Practical Hex-Mesh Optimization via Edge-Cone Rectification

Marco Livesu [1] Alla Sheffer [1] Nicholas Vining [1] Marco Tarini [2,3]
[1] University of British Columbia [2] Universitá dell'Insubria [3] ISTI-CNR

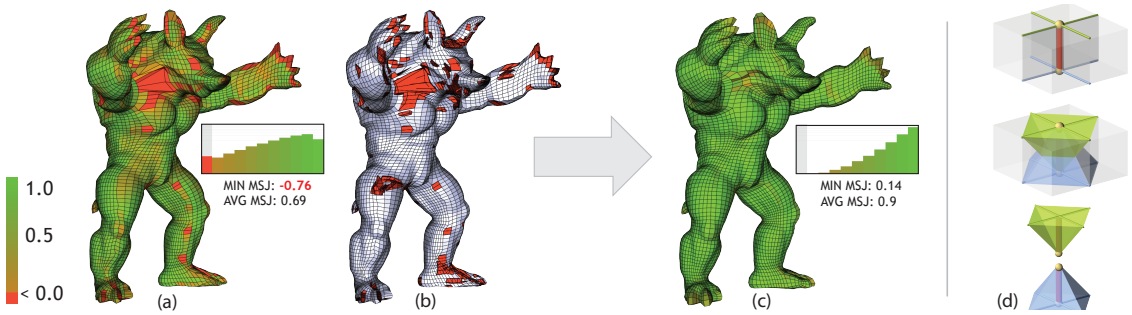


Figure 1: *We optimize poorly shaped hex meshes (a), with multiple inverted (red) elements (b), to produce high-quality, inversion-free outputs (c) using an* edge-cone *(d) based framework. The log-scale histogram insets highlight the improvement in worst element quality. Quality is measured using minimal Scaled Jacobian, with 1 being optimal and negative values corresponding to inverted elements.*

Abstract

The usability of hexahedral meshes depends on the degree to which the shape of their elements deviates from a perfect cube; a single concave, or *inverted* element makes a mesh unusable. While a range of methods exist for discretizing 3D objects with an initial topologically suitable hex mesh, their output meshes frequently contain poorly shaped and even inverted elements, requiring a further quality optimization step. We introduce a novel framework for optimizing hex-mesh quality capable of generating inversion-free high-quality meshes from such poor initial inputs. We recast hex quality improvement as an optimization of the shape of overlapping *cones*, or unions, of tetrahedra surrounding every *directed* edge in the hex mesh, and show the two to be equivalent. We then formulate cone shape optimization as a sequence of convex quadratic optimization problems, where hex convexity is encoded via simple linear inequality constraints. As this solution space may be empty, we therefore present an alternate formulation which allows the solver to proceed even when constraints cannot be satisfied exactly. We iteratively improve mesh element quality by solving at each step a set of local, per-cone, convex constrained optimization problems, followed by a global energy minimization step which reconciles these local solutions. This latter method provides no theoretical guarantees on the solution but produces inversion-free, high quality meshes in practice. We demonstrate the robustness of our framework by optimizing numerous poor quality input meshes generated using a variety of initial meshing methods and producing high-quality inversion-free meshes in each case. We further validate our algorithm by comparing it against previous work, and demonstrate a significant improvement in both worst and average element quality.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms;

Keywords: hexahedral meshes, mesh optimization

1 Introduction

Hexahedral meshes are the finite element discretization of choice in multiple engineering domains [Blacker 2000; Shepherd and Johnson 2008; Owen 2009]. The reliability of the finite element simulation is highly dependent on the mesh element quality, or the degree to which they deviate from a perfect cube [Pébay et al. 2007]. Simulation results depend on both average and minimum element quality [Labelle and Shewchuk 2007; Pébay et al. 2007]. Even a single *inverted*, or non-convex element, makes a mesh unusable for simulation. The generation of quality hexahedral meshes is typically a two step process which first generates an initial mesh whose connectivity is designed to fit the input at hand, and then modifies the vertex positions so as to optimize the mesh element shape while keeping the connectivity fixed [Owen 2009]. The initial meshes that serve as input to the optimization step frequently contain numerous poorly shaped, and even inverted, elements. We introduce a new mesh optimization method which takes an initial low quality hex mesh as input and generates an inversion-free high quality mesh closely conforming to the input surface geometry. This conformity is important for the finite element analysis to capture the true behavior of the meshed object. Our method recasts mesh optimization as a sequence of convex, easy to optimize problems and produces high-quality meshes starting from inputs with numerous inversions and a range of diverse connectivities representative of the commonly used initial hex mesh generation approaches (Section 4).

Hex mesh optimization is an open, challenging research problem [Knupp 2001; Shepherd and Johnson 2008; Ruiz-Gironés et al. 2014b]. Existing state-of-the-art methods focus on directly optimizing a range of hex shape quality metrics such as minimal [Knupp 2001; Brewer et al. 2003] or average [Ruiz-Gironés et al. 2014b] scaled Jacobian, or condition number [Knupp 2003; Brewer et al. 2003]. Optimizing shape quality directly is difficult since both the quality metrics and the element-convexity constraints, expressed as a function of vertex positions, are non-linear; the resulting solution space is non-convex, which in turn motivates these frameworks to employ Gauss-Seidel schemes that update one vertex position at a time.Our indirect, global approach, significantly improves on the results of these methods as demonstrated in Sections 2 and 4.

The key observation behind our optimization framework is that we can formulate shape quality improvement in an easier to optimize

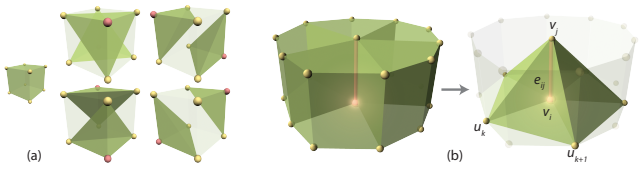


Figure 2: *A hexahedron is convex if all the tetrahedra associated with its corners (a) have positive volume; hex quality is maximized when the edges at each corner are orthogonal to one another. The set of tetrahedra defining the hex mesh quality can be iterated on by traversing the tets in the cones surrounding each directed edge (b).*

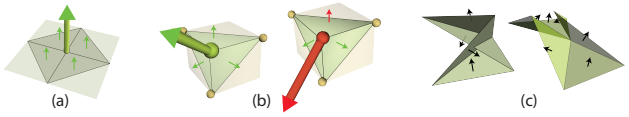


Figure 3: *(a) Edge-cone quality is maximized when its axis is parallel to the base face normals, decreasing as the angle between the normals and the axis grows (b), a cone becomes inverted (red) once the angle between at least one base face and the axis grows above* 90°*; (c) some base configurations may not allow an inversion free axis direction.*

