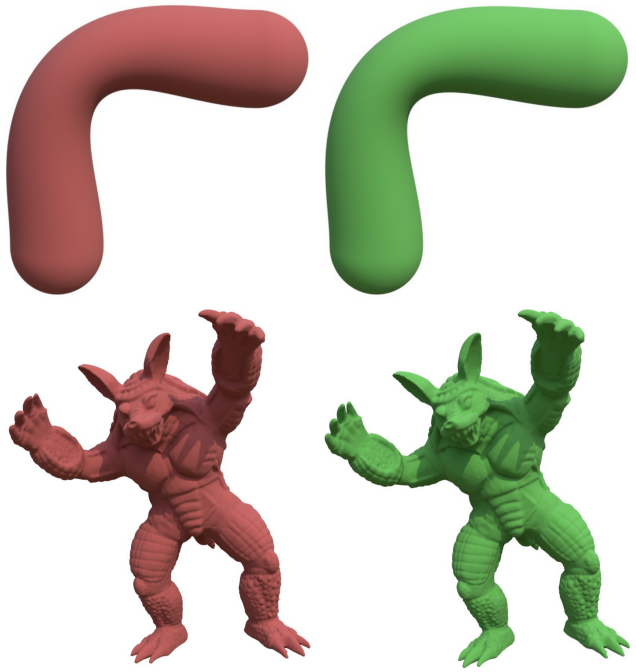


Figure 10: *Augmenting the ARAP Energy with a smoothness term [OHSH25] results in smoother meshes. Optimization with ARAP took 24.6s for the armadillo and 25.9s for the cylinder. For Hierarchical ARAP, it took 5.7s for the armadillo and 13.6s for the cylinder (including precomputation).*

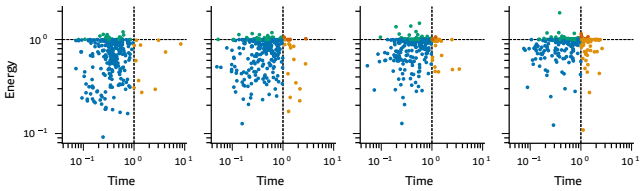


Figure 11: *Relative total runtime versus relative final energy of the hierarchical ARAP method, normalized by standard ARAP. Results are shown for convergence tolerances (left to right)* 10^{-6}, 10^{-7}, 10^{-8}, *and* 10^{-9}. *Each point represents one test case. Colors encode the quadrant of each result, indicating whether the hierarchical method converged faster and/or achieved a lower energy than the baseline. The ideal outcome corresponds to the bottom-left quadrant, where the hierarchical approach both terminates earlier and yields a lower energy.*

Thingi10K dataset [ZJ16]. Figure 11 shows that for most meshes hierarchical ARAP terminates earlier than ARAP at a lower energy. With a stricter convergence criterion, the hierarchical approach often terminates later than ARAP at the benefit of an even lower energy. For some meshes the hierarhical approach terminates later with a higher energy. In most of these cases, this was caused by poorly chosen constraints, which meant that a coarse deformation approximation did not coincide with the fine result. Examples of these deformations can be found in the supplementary material.

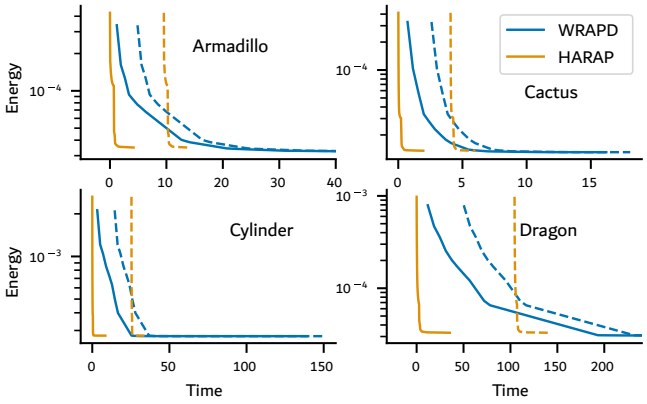


Figure 12: *Comparison of ARAP energy for HARAP and WRAPD [BN21]. Solid lines represent solve only and dashed lines include all precomputations. Meshes are tetrahedralized versions of the meshes in Figure 7 with the same constraints.*

This is consistent with the assumption that the deformations on coarse level should provide increasingly accurate approximations on each level.