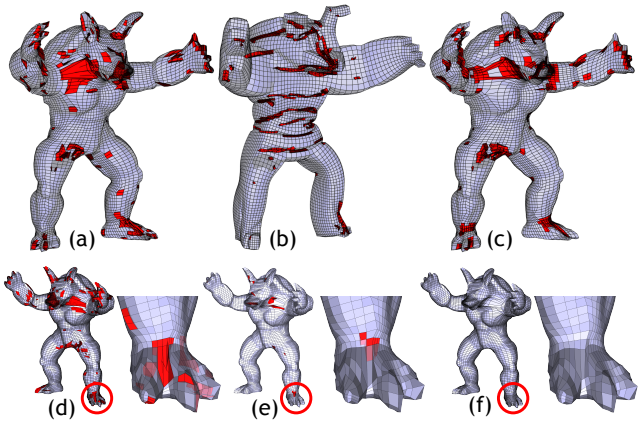


Figure 4: *Our method untangles meshes on which previous methods fail. (a,d) Inverted (red) elements in input Armadillo mesh (Figure 1); results of Mesquite [Brewer et al. 2003] without (b) and with (c) surface preservation. (e) Result of [Ruiz-Gironés et al. 2014a], (f) our result. As the zoomed in areas show the earlier method fails to untangle hexahedra with valid surface quad faces, and inverts a previously convex hex.*

form and explicitly constrain the minimal element quality within the mesh by changing the atomic elements we operate on. To this end, we note that the quality of a hexahedron is traditionally measured by analyzing the eight tetrahedra defined by the outgoing edges of each of its eight corner vertices [Pébay et al. 2007] (Figure 2a). In an ideal hex element, each directed edge originating from the corner is *parallel* to the normal of the face formed by the other hex edges originating from the same corner. A hex is inverted whenever the angle between even one edge direction and its corresponding face normal is over 90°. This observation motivates us to consider the hex mesh quality in terms of the relationships between these paired directed-edge and face-normal directions (Figure 3). Each such directed-edge and face pair forms a tetrahedron, and a union of such tetrahedra around the directed edge forms what we call a *cone* (Figures 1d, 2b). The cones of tetrahedra surrounding each directed edge in the mesh contain all the corner tetrahedra of its hex elements. We can therefore express mesh quality via the difference between the directions of the directed edges which form the *axes* of the cones and the normals of the triangular faces at their *base*. If these angles are under 90° for all the faces, for all cones in the mesh, the mesh is inversion free.

Instead of optimizing the shape of the mesh hexahedra, we optimize its equivalent - the shape of the edge-cones (Figures 1d, 2b). By recasting the optimization in terms of cone shape, and specifically the directions of the cone axes with respect to their bases, we formulate mesh quality optimization as a closed form constrained optimization problem. We demonstrate that mesh quality can then be optimized using an iterative algorithm, where at each iteration we minimize a convex quadratic function with linear inequality constraints which express the requirement for the angles between the axis and the face-normals within each cone to be under 90°. When this algorithm reaches a solution, the resulting mesh vertex positions are guaranteed to satisfy the inequalities and form an inversion-free mesh (Section 3).

However, this global algorithm can terminate without producing a solution; depending on the initial vertex positions, an intermediate constrained problem may become unsatisfiable (Figure 3c). The question of whether a given mesh connectivity allows for an inversion free mesh remains open [Shepherd and Johnson 2008] and such failure cases may be indicative of an empty solution space. However, in practice we observed that in most cases an inversion-free solution can be achieved by temporarily relaxing the constraints on the intermediate solutions. We therefore relax the stringent global constraints, replacing the global constrained optimization setting with an iterative local-global approach (Section 3.1.2). At each iteration we first solve a collection of independent local convex constrained optimization problems, computing an optimal inversion-free *target* axis direction for each edge cone with respect to the current geometry; if and when the solution space of a local problem is empty, we proceed using an unconstrained solution. We then update the vertex positions across the mesh using a global update step that moves vertices so that mesh edge orientations maximally conform to the target axis directions. The global step penalizes high local deviation from the desired target directions by using an incrementally reweighed penalty function guiding the optimization away from solutions with inverted elements. The resulting functional is quadratic, enabling robust and easy minimization.

The cone-based quality optimization formulation is augmented with surface preservation constraints, as well as additional quadratic regularization terms designed to stabilize the solution when the initial vertex positions are particularly poor (Section 3.2). For a typical mesh, with up to 200K hexahedral elements, the combined local-global framework requires less than 5 global iterations to converge to an inversion-free, high quality mesh in under 2 minutes. Furthermore the framework can be easily augmented to further improve the worst element quality, by gradually increasing the threshold in the local inequality constraints we solve for and tightening the target alignment in the global solution step (Section 3.6).

We validate our optimization framework by applying it to a range of initial meshes with different connectivity patterns created by recent meshing techniques, as well as to artificially corrupted versions of these meshes created by a combination of hex-element rotation and vertex displacement (Section 4). In all cases our method converges to an inversion-free, good quality mesh. We compare our approach to current state-of-the-art methods, showcasing the improvement in the average and worst element quality.

2 Related works

Our work builds upon recent advances in hexahedral meshing and mesh optimization, and is closely related to recent methods for low-distortion volumetric parameterization.

Hexahedral Meshing. Blacker [2000] highlights the importance of hex meshing to finite element simulation, referring to it as the "holy grail" of meshing research. Comprehensive surveys of hexahedral meshing methods can be found in [Shepherd and Johnson 2008; Owen 2009]. Recent methods can successfully create initial hexahedral meshes of reasonable quality employing grid-based [Schneiders 1996], octree-based [Marechal 2009], dual-graph [Tautges et al. 1996], swept [Miyoshi and Blacker 2000], frame-field based [Nieser et al. 2011; Li et al. 2012], and PolyCube-based [Gregson et al. 2011; Livesu et al. 2013; Huang et al. 2014] approaches. All these methods follow the initial mesh generation with a quality optimization step which employs one of the methods reviewed below. Section 4 demonstrates our method's application to the unoptimized outputs of a range of such approaches.

Shape Metrics Practitioners use a range of shape metrics, such as minimal or average scaled Jacobian [Pébay et al. 2007], to measure the quality of a hexahedral mesh, all aiming to estimate the degree to which the mesh elements differ from canonical cubes. These metrics, e.g. Minimal Scaled Jacobian (MSJ), are typically designed to have a value of one when quality is optimal, and to be negative when a hex is inverted. The main property these metrics try to capture is the degree to which the solid angles at the corners of each hex differ from the canonical 90° angles. The cone-shape metric we optimize maximizes the alignment between the cone axes and the normals of the base faces. Maximizing the alignment for each of the three cones each tetrahedron participates in is essentially equivalent to optimizing the canonical angles at the corners. Algorithms using traditional shape metrics also take into account disparity in hex edge lengths, prioritizing equilateral cube elements over elongated box-shaped ones and targeting edge lengths that conform to a user-specified sizing function. Our framework achieves this effect by optimizing the lengths of the edges toward their estimated target lengths computed to be similar for topologically parallel edges within each hex and for consecutive edges in face-adjacent hexes (Section 3.4).

Mesh Optimization Methods for mesh optimization [Owen 2009; Shepherd and Johnson 2008] are designed to improve the quality of the mesh elements while keeping the connectivity fixed and preserving the meshed domain boundary surface as much as possible. They strive to simultaneously achieve two goals: improving worst element quality - in particular correcting, or *untangling*, inverted elements - and optimizing the average mesh element quality. A large range of methods, e.g. [Freitag Diachin et al. 2006; Aigerman and Lipman 2013; Sastry and Shontz 2014], focus on tetrahedral mesh optimization. These results are not applicable as-is to hex meshes, motivating a separate line of research specifically targeting hex-mesh optimization, as reviewed by [Wilson 2011].

Moving vertices to some weighted average, or center, of their neighbors [Frey and George 2007; Zhang et al. 2009] is one of the oldest optimization approaches,, but it frequently results in inverted elements in concave mesh regions [Owen 2009]. Several methods [Vartziotis and Papadrakakis 2013; Knupp 2003] use an inversion-free mesh as a starting point, and relocate vertices one at a time while constraining each move to avoid inversions. In practice many raw hex-meshing outputs contain inverted elements, limiting the utility of this approach. In addition, by constraining the intermediate solutions at all times to remain in the inversion-free space, such method can converge to a local, far from optimal solution. Recently, Vartziotis and Himpel [2014] have proposed new formulations of optimization designed for mixed element meshes, but have yet to demonstrate those on a hexahedral or hex-dominant input. Sun et al. [2012] propose an optimization method specific for grid-based meshing.Our work addresses general mesh topology and preserves the input connectivity; we demonstrate its robustness across a vast range of inputs (Section 4).

The most common optimization approaches for correcting inverted hex elements use Gauss-Seidel style optimization operating one vertex at a time. Knupp [2001] proposes a dedicated method for correcting inverted hex elements, but highlights the difficulty of the problem and acknowledges that this method fails on many inputs. The widely used Mesquite library [Brewer et al. 2003] uses a combination of the algorithms in [Knupp 2001] and [Knupp 2003] to first untangle a hex mesh and then improve its quality iteratively moving one vertex at a time. Recent research [Wilson 2011; Ruiz-Gironés et al. 2014a; Wilson et al. 2012; Ruiz-Gironés et al. 2014b] uses iterative local Gauss-Seidel approaches to simultaneously correct inverted elements and improve the overall element quality. They use shape metrics specifically designed to prevent convex elements from becoming inverted. Our indirect global framework successfully untangles inputs that these methods fail on (Figure 4).

The use of global non-linear methods for optimizing mesh quality was investigated by Sastry et al [Sastry and Shontz 2009] and Wilson [Wilson 2011] for tetrahedral and hexahedral meshes respectively. Both concluded that global methods that directly optimize hex shape metrics as a function of vertex positions are typically less robust than the Gauss-Seidel approach, and frequently converge to poorer solutions.

Marechal [2009] computes the closest optimal hex (cube) for each individual hexahedron, and gradually moves mesh vertices towards the average of the obtained optimal vertex positions. The formulation balances shape quality against boundary surface preservation. As the author notes, using this method to achieve the hex quality necessary for accurate finite element simulation frequently requires large deviation from the input surface. Thus this method typically terminates with barely convex meshes, with Minimal Scaled Jacobian (MSJ) around 0.01; this quality is not sufficient for many simulation tools [Pébay et al. 2007].

Section 4 compares our results against those generated by the main methods above. We show that our method generates outputs with better average and worst element quality, while closely preserving the input surface geometry, and can produce inversion free outputs when others fail to do so.

Simplicial and Polynomial Map Optimization. Recent work on computation and optimization of maps between simplicial complexes (triangles and tetrahedra), e.g. [Aigerman and Lipman 2013; Schüller et al. 2013] is applicable to triangle and tetrahedral mesh optimization. E.g. Aigerman et al. [2013] successfully approximate an input tet mesh with multiple inverted elements with an inversion-free one. These methods use local-global approaches which interleave local solutions of a hard optimization problem with global solutions of simpler, linear or quadratic ones. A tempting solution for hex optimization would be to represent a hex mesh via a union of 8 overlapping tets and apply one of these methods. As discussed in Section 4, this formulation fails to reach the desired solution on many inputs. Our method is inspired by these algorithms and takes a similar local-global approach, but is designed specifically for hexahedral meshes. It successfully optimizes models that this mapping-based alternative fails on (Section 4).

Paille et al. [2013] optimize a mapping between an input surface and a higher-order polynomial hexahedral mesh with near perfect element shape, increasing the order of the hex elements as they progress to improve quality and surface fitting. The method is not applicable to standard linear hex meshes; while the higher-order elements may be inversion-free and high quality, the underlying linear mesh elements may be inverted (Section 4).

3 Formulation

As noted earlier, the quality of a hex mesh is measured using the shape of its eight corner tetrahedra (Figure 2a). These tetrahedra can be re-organized by grouping together each set of tetrahedra which

form a cone around a common oriented edge (Figure 2b). By traversing the cones around each oriented edge, we consider the same set of tets as traversing all corner tets of each mesh hexahedron. Contrary to all previous approaches which considered hex element quality directly, we formulate mesh quality optimization using the edge-cones as our atomic elements. As we show below, this alternative view avoids some of the pitfalls of earlier approaches, allowing for effective and robust mesh optimization.

Our formulation centers around a shape optimization term aimed at generating non-inverted tetrahedra inside each cone surrounding a directed edge and is framed in terms of the relationship between the directed edge, or *axis*, at the center of the cone and its base (Section 3.1). We cast cone shape optimization in a simple to optimize form, and design an iterative dedicated solver suitable for this task. The cone shape optimization term is augmented with a global regularity term aiming to stabilize the optimization process when the initial geometry is too unreliable to estimate axis directions locally (Section 3.2). We maintain the boundary surface geometry via additional, quadratic optimization terms discussed in Section 3.3 and describe the combined solution framework in Section 3.5.

Definitions Let $\mathcal{H}$ be a hexahedral mesh with vertices $\mathcal{V}(\mathcal{H})$ and edges $\mathcal{E}(\mathcal{H})$ respectively. For each directed mesh edge (v_i, v_j) we use the notation e_{ij} as a shorthand to denote the vector $v_j - v_i$. Thus while the equations below are frequently expressed in terms of e_{ij} the variables we optimize for are the underlying vertex positions. All equations are ultimately solved per-coordinate unless otherwise specified.

3.1 Cone Shape Optimization

Let $e_{ij} \in \mathcal{E}$ be a *directed* edge. If we consider the collection of all hexes H containing e_{ij}, let $Q = \{q_k\}$ be the umbrella of quads belonging to H containing i and not j, and let $T = \{t_k\}$ be the triangular fan defined by all vertices on Q that are connected to v_i, with the understanding that T has a winding order consistent with the winding order of the quads of Q, and that $t_{|T|} = t_0$ (Figure 2b). Denoting the vertices of t_k as u_k, v_i, u_{k+1}, in order for the tetrahedron e_{ij}, u_k, u_{k+1} to have a positive volume, v_j must lie on the positive side of the supporting plane of the triangle $v_i u_k u_{k+1}$ - for every triangle in T (Figure 3). The shape of each tetrahedron, e.g. its scaled Jacobian, is optimized if the edge is aligned with the triangle normal n_k.

Given a quadratic term $E_{boundary}$ expressing the conformity of the mesh boundary to the input surface (Section 3.3), we express mesh shape optimization balancing edge-cone quality against boundary preservation as computing vertex positions v_i that minimize

$$E_{cone} + E_{boundary} \tag{1}$$

$$E_{cone} = \sum_{e_{ij} \in E} \sum_{k=1}^{|T(e_{ij})|} (e_{ij}/\|e_{ij}\| - n_k)^2, \tag{2}$$

subject to non-inversion constraints and explicitly defining the normal n_k as a function of vertex positions,

$$e_{ij} \cdot n_k > 0 \; \forall e_{ij} \in E, t_k \in T(e_{ij}) \tag{3}$$

$$n_k - \frac{(u_k - v_i) \times (u_{k+1} - v_i)}{\|(u_k - v_i) \times (u_{k+1} - v_i)\|} = 0 \; \forall e_{ij} \in E, t_k \in T(e_{ij}). \tag{4}$$

The energy term intentionally counts the opposite directions of each edge separately, as the tightness of the constraints on these directions may significantly differ.

Given the typical size of real-world meshes, optimizing this formulation directly using standard non-linear optimization methods is impractical. In particular, the normalization by edge length (Equations 1, 4) may lead to ill-defined gradients and Hessians further complicating optimization. Inspired by the successful use of iterative local-global methods for other mesh processing tasks [Sorkine and Alexa 2007; Aigerman and Lipman 2013], we therefore design a targeted solver which makes this non-linear optimization tractable.

We observe that for typical input meshes, the lengths of the edges in the target output mesh can be pre-estimated based on either user-specified optimal sizing or the input mesh geometry and connectivity (Section 3.4). Using these estimated lengths L_{ij} in Equation 2 reduces it to a simple quadratic function:

$$E_{Qcone} = \sum_{e_{ij} \in E} \sum_{k=1}^{|T(e_{ij})|} (e_{ij}/L_{ij} - n_k)^2. \tag{5}$$

and reduces Equation 4 to a simpler quadratic form

$$n_k - \frac{(u_k - v_i) \times (u_{k+1} - v_i)}{L_{k,i} \cdot L_{k+1,i}} = 0, \; \forall e_{ij} \in E, t_k \in T(e_{ij}). \tag{6}$$

3.1.1 Global Formulation

Given this simpler formulation, we can solve the constrained optimization problem above using the following iterative solver.