As our approach is also easily extendable to the deformation of volumetric meshes (see Section 6) , we compare it to WRAPD [BN21]. While WRAPD is an improvement with regard to the classic ARAP optimization, our hierarchical ARAP optimization still outperforms WRAPD (Figure 12).

We also modified the hierarchical approach by using WRAPD at each level of our hierarchy. This requires filtering all inverted elements from the levels of the hierarchy. This can be easily done by extending the definition of degenerated elements to inverted elements. Additionally we are not able to upsample rotations since WRAPD internally does not exhibit rotations explicitly. We noticed that applying the hierarchy allowed for even faster convergence to slightly lower energies in most cases, as seen in the inset, showing precomputation time (P), solve time (S) and final energy (E) (top - HARAP, bottom - hierarchical WRAPD).

Mesh	$P_{(s)}$	$S_{(s)}$	$E_{(10^{-5})}$
Armadillo	9.3	30.1	3.3
	7.2	28.5	3.2
Cactus	4.1	10	1.32
	3.7	1.9	1.36
Cylinder	25.2	16.1	27.5
	17.9	7	27.4
Dragon	98.6	112.9	3.1
	70.8	85.7	3

We adapted classic vertex clustering to avoid merging separate components and show the potential influence of this step in Figure 13. The Cow mesh is self-intersecting, resulting in a connection between intrinsically distant part of the shape all coarse levels for standard vertex clustering. As the connection is only severed at the highest resolution, the hierarchy cannot provide a good initial solution for the high resolution requiring expensive correction. For the dragon mesh, this effect is less pronounced as the mesh is not self-intersecting, yet the simplified version would still create undesirable connections, hampering optimization. Avoiding these undesirable clusters with our approach leads to lower energy in a shorter time, because of reduced cost on the coarse levels.

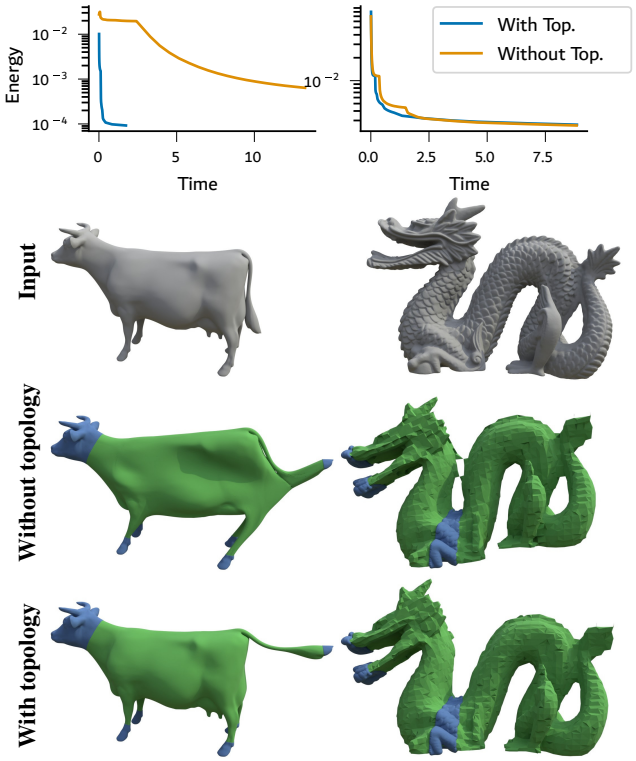


Figure 13: *Comparison of hierarchical optimization with and without topology awareness. Cow with self-intersection (left) and Dragon without self-intersection (right). Without topology awareness, connecting intrinsically distant parts of the mesh may inhibit the desired deformation on coarse levels.*