1. Compute triangle normals n_k for each cone triangle of each directed edge (Equation 6).
2. Compute new vertex positions v_i by minimizing

   $$E_{Qcone} + E_{boundary} \tag{7}$$

   subject to the constraints

   $$e_{ij} \cdot n_k > 0, \; \forall e_{ij} \in E, t_k \in T(e_{ij}) \tag{8}$$

   using the approximate normals n_k computed in step 1. Note that this is a quadratic minimization problem with linear inequalities, solvable with standard quadratic programming methods. The problem is convex and has a unique minimum.
3. Repeat until convergence, i.e. until the vertex positions no longer change.

If and when the method converges, the resulting output mesh is **guaranteed** to have no inverted elements; when the process terminates, the vertex positions - and hence the normals - no longer change, and accordingly the non-inversion constraints (Equation 8) hold for the normals of the output mesh.

While simple and elegant, this algorithm may terminate prematurely failing to find a set of vertex positions that satisfies all the non-inversion constraints in Equation 8 at the same time. Such a situation can arise in the scenarios visualized in Figure 3c or in the case of degenerate base triangles, whose normals computed using Equation 6 have zero length. Such scenarios can and do occur in practice if the initial mesh is badly tangled. Note that since triangle normals can and do change as mesh vertices move, the fact that a current set of normals does not allow for a solution does not necessarily imply that a solution does not exist for the given mesh topology.

3.1.2 Local-Global Framework

We therefore design a more robust iterative solution framework which relaxes the inversion constraints just enough to avoid forming an empty solution space at any point. Specifically, we break the iterative optimization process into local and global steps.

In the local step, we find *target* edge directions for each cone axis which satisfy the non-inversion constraints with respect to the base triangle normals computed by the previous global step. If no local

solution exists that satisfies the constraints, the local step returns an unconstrained solution, as discussed below. In the global step we solve for new vertex positions that align the resulting mesh edges as closely as possible with the newly-computed target directions. To ensure that the method does not terminate prematurely, the global step uses penalty functions to minimize the occurrence of inversions rather than hard constraints.

Local Step. For each interior directed edge e_{ij}, we compute a target direction $\hat{n}_{ij}$ such that

$$\hat{n}_{ij} = \arg\min \sum_{k=1}^{|T(e_{ij})|} (\hat{n}_{ij} - n_k)^2 \tag{9}$$

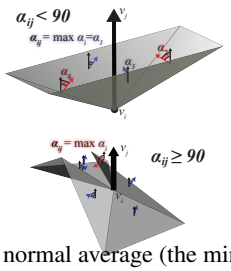
subject to the constraints

$$\hat{n}_{ij} \cdot n_k > \varepsilon > 0, \forall t_k \in T(e_{ij}) \tag{10}$$

$$\|\hat{n}_{ij}\|^2 \leq 1 \tag{11}$$

The last convex term replaces the explicit, but non-convex, requirement for $\hat{n}_{ij}$ to have unit length. The normals n_k are computed from the current, fixed, vertex positions using Equation 6 and are normalized to have unit length. The resulting vectors $\hat{n}_{ij}$ are similarly normalized. Note that if all we required is for the dot product $\hat{n}_{ij} \cdot n_k$ to be positive, we could eliminate the last constraint (Equation 11), as from an optimization perspective it would make no difference what the length of the solution was. However, if we intend to use a higher than zero threshold ε, set as discussed in Section 3.6, then we must limit the length explicitly to avoid the vectors from satisfying the threshold by scaling.

This is a classic quadratic convex optimization problem with linear and quadratic inequality constraints on the three coordinates of $\hat{n}_{ij}$. It can be solved efficiently and robustly using standard quadratic programming tools such as [Gurobi Optimization 2013]. Moreover, in most input configurations the simple average of the base triangle normals (the minimizer of Equation 9) satisfies the constraints as-is; therefore, to speed-up computation, we employ the QP solver only if this basic solution violates the constraints.



Since this is a constrained problem, a direction that satisfies all the non-inversion constraints may not always exist (Figure 3c). However, the advantage of a localized solution is that one can still proceed with the global iterations in a meaningful way.

If no inversion-free solution exists we return either the current edge direction or the normal average (the minimizer of Equation 9) basing the decision on the largest angle α_{ij} (see inset) between that vector's direction and the base face normals. We select the direction for which this angle is smallest. We make the same selection if one of the face normals has zero length. Since our optimization seeks for all edges to have a known target length and indirectly optimizes base triangle angles, such degenerate configurations are typically resolved during the first iteration of our method. For surface edges, we use a slightly modified formulation discussed in Section 3.3.

Global Step. Our global step computes new mesh vertex positions using the locally computed target edge directions, and aims to align the mesh edges with these directions by minimizing

$$E = E_{Qcone} + E_{boundary} \tag{12}$$

augmented by penalty functions that keep the new edge directions close enough to the target ones so as to avoid inversions. We use penalty functions for this task instead of explicit constraints in order to allow the method to proceed to the next iteration in cases where such constraints cannot be immediately satisfied.

Penalty Scheme. Traditional penalty schemes are formulated as augmenting a minimized energy function $\hat{E}$ with weighed penalty terms E_i^P which replace the hard constraints,

$$E = \hat{E} + \sum_i W_i^P E_i^P \tag{13}$$

where W_i^P are individual penalty weights [Nocedal and Wright 2006]. The methods then repeatedly minimize the augmented function, while increasing the weights W_i^P till the constrains are satisfied. Our observation is that we can use this same scheme but with an implicit upper bound on the weights W_i^P, such that if the constraints are not strictly satisfied we can still proceed with the next iteration of our local/global framework.

We introduce per-cone penalty terms aimed to keep the edge directions sufficiently close to the target ones. Our penalty formulation is based on the observation that the norm of the difference between unit-length vectors is above 1 when the angle between them is obtuse, and less than one, when the two are relatively close (under 60°). Thus for larger angles, a high order monomial of this norm serves as a simple penalty whose cost increases dramatically as the angle increases, strongly discouraging any solution where the new edge directions deviate far enough from the target to the point where a tetrahedron in the edge-cone becomes inverted. At the same time, the maximal distance between unit-length vectors is 2, providing a natural upper bound on the size of the penalty. Our penalty scheme is therefore guaranteed to terminate as the penalty cannot increase indefinitely. To keep the optimized functional quadratic, we separate the monomial into a quadratic term, which we treat as a penalty function

$$E_{ij}^P = \| \frac{e_{ij}}{\hat{L}_{ij}} - \hat{n}_{ij} \|^2 \tag{14}$$

and a higher-order weight term w_{ij}^P which is updated between penalty iterations, and is never decreased. The computation of the normalization factors $\hat{L}_{ij}$ is described below. The weights w_{ij}^P are initially set to 1. After a solution is obtained, we update the penalty weights by setting

$$w_{ij}^P = \max(w_{ij}^P, \| \frac{e_{ij}}{\hat{L}_{ij}} - \hat{n}_{ij} \|^6). \tag{15}$$

The choice of power six was empirical. We found that lower values did not sufficiently penalize large angular deviations, leading to decrease in solution quality, while larger powers reduced numeric stability: increasing the power increases the ratio between the largest and smallest elements of the gradient matrix of our energy function, which in turn increases its condition number.

When selecting the normalization factor $\hat{L}_{ij}$, focusing on angle only argues for considering a pure directional difference - i.e. normalizing the current edge by its length at the beginning of the global step. However, this choice is sub-optimal if and when the current length is very different from the target, as the current length may be a poor predictor of the length after optimization, one expected to get closer to the target. We anticipate this change by setting $\hat{L}_{ij} = (\|e_{ij}\| + L_{ij})/2$, where $\|e_{ij}\|$ is the current edge length.