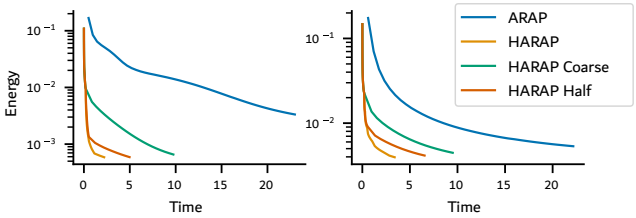


Figure 14: *Comparison of ARAP with using only the coarse level, every second level of the hierarchy and using all levels of the hierarchy. Cylinder (left) and Happy Buddha (right).*

In Section 5 we have explained how to potentially up-sample from any coarse intermediate solution directly to all finer meshes without the need for solving a global system at the intermediate levels. One might hope that this *skipping* could reduce to the total time for optimization, however, skipping rather tends to be a disadvantage (Figure 14). After optimizing at the coarsest level, we either up-sample directly to the finest level, or skip every other level in the hierarchy. In both scenarios the optimization on the next level quickly improves the solution but using coarse levels still provides a faster convergence. We argue that while the coarse solution helps propagate the information on the mesh, the coarse solution is still

Figure 15: *Upsampling rotations directly from a coarse solution leads to globally smooth approximations of the ARAP energy.*

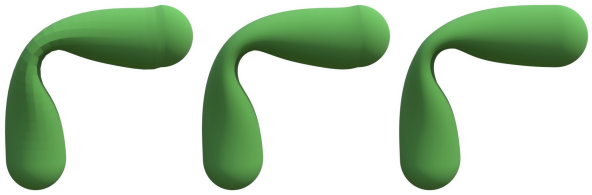


Figure 16: *Approximating smooth surfaces produces smooth patches (left). Averaging rotations during upsmapling smooths the rotation field (middle). U pdating c onstraints w ith c urrent rotations produces smooth transition between constrained and unconstrained regions (right).*

far from being a good ARAP solution. Similar to most multigrid solvers, the intermediate levels help by correcting errors and also allow for local smoothing, making them a valuable step during the hierarchical optimization.

As mentioned this also enables us to use this as an approximative ARAP scheme by upsampling the coarse rotations to the fine mesh directly and solving once, see Figure 15. However, for smooth surfaces, e.g. the cylinder, this results in a visual set of *patches* since the rotation within each *patch* is constant (Figure 16). A smoother result can be obtained by smoothing the rotation field during each upsampling step. We can do this by averaging the rotations of the 1-ring neighbourhood of the vertex and reprojecting to a valid rotation. This generates a smoother rotation field and, consequently, a smoother surface. The coarse constraints are based on the initial rotation and therefore are not correctly placed for the coarse mesh, leading to the bending at the end of the cylinder. Recomputing the coarse constraint positions improves the smoothness of the mesh.

Similar to an approximation, one could also use other embedded deformation methods [SSP07]. These approaches allow for fast computation of globally smooth results, see Figure 17. The optimization mostly depends on the dimension of the embedding space, so deforming large meshes with few handles can be faster. However, the efficiency c omes a t t he c ost l imiting s mall s cale deformations. Moreover, the choice of embedding limits deformations apriori and may cause artifacts. This can be related to our method on coarser levels, which also contain undesirable effects due to the simplification, yet these are usually resolved on finer levels of the hierarchy.

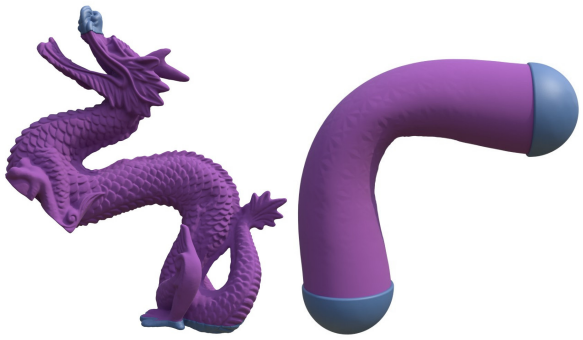


Figure 17: *Embedded deformations [SSP07] yields globally smooth results, but are not necessarily free of local artifacts (such as on the cylinder).*