We also note that the degree to which we want the edge and its target direction to be aligned depends on the geometry of cones computed in the local step. Specifically, we consider the largest angle α_{ij} between the target axis direction and the cone's base triangle normals used in the local step. To minimize inversion likelihood, when α_{ij} is below 90°, the alignment between the edge and the target direction should become tighter as the angle grows. We therefore multiply all weights w_{ij}^P by the following function of this angle,

$$W(\alpha_{ij}) = 1 + 10e^{\frac{-\cos\alpha_{ij}^2}{2\sigma^2}} \tag{16}$$

to penalize deviation more when the cone is "tighter". We set $\sigma = 0.3$ using the three sigma rule for the weight to drop to 1 when the angle is zero allowing for larger deviation.

Our combined penalty scheme optimizes at each iteration the following quadratic function,

$$E = E_{Qcone} + E_{boundary} + E_{penalty} \tag{17}$$

$$E_{penalty} = \sum_{ij} W(\alpha_{ij}) w_{ij}^P E_{ij}^P \tag{18}$$

We repeat the minimization and weight update steps (Equation 15) until the vertex positions or the combined energy function no longer change. Note that, as expected from penalty terms, the weights $W(\alpha_{ij})w_{ij}^P$ never decrease. Lastly we observe that a penalty should be applied only if the target directions satisfied the inequalities in the local step (i.e. $\alpha_{ij} < 90^\circ$). If this constraint is not satisfied we skip the corresponding edge in the sum above. The combined penalty scheme typically requires 5 or fewer linear iterations to converge.

3.2 Global Regularization

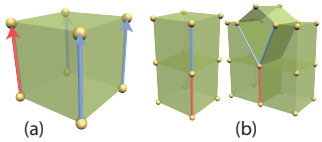


Figure 5: *(a) Topologically parallel edges; (b) consecutive edges.*

The formulation as described so far converges to a high quality mesh when initialized from a good starting point (e.g. models such as clef2 or the bust in Figure 6). However the framework can become unstable when the initial mesh contains large numbers of inverted elements (e.g. part29, Figure 6). In this scenario multiple solution spaces for the local step (Section 3.1.2) can be either empty, or highly constrained, resulting in very different target directions for the two half-edges e_{ij} and e_{ji} belonging to the same edge. We stabilize the solution framework by introducing a regularization term aimed at generating more unified edge directions for adjacent mesh edges which leverages the structure of the hexahedral mesh. We observe that for high quality hex meshes the directions of topologically parallel edges within the same hex (Figure 5a) and of consecutive edges on face-adjacent hexes (Figure 5b) are expected to be quite similar. Optimizing for direction similarity of such edges helps guide the target edge directions across the mesh toward a more uniform solution, stabilizing the algorithm, motivating our regularity term,

$$E_{regularize} = \sum_{e \in E} \frac{1}{|P(e)|} \sum_{e_p \in P(e)} \| \frac{e}{L_e} - \frac{e_p}{L_p} \|^2 + \sum_{e \in E} \frac{1}{|C(e)|} \sum_{e_c \in C(e)} \| \frac{e}{L_e} - \frac{e_c}{L_c} \|^2 \tag{19}$$

Here $e_p \in P(e)$, are the edges topologically parallel to e and $e_c \in C(e)$ are its consecutive edges. We normalize all vectors by target edge lengths to optimize for the desired length proportion. Note that while this regularizer is very useful in stabilizing the solution and speeding convergence, it is not a substitute to the cone optimization. Optimizing $E_{regularize}$ alone would result in meshes with inverted elements on most inputs.

3.3 Boundary Surface Preservation

A key requirement in mesh optimization is to preserve the outer surface of the mesh. The simplest, and most frequently used way to achieve this is to fix the positions of the surface vertices, e.g. [Ruiz-Gironés et al. 2014a; Brewer et al. 2003]. Our system supports this option, when the surface needs to be preserved exactly. However, such positional constraints are frequently too restrictive, as often mesh quality can be improved by allowing surface vertices to slide along the boundary surface, see Figure 1). Moreover, in some configurations (see the degenerate and near degenerate elements on the Dragon model, Figure 11) one cannot obtain an inversion-free mesh without allowing the mesh vertices to deviate from the surface. Thus we propose a boundary formulation that balances surface preservation against mesh quality.

To optimize the quality of hexahedra along the surface, we include in our set of cones the partial cones around each surface edge. As we aim for surface edges to remain on the surface, in our local step (Section 3.1.2), we modify Equation 9 for surface edges:

$$\hat{n}_{ij} = \arg\min \sum_{k=1}^{|T(e_{ij})|} (\hat{n}_{ij} - n_k)^2 + \beta n_{ij} \cdot \breve{n} \tag{20}$$

where $\breve{n}$ is the average of the original surface normals at v_i and v_j and $\beta = 10$. The treatment of boundary edges in the global step is similar to that of interior edges.

Expressing exact or approximate surface preservation in closed form (instead of just preserving the current locations) requires an analytic surface definition, which is rarely available. Thus to constrain vertices to remain on or close to the surface we use a local surface approximation. We distinguish between regular surface vertices $v \in S$, feature vertices $v \in F$, and corners (vertices at the junction of multiple features) $v \in C$. To preserve the surface we use three types of per-vertex energy terms, selected by vertex type:

$$E_{boundary} = \sum_{v \in S} \beta(\hat{n} \cdot v + \hat{d})^2 + \sum_{v \in F} (\alpha(v - (\hat{v} + a\hat{t}))^2 + a^2) + \sum_{v \in C} \alpha(v - \hat{v})^2 \tag{21}$$

Here $\hat{v}$ is the reference, or closest, surface position for each vertex, $<\hat{n}, \hat{d}>$ is the implicit equation of the plane passing through $\hat{v}$ and orthogonal to the input surface normal $\hat{n}$ at that point, and $\hat{t}$ is the feature tangent at $\hat{v}$. a is an auxiliary variable added to the system to enable feature constraints. We use $\alpha = 20$ as the feature and corner weight. The boundary term is quadratic, thus augmenting our combined energy functions with it does not affect the problem convexity.

3.4 Optional Preprocessing

Our framework uses target, or optimal, edge lengths in a number of places in the formulation above. In most meshing frameworks, such targets are typically provided by users and are based on simulation needs. Absent this high-level knowledge, when the mesh density is by design non-uniform (e.g. for meshes created with octree-based methods), the input edge lengths provide the most plausible estimation of the target edge length. If the mesh density is expected to be uniform, we can provide a better initial length estimate by leveraging the existing surface mesh sizing. We note that in general one expects mesh size changes to be gradual, indicating a preference for similar lengths on both topologically parallel and consecutive edges, and that as we seek a uniformly sized mesh, the average edge length in the current mesh L_A serves as a good initial approximation of target interior mesh edge lengths. These observations combined motivate the following formulation for obtaining target edge lengths L_e

$$E_{length} = \sum_{e \in S'} (L_e - \|e\|)^2 + \sum_{e \in E \setminus S'} (L_e - L_A)^2 + \sum_{e_p \in P(e)} (L_e - L_p)^2 + \sum_{e_c \in C(e)} (L_e - L_c)^2. \tag{22}$$

where S' are the surface edges and $\|e\|$ are original edge lengths. The resulting edge lengths balance the expectation of uniformity against the desire to preserve the current surface edge lengths.

Once our solver converges, the resulting set of edge lengths better reflects the established balance between preservation of the initial target lengths and the desire for high element quality. As such, these lengths serve as better targets if one wishes to further improve the mesh quality. The optimization can be either simply restarted with these new lengths, or alternatively target lengths can be updated selectively, reflecting the local mesh element quality, by linearly interpolating between the target L_e and current $\|e\|$ edge lengths,

$$L_e = (1 - w) \cdot L_e + w\|e\| \tag{23}$$

Here w is a Gaussian function of the smallest hex element MSJ q of any hex connected to the target edge:

$$w = e^{-0.5 \cdot (\frac{q}{\sigma})^2} \tag{24}$$

where $\sigma = 0.15$. This selective update aims to preserve the original target lengths in areas where the mesh quality is sufficiently high, so as to preserve the mesh in these areas as is, while updating them in areas where the quality is poor.