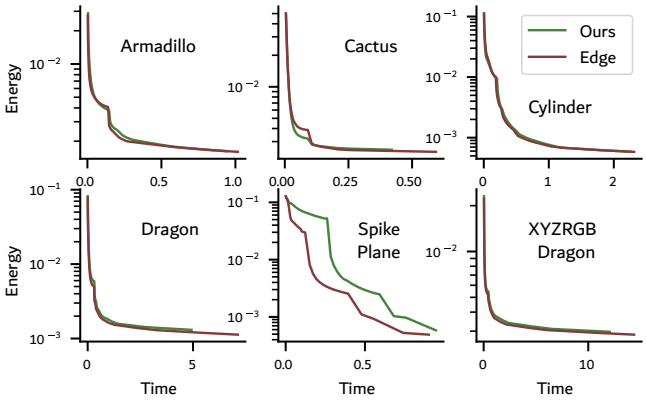


Figure 18: *The influence of the quality of the mesh geometry on the intermediate levels is negligible for most meshes. Comparing our hierarchy with a manifold preserving hierarchy results based on edge collapse results in similar runtimes and energies. For the spike plane, the edge collapse approach better preserves the original surface, resulting in improved convergence.*

The advantage of the proposed hierarchy is the simplicity of its calculation. However, this comes at the expense of the manifoldness of the mesh geometry. As shown, the hierarchy becomes largely non-manifold with some ares becoming flat. In comparison using an edge collapse algorithm [LZBCJ21] we can ensure its manifoldness. In Figure 18 we show that the influence of the manifoldness of the hierarchy is negligible for most meshes. This also has to be seen in correspondence to the computational complexity of the edge collapse algorithm, which surpasses the required computation time of the presented hierarchy.

8. Discussion & Future Work

We have shown that the ARAP energy can be minimized effectively in a hierarchical fashion. Remarkably, very good results are achieved if the energy is minimized on each level exactly in the same way usually done globally. This makes the approach very easy to adapt to variations of the energy.

Although the simplicity of the ARAP optimization is retained

within each level, the cascadic approach exhibits a few more parameters that have to be considered. First of all, the convergence criterion allows the user to control the tradeoff between time and a lower state of energy upon entering the highest resolution. Additionally, the number of levels in our hierarchy can be modified by omitting whole levels. Although this reduces precomputation time, it might lead to additional cost on finer levels during the optimization.

We have found that the choice of hierarchy has some effect on the optimization: rather than high quality in terms of geometry it is important to retain the intrinsic distance, which is what our suggested approach does at very small cost.

The benefit of the hierarchical approach compared to the direct global optimization also depends on the amount of deformation. As deformations get smaller, also the differences become smaller. Using a hierarchy improves global convergence, yet local changes still need to be considered on the fine levels.

We have demonstrated that our approach outperforms a state-of-the-art ADMM-based optimizer on tetrahedral meshes – even when accounting for precomputation. The local-global solver could also be replaced by the ADMM solver, further improving performance.

However, some of the acceleration techniques discussed in Section 3.3 impose additional constraints on intermediate solutions, such as requiring embedded triangulations. Our proposed upsampling strategy does not enforce these properties, and in our experiments, restoring them using existing methods diminished the overall effectiveness of the pipeline. Nevertheless, applying the classic local-global optimization at each hierarchical level still yields fast and reasonable initial solutions, making it a practical starting point for a wide range of subsequent optimization techniques.

Thin structures in the shape might collapse during vertex clustering, creating multiple components, which are optimized independently and could incur additional cost at higher levels.

Modern Cholesky solvers have become incredibly efficient. We have found that in our implementation the cost for setting up the RHS of the linear system takes up a substantial part of the time for the linear solve. Additionally, considering sparse updates of the system matrix, e.g. reconstraining, can also be efficiently handled by some more modern techniques.

Acknowledgment

This work was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant agreement No. 101055448, ERC Advanced Grand EMERGE). This material is based on work that is partially funded by an unrestricted gift from Google. Open Access funding enabled and organized by Projekt DEAL. We also thank Lars Munaf for his contributions to the implementation.