3.5 Combined Framework

While the derivations above may seem lengthy the resulting formulation and code are surprisingly simple as highlighted by the pseudocode in Algorithm 1. We begin the optimization by estimating the ideal mesh edge lengths (Section 3.4). We then iterate the local and global steps (Section 3.1.2). In the global step we employ penalty functions to optimize the alignment between the mesh edges and their target directions. Each penalty step minimizes the combined energy function

$$E = E_{Qcone} + E_{penalty} + E_{regularize} + E_{boundary} \tag{25}$$

Since the minimized functional is quadratic, we use a standard linear solver (GMRES) to compute the minimizer. Once the iterations converge (i.e. the vertex positions or the energy value no longer change), one can optionally update the target lengths and repeat the process.

Algorithm 1 Hex-mesh optimization via cone rectification.

```
1: procedure OPTIMIZE(hexmesh)
2:     Compute target edge lengths, minimizing Eq. 22 (optional)
3:     repeat/*Local/Global Solve */
4:         Local Step: Compute Target Axis Directions n̂
5:         Global Step:
6:         Set weights w_ij^P = 1
7:         repeat/* Penalty Function Update */
8:             Minimize Eq. 25
9:             Update weights w_ij^P (Eq. 15)
10:        until Positions/energy (Eq. 25) converge w.r.t. previous penalty step
11:    until Positions/energy (Eq. 25) converge w.r.t. previous global solve step
```

3.6 Maximizing Minimal Quality

As described, the framework above is designed to obtain inversion-free meshes with high average element quality. However, many simulation setups have minimal, or worst, element quality restrictions that go beyond simple convexity [Pébay et al. 2007]. Our default setup aims for minimal angle between the cone axis and face normals to be 85° and uses $\varepsilon = cos(85)$ in Equation 10. To achieve a higher minimum, we propose a slight variation on our algorithm, designed to progressively improve the minimal quality.

To this end we introduce two simple changes to Equations 10 and 16. Once all local solution spaces are not empty, i.e. all cone angles α_{ij} are below 90°, we replace the default minimal angle threshold ε (Equation 10) with a local threshold whose goal is to further improve the local cone-quality,

$$\hat{n}_{ij} \cdot n_k > \cos(\alpha_{ij} + \epsilon) \text{ for all } t_k \in T(e_{ij}) \tag{26}$$

Here α_{ij} is the worst current angle between the directed edge e_{ij} and its corresponding cone base and $\epsilon = 0.01$. This change requires all target axis directions computed by the local step to be better aligned with respect to the cone's base than they are right now. We keep the previous target direction if and when this requirement results in an empty solution space. We make no effort to improve the local cones if α_{ij} is above the high-quality threshold of 0.5 used in literature [Pébay et al. 2007]. We also compute the current worst angle across all cones ω. If this value is under 90°, we update the angle based weight in the penalty function (Equation 16) to be maximized at this worst angle,

$$W(\alpha_{eij}) = 1 + 10e^{\frac{-(\cos(\alpha_{ij}) - cos(\omega))^2}{2\sigma^2}} \tag{27}$$

We run the method with these updates, recomputing α_{ij} and ω each time the process converges. We stop when further improvement is no longer possible, i.e. ω does not increase or once it reaches a user specified minimal quality value.

4 Results

We tested our method on a range of input hex meshes generated using a variety of state-of-the-art algorithms: PolyCube-based [Gregson et al. 2011; Livesu et al. 2013; Huang et al. 2014] (bust, armadillo, bunny, dancing children, cap, block, and dragon), singularity-restricted field [Li et al. 2012] (hanger, impeller), grid-based [Schneiders 1996] (cad6, part29, it-vhs) and octree-based [Marechal 2009] (asm001, asm106, clef2) approaches. The number of inverted elements in these meshes varied from none, for the octree-based method, to several thousand for the grid-based. In all cases our method generated inversion-free outputs and significantly improved the worst element quality, with minimal deviation from the original surface (Table 1, Figure 6). The deviation, measured using [Cignoni et al. 1998], is typically on-par or even smaller than what is reported by recent hex-meshing methods, e.g. [Gregson et al. 2011; Huang et al. 2014]. We use the input edge lengths as targets for models with grading (octree-based models and sphere plane), and estimate the target edge lengths on all other models.

Model	# hexa	Pre neg. el.	Pre avg / min	Post avg / min	avg. dist.	time sec.
asm001	25K	0	.83 / .08	.87 / .17	2.0×10^{-4}	31
asm106	120K	0	.86 / .02	.91 / .25	8.6×10^{-5}	110
clef2	10K	0	.86 / .05	.90 / .34	1.5×10^{-4}	3
Bunny	37K	1	.90 / -.01	.97 / .61	4.6×10^{-4}	20
Dancing ch.	35K	5	.86 / -.01	.94 / .35	2.2×10^{-4}	19
King-Kong	160K	11	.95 / -.76	.97 / .27	4.1×10^{-4}	119
Cap	4.5K	50	.74 / -.94	.87 / .11	6.6×10^{-4}	91
Bust	5.2K	30	.73 / -.59	.92 / .11	9.0×10^{-4}	4
Dragon	14K	84	.82 / -.99	.90 / .10	5.9×10^{-4}	250
part29	52K	1.3K	.87 / -.88	.93 / .35	2.9×10^{-4}	69
it-vhs	72K	3K	.78 / -.86	.88 / .35	1.4×10^{-4}	85
cad6	73K	1K	.86 / -.89	.93 / .29	1.6×10^{-4}	58

Table 1: *Hex quality statistics for various models processed with our algorithm. Left to right: Model name, # of hex elements, # of inverted elements in input, average/minimum MSJ prior to optimization, average/minimum MSJ after optimization, average deviation from input surface measured using Metro [Cignoni et al. 1998], and run-time of our algorithm in seconds. MSJ optimal value is one.*

To show that our method can successfully converge to a good solution from a poor initial guess with unreliable edge lengths we