References

[And65] ANDERSON D. G.: Iterative procedures for nonlinear integral equations. *J. ACM 12*, 4 (Oct. 1965), 547–560. URL: https://doi.org/10.1145/321296.321305, doi:10.1145/321296.321305. 3

[BA09] BOUBEKEUR T., ALEXA M.: Mesh simplification by stochastic sampling and topological clustering. *Computers & Graphics 33*, 3 (2009), 241–249. IEEE International Conference on Shape Modelling and Applications 2009. doi:https://doi.org/10.1016/j.cag.2009.03.025. 3

[BBK05] BOTSCH M., BOMMES D., KOBBELT L.: Efficient linear system solvers for mesh processing. In *Mathematics of Surfaces XI* (Berlin, Heidelberg, 2005), Martin R., Bez H., Sabin M., (Eds.), Springer Berlin Heidelberg, pp. 62–83. 2

[BCDD22] BOURQUAT P., COEURJOLLY D., DAMIAND G., DUPONT F.: Hierarchical mesh-to-points as-rigid-as-possible registration. *Computers & Graphics 102* (2022), 320–328. doi:https://doi.org/10.1016/j.cag.2021.10.016. 3

[BDS*12] BOUAZIZ S., DEUSS M., SCHWARTZBURG Y., WEISE T., PAULY M.: Shape-up: Shaping discrete geometry with projections. *Computer Graphics Forum 31*, 5 (2012), 1657–1667. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03171.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03171.x, doi:https://doi.org/10.1111/j.1467-8659.2012.03171.x. 3

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial*. SIAM, 2000. 3

[BK04] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (New York, NY, USA, 2004), SGP '04, Association for Computing Machinery, p. 185–192. doi:10.1145/1057432.1057457. 3

[BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph. 33*, 4 (July 2014). URL: https://doi.org/10.1145/2601097.2601116, doi:10.1145/2601097.2601116. 3

[BN21] BROWN G. E., NARAIN R.: Wrapd: Weighted rotation-aware admm for parameterization and deformation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 40*, 4 (8 2021). 4, 10

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: Primo: coupled prisms for intuitive surface modeling. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Goslar, DEU, 2006), SGP '06, Eurographics Association, p. 11–20. 3

[CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (7 2010). doi:10.1145/1778765.1778775. 2, 7

[DGG14] DERZAPF E., GRUND N., GUTHE M.: Parallel progressive mesh editing. In *Proceedings of the 14th Eurographics Symposium on Parallel Graphics and Visualization* (Goslar, DEU, 2014), PGV '14, Eurographics Association, p. 33–40. 3

[EDD*95] ECK M., DEROSE T., DUCHAMP T., HOPPE H., LOUNSBERY M., STUETZLE W.: Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, Association for Computing Machinery, p. 173–182. doi:10.1145/218380.218440. 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216. doi:10.1145/258734.258849. 3

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., p. 325–334. doi:10.1145/311535.311577. 3

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd*

Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA, 1996), SIGGRAPH '96, Association for Computing Machinery, p. 99–108. doi:10.1145/237170.237216. 3

[HSL*06] HUANG J., SHI X., LIU X., ZHOU K., WEI L.-Y., TENG S.-H., BAO H., GUO B., SHUM H.-Y.: Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3 (July 2006), 1126–1134. URL: https://doi.org/10.1145/1141911.1142003, doi:10.1145/1141911.1142003. 3

[JBK*12] JACOBSON A., BARAN I., KAVAN L., POPOVIĆ J., SORKINE O.: Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4 (2012), to appear. 3

[JP*18] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 8

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, Association for Computing Machinery, p. 105–114. doi:10.1145/280814.280831. 3

[KGL16] KOVALSKY S. Z., GALUN M., LIPMAN Y.: Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph. 35*, 4 (July 2016). URL: https://doi.org/10.1145/2897824.2925920, doi:10.1145/2897824.2925920. 3

[LB15] LEGRAND H., BOUBEKEUR T.: Morton integrals for high speed geometry simplification. In *Proceedings of the 7th Conference on High-Performance Graphics* (New York, NY, USA, 2015), HPG '15, Association for Computing Machinery, p. 105–112. doi:10.1145/2790060.2790071. 3