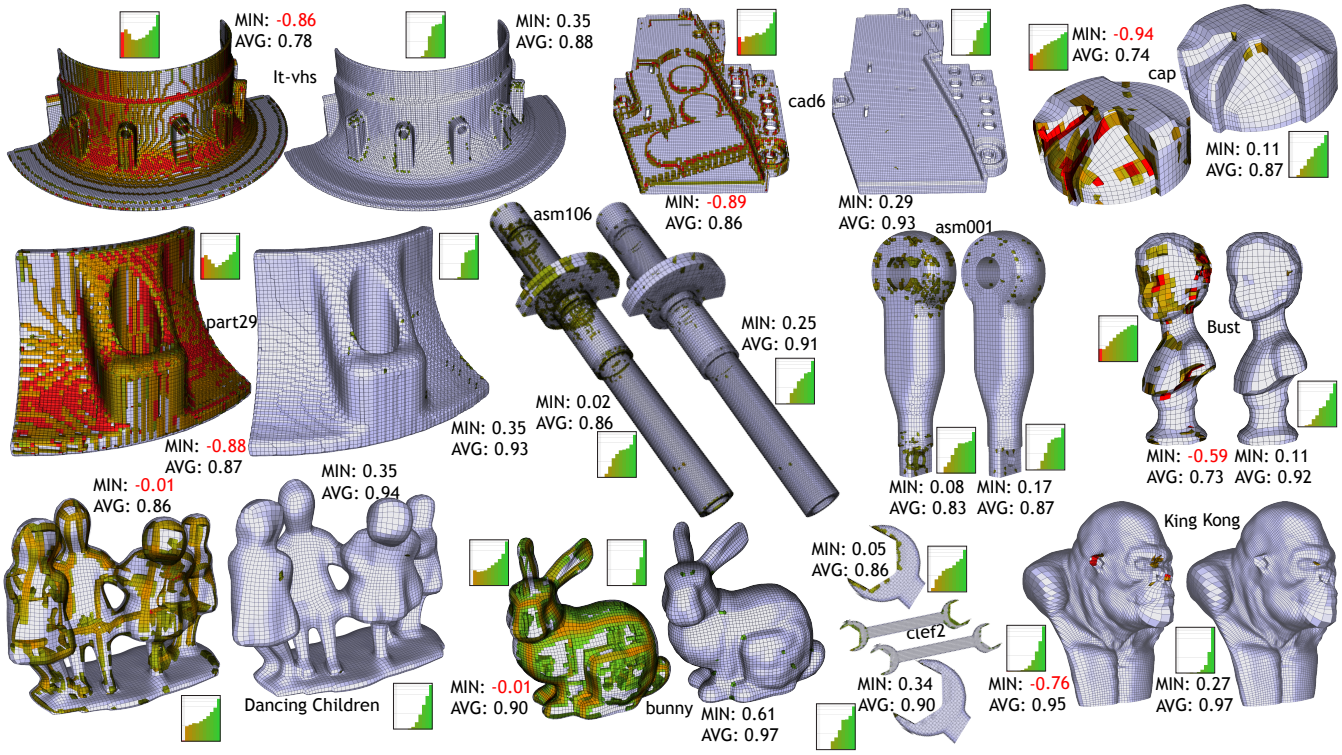


Figure 6: *Representative optimization results. For each model we show semi-transparent views visualizing the worst quality output elements and all input elements of same or lower quality. Non-convex elements shown in red. Insets show MSJ histograms using a log scale.The bust and dancing children models are provided courtesy of IMATI/Frankter Haar by the AIM@SHAPE-VISIONAIR Shape Repository.*

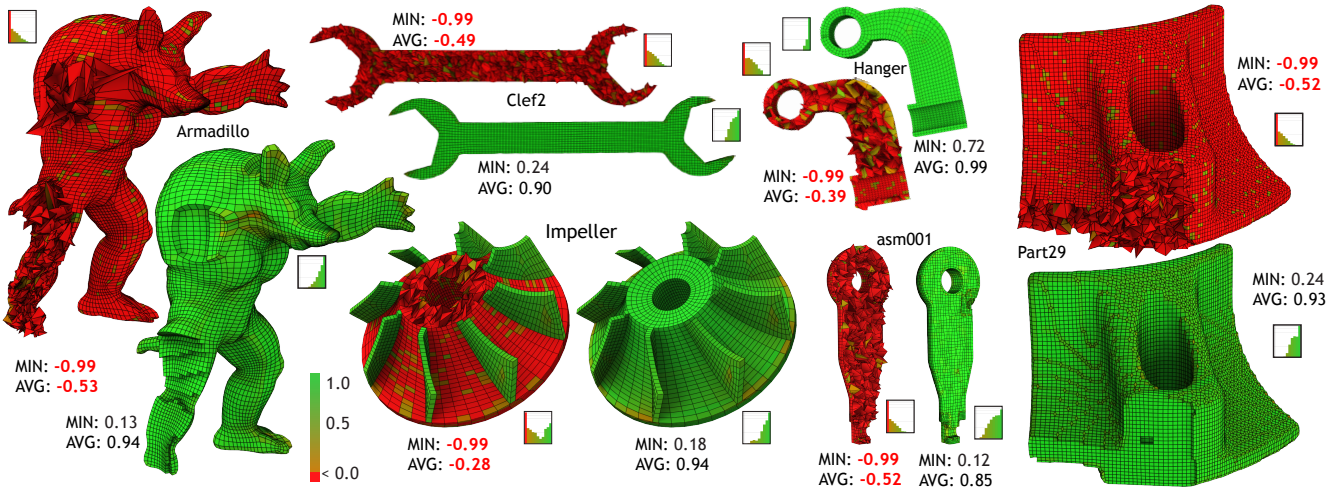


Figure 7: *Results on artificially corrupted inputs highlight our method's robustness. Mesh cross-sections used to better visualize hex shape.*

Model	# hexa	Pre neg. el.	Pre avg / min	Post avg / min	avg. dist.	time sec.
asm001	25K	24K (96%)	-.52 / -.99	.85 / .12	2.7×10^{-4}	779
clef2	10K	9K (90%)	-.49 / -.99	.90 / .24	1.4×10^{-4}	109
Hanger	4.5K	4K (89%)	-.39 / -.99	.99 / .72	1.0×10^{-4}	11
Impeller	11K	9K (82%)	-.28 / -.99	.94 / .18	5.5×10^{-5}	37
Armadillo	30K	28K (93%)	-.53 / -.99	.94 / .13	2.5×10^{-3}	102
part29	52K	50K (96%)	-.52 / -.99	.93 / .24	3.4×10^{-4}	180

Table 2: *Statistics for corrupted versions of our input models.*

generated a range of "stress tests" by artificially corrupting the input meshes, via a combination of hex-element rotation and vertex displacement (Table 2, Figure 7). These corrupted meshes have up to 90% inverted elements. Our method successfully untangled these inputs, generating inversion-free, good quality results. We also tested our method by displacing a random interior vertex and holding it in place using hard constraints while optimizing the mesh. In our experiments, we were able to displace an interior vertex by up to 400% of the average interior edge length and still achieve an inversion-free result.

Convergence and Runtimes. Figure 10 visualizes the typical convergence behavior of our method. As demonstrated, on regular

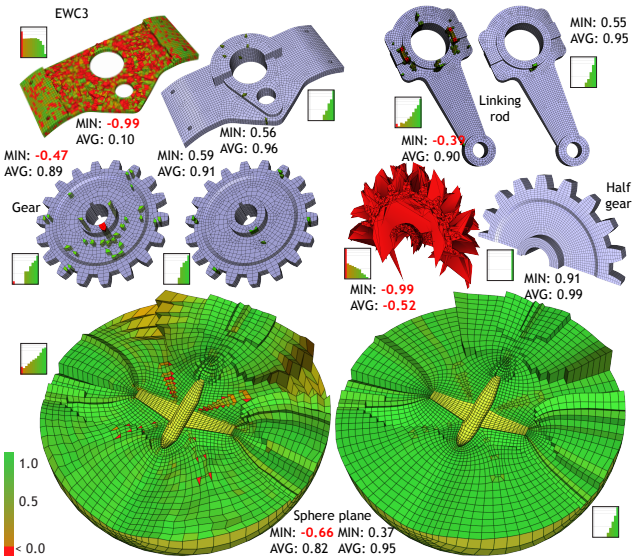


Figure 8: *Optimizing inputs from Ruiz-Girones et al. [2014a; 2014b]. We improve both the worst and average element quality, increasing the lowest element MSJ by at least 0.1.*

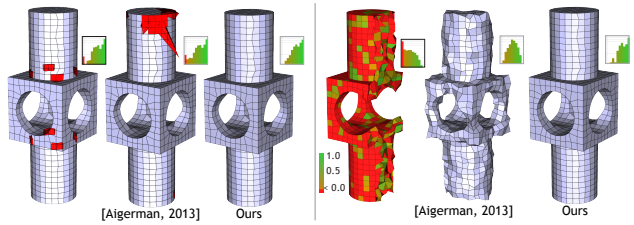


Figure 9: *Employing the tetrahedral mesh optimizer of [Aigerman and Lipman 2013]) on the overlapping tetrahedral decomposition of an input hex mesh fails to produce satisfactory results. With the surface vertices fixed (left) the method fails to untangle even simple inputs, (right) with the surface relaxed the output surfaces may deviate too far from the input. Our method successfully handles both inputs, minimally deviating from the input.*

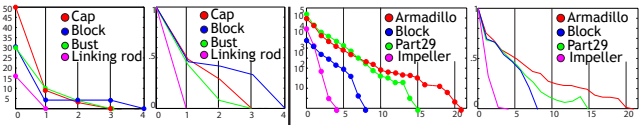


Figure 10: *Typical convergence behavior on standard (left) and artificially corrupted (right) inputs. For each we plot both the number of inverted elements (left) and the energy in Eq. 25 (right). X axis: iterations. Y axis, left to right: number of inverted elements, log of energy (normalized); log of inverted element count; log of energy.*

inputs the method typically converges in under five iterations, and requires up to 20 or 25 iterations for the corrupted inputs. Our runtimes (Table 1) (measured on an Intel i7-4770K CPU with 16 gigabytes of RAM), depend on both the size of the input and the number of *a priori* inverted elements. Runtimes range from 2 seconds for the block (Figure 9), which has 2.5K elements, 30 of them inverted, to 4 minutes for the dragon (Figure 6) which has 14K elements, 84 of them inverted. For artificially corrupted models, our runtimes range from 11 seconds for the Hanger, which has 4.5K elements, 4K of them inverted, to just under 13 minutes for the asm001 which has 25K elements, 24K of which are inverted.