[LBK17] LIU T., BOUAZIZ S., KAVAN L.: Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph. 36*, 4 (July 2017). URL: https://doi.org/10.1145/3072959.2990496, doi:10.1145/3072959.2990496. 3

[Lin03] LINDSTROM P.: Out-of-core construction and visualization of multiresolution surfaces. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2003), I3D '03, Association for Computing Machinery, p. 93–102. doi:10.1145/641480.641500. 3

[LJBA13] LIMPER M., JUNG Y., BEHR J., ALEXA M.: The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum 32*, 7 (2013), 197–206. doi:https://doi.org/10.1111/cgf.12227. 3, 4

[LZBCJ21] LIU H.-T. D., ZHANG J. E., BEN-CHEN M., JACOBSON A.: Surface multigrid via intrinsic prolongation. *ACM Trans. Graph. 40*, 4 (2021). 3, 11

[LZX*08] LIU L., ZHANG L., XU Y., GOTSMAN C., GORTLER S. J.: A local/global approach to mesh parameterization. In *Proceedings of the Symposium on Geometry Processing* (Goslar, DEU, 2008), SGP '08, Eurographics Association, p. 1495–1504. 7

[MAK24] MERCIER-AUBIN A., KRY P. G.: A multi-layer solver for xpbd. *Computer Graphics Forum 43*, 8 (2024), e15186. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.15186, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.15186, doi:https://doi.org/10.1111/cgf.15186. 3

[MS11] MANSON J., SCHAEFER S.: Hierarchical deformation of locally rigid meshes. *Computer Graphics Forum 30*, 8 (2011), 2387–2396. doi:https://doi.org/10.1111/j.1467-8659.2011.02074.x. 3

[Mü08] MÜLLER M.: Hierarchical Position Based Dynamics. In *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2008)* (2008), Faure F., Teschner M., (Eds.), The Eurographics Association. doi:/10.2312/PE/vriphys/vriphys08/001-010. 3

[OBLN17] OVERBY M., BROWN G. E., LI J., NARAIN R.: Admm ⊇ projective dynamics: Fast simulation of hyperelastic models with dynamic constraints. *IEEE Transactions on Visualization and Computer Graphics 23*, 10 (Oct 2017), 2222–2234. doi:10.1109/TVCG.2017.2730875. 3

[OHSH25] OEHRI A., HERHOLZ P., SORKINE-HORNUNG O.: Higher order continuity for smooth as-rigid-as-possible shape modeling, 2025. URL: https://arxiv.org/abs/2501.10335, arXiv:2501.10335. 9

[PDZ*18] PENG Y., DENG B., ZHANG J., GENG F., QIN W., LIU L.: Anderson acceleration for geometry optimization and physics simulation. *ACM Trans. Graph. 37*, 4 (July 2018). URL: https://doi.org/10.1145/3197517.3201290, doi:10.1145/3197517.3201290. 3

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experim. Math. 2* (1993), 15–36. 2

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics* (Berlin, Heidelberg, 1993), Falcidieno B., Kunii T. L., (Eds.), Springer Berlin Heidelberg, pp. 455–465. 3

[RPPSH17] RABINOVICH M., PORANNE R., PANOZZO D., SORKINE-HORNUNG O.: Scalable locally injective mappings. *ACM Trans. Graph. 36*, 2 (Apr. 2017). URL: https://doi.org/10.1145/2983621, doi:10.1145/2983621. 3

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (Goslar, DEU, 2007), SGP '07, Eurographics Association, p. 109–116. 1, 2, 8

[SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (New York, NY, USA, 2004), SGP '04, Association for Computing Machinery, p. 175–184. URL: https://doi.org/10.1145/1057432.1057456, doi:10.1145/1057432.1057456. 3

[SG05] SHAFFER E., GARLAND M.: A multiresolution representation for massive meshes. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 139–148. doi:10.1109/TVCG.2005.18. 3

[SHR17] SORKINE-HORNUNG O., RABINOVICH M.: Least-squares rigid motion using svd, 2017. 2

[SLS22] STEIN O., LI J., SOLOMON J.: A splitting scheme for flip-free distortion energies. *SIAM Journal on Imaging Sciences 15*, 2 (2022), 925–959. URL: https://doi.org/10.1137/21M1433058, arXiv:https://doi.org/10.1137/21M1433058, doi:10.1137/21M1433058. 4

[SPSH*17] SHTENGEL A., PORANNE R., SORKINE-HORNUNG O., KOVALSKY S. Z., LIPMAN Y.: Geometric optimization via composite majorization. *ACM Trans. Graph. 36*, 4 (July 2017). URL: https://doi.org/10.1145/3072959.3073618, doi:10.1145/3072959.3073618. 3

[SS15] SMITH J., SCHAEFER S.: Bijective parameterization with free boundaries. *ACM Trans. Graph. 34*, 4 (July 2015). URL: https://doi.org/10.1145/2766947, doi:10.1145/2766947. 3

[SSP07] SUMNER R. W., SCHMID J., PAULY M.: Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (July 2007), 80–es. URL: https://doi.org/10.1145/1276377.1276478, doi:10.1145/1276377.1276478. 2, 11

[SW03] SCHAEFER S., WARREN J.: Adaptive vertex clustering using octrees. *SIAM geometric design and computing 2*, 6 (2003). 3

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3 (jul 2006), 1108–1117. doi:10.1145/1141911.1142001. 3

[SZT*07] SHI X., ZHOU K., TONG Y., DESBRUN M., BAO H., GUO B.: Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM Trans. Graph. 26*, 3 (July 2007), 81–es. URL: https://doi.org/10.1145/1276377.1276479, doi:10.1145/1276377.1276479. 3

[Wan15] WANG H.: A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph. 34*, 6 (Nov. 2015). URL: https://doi.org/10.1145/2816795.2818063, doi:10.1145/2816795.2818063. 3

[WJBK15] WANG Y., JACOBSON A., BARBIČ J., KAVAN L.: Linear subspace design for real-time shape deformation. *ACM Trans. Graph. 34*, 4 (July 2015). doi:10.1145/2766952. 3

[WN11] WALKER H. F., NI P.: Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis 49*, 4 (2011), 1715–1735. URL: https://doi.org/10.1137/10078356X, arXiv:https://doi.org/10.1137/10078356X, doi:10.1137/10078356X. 3

[WNEH23] WIERSMA R., NASIKUN A., EISEMANN E., HILDEBRANDT K.: A fast geometric multigrid method for curved surfaces. *SIGGRAPH 2023 41*, 4 (July 2023). doi:10.1145/3588432.3591502. 3

[XTL19] XIAN Z., TONG X., LIU T.: A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Trans. Graph. 38*, 6 (Nov. 2019). URL: https://doi.org/10.1145/3355089.3356486, doi:10.1145/3355089.3356486. 3

[ZBK18] ZHU Y., BRIDSON R., KAUFMAN D. M.: Blended cured quasi-newton for distortion optimization. *ACM Trans. Graph. 37*, 4 (July 2018). URL: https://doi.org/10.1145/3197517.3201359, doi:10.1145/3197517.3201359. 3

[ZJ16] ZHOU Q., JACOBSON A.: Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016). 2, 9

[ZJA21] ZHANG J. E., JACOBSON A., ALEXA M.: Fast updates for least-squares rotational alignment. *Computer Graphics Forum 40*, 2 (2021), 13–22. doi:https://doi.org/10.1111/cgf.142611. 2

[ZPOD19] ZHANG J., PENG Y., OUYANG W., DENG B.: Accelerating admm for efficient simulation and optimization. *ACM Trans. Graph. 38*, 6 (Nov. 2019). URL: https://doi.org/10.1145/3355089.3356491, doi:10.1145/3355089.3356491. 3

[ZSGS12] ZOLLHÖFER M., SERT E., GREINER G., SÜSSMUTH J.: Gpu based arap deformation using volumetric lattices. In *Eurographics (Short Papers)* (2012), Andújar C., Puppo E., (Eds.), Eurographics Association, pp. 85–88. 3

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 259–268. doi:10.1145/258734.258863. 3