Comparisons. We provide both direct and indirect comparisons to existing optimization methods. As shown by Table 3 our method consistently generates higher quality (both lowest and average) meshes than the popular Mesquite software [Brewer et al. 2003] based on the methods of [Knupp 2001; Knupp 2003]. Since Mesquite has two boundary preservation options - hard constraints, or none - we tested both settings. With either setting, their method fails to untangle many input meshes that we succeed on, such as the armadillo in Figures 1 and 4. We also tested our method on a range of inputs used in [Ruiz-Gironés et al. 2014b; Ruiz-Gironés et al. 2014a] (Gear, EWC3, Linking and sphere plane) and, at our request, had the authors of these papers run their methods on inputs we provided. As the numbers show, our method consistently outperform these methods, and is able to untangle inputs on which they fail (Figure 4,bottom). While this is partly attributable to their implementation choice to keep all surface vertex positions fixed, one can also observe areas on the input where the method fails to untangle elements despite the surface mesh being high quality (see the leg of the armadillo in Figure 4, e).

Model	# hexa	Pre neg. el.	Pre avg / min	Post avg / min	avg. dist.
EWC3					
[Ruiz-Gironés'14a]	11K	4.5K	.10 / -.99	.93 / .45	0
[Brewer'03]	-	-	-	.94 / -.03	0.02
(ours)	-	-	-	**.96 / .56**	1.3×10^{-5}
Gear					
[Ruiz-Gironés'14a]	4K	2	.89 / -.47	.89 / .46	0
[Brewer et al. 03]	-	-	-	.88 / .37	0
(ours)	-	-	-	**.91 / .59**	7.2×10^{-6}
Linking rod					
[Ruiz-Gironés'14a]	11K	16	.90 / -.39	.92 / .41	0
[Brewer et al. 03]	-	-	-	.92 / .26	0
(ours)	-	-	-	**.95 / .55**	5.1×10^{-6}
Sphere plane					
[Ruiz-Gironés'14a]	4K	147	.82 / -.66	.90 / .20	0
(ours)	-	-	-	**.95 / .37**	5.4×10^{-6}
Half gear					
[Ruiz-Gironés'14b]	16K	15K	-.52 / -.99	.98 / .70	N/A
(ours)	-	-	-	**.99 / .91**	5.3×10^{-4}
Block					
[Aigerman'13]	-	-	-	.79 / -.98	0
(ours)	2.5K	31	.77 / -.70	**.87 / .25**	6.5×10^{-5}
Block (stress)					
[Aigerman'13]	2.5K	2.3K	-.51 / -.99	.50 / .07	0.01
(ours)	-	-	-	**.83 / .25**	7.8×10^{-5}
Armadillo					
[Ruiz-Gironés'14a]	30K	323	.69 / -.76	.80 / -.74	0
[Brewer'03]	-	-	-	.91 / -.99	0
[Brewer'03]	-	-	-	.79 / -.99	0.02
(ours)	-	-	-	**.90 / .14**	7.2×10^{-4}

Table 3: *Numerical comparisons with [Ruiz-Gironés et al. 2014a; Ruiz-Gironés et al. 2014b; Brewer et al. 2003; Aigerman and Lipman 2013]. Our method successfully untangles inputs on which previous methods fail and improves the lowest element quality significantly on others.*

Most raw hex-meshing outputs contain degenerate surface elements and surface vertex locations which, if unchanged, prevent interior improvement (see the chest of the Armadillo in Figure 1). While methods such as [Ruiz-Gironés et al. 2014b] keep surface vertex positions fixed, methods that aim to process real data must support vertex movement while minimizing deviation from the input surface. Our framework provides this trade-off, allowing vertices to move tangentially, indirectly restricting Hausdorff distance. We experimented with differerent values for the surface attraction α and boundary attraction β on the block model, which has a valid surface mesh and where hex mesh quality improvement is bounded by how much our method is allowed to deviate from the circular shape of the cylinder base. With $\alpha = \beta = 100.0$, we produce an inversion-free mesh with MSJ 0.12 and average distance 2.3×10^{-5}; with $\alpha = \beta = 5000.0$, we produce an inversion free mesh with MSJ

0.03 and average distance $2.0 \times 10^{-}5$.

We indirectly compare our method to those of Paille et al [2013] and Marechal [2009], as both frameworks intertwine connectivity and positional optimization, preventing direct comparisons. While Paille et al [2013] successfully generate inversion-free high-order meshes, the finest-level linear meshes they produce may, as demonstrated by the KingKong model, contain inverted elements. We successfully untangle this model generating a high quality mesh.

We provide a two-fold comparison to the method of Marechal [2009] that untangles meshes generated using the octree-based technique by selectively deviating from the input surface, but, as the author notes, it generates meshes with very low minimal quality (lowest MSJ under 0.1, see Table 1). Our method is able to significantly improve the quality of these meshes (asm001, asm106, clef2) with only negligible deviation from the input surface. It is also able to untangle and improve randomized versions of these meshes which contain numerous inverted elements (Table 2, Figure 7) .

An alternative hex mesh optimization strategy is to treat the mesh as a union of overlapping tetrahedra positioned at hex element corners, then attempting to optimize the hex mesh quality using state of the art frameworks for tetrahedral optimization, e.g. [Aigerman and Lipman 2013]. This paper provides two frameworks - one for direct tet mesh quality optimization and one for optimization of a mapping between two tet mesh domains; and has two sets of boundary conditions - keeping all surface vertices fixed or allowing them to deviate freely. The direct framework fails to correct inverted elements on all the inputs we tested. The mapping-based framework can only be used if an inversion-free mesh with the same connectivity is available, which is the case for PolyCube based hex-meshes, but not for other inputs. As demonstrated in (Figure 9, top) with all boundary vertices fixed, the method fails to untangle even simple inputs. For this simple example, we use the parameter $K = 10$ as their paper suggests. With the boundary free to move the method is able to untangle the inputs we tried, but often drastically deviates from the input surface (Figure 9, bottom), making it unsuitable for simulation needs. Our method closely preserves vertex positions while untangling the meshes in both examples.

Design Alternatives. We validate our key design choices in a number of ways. We experimented with removing the regularization energy $E_{regularize}$ from our global minimization step, and found that 22% of our meshes failed to converge to valid solutions. Similarly, we attempted to use only the regularization energy $E_{regularize}$ for our global minimization step - effectively removing our tetrahedral cone alignment, and only relying on parallel and consecutive edge alignment to attempt to improve mesh quality. When we did this, 66% of regular input meshes and all the corrupted ones failed to converge to valid solutions. To highlight the necessity of our relaxed iterative optimization framework, we note that two thirds of our regular meshes tested, and all corrupted meshes tested, experienced at least one failure to find a valid target edge direction $\hat{n}_{ij}$ (Equation 9) before successfully improving; these meshes would fail to converge to a valid solution using the strict global formulation of Section 3.1.1.

Limitations. Our local/global framework offers no theoretical guarantee that it will return an inversion free solution: despite recent progress [Erickson 2014], the question of whether or not a given hex mesh connectivity has an inversion free embedding remains open, and multiple initial mesh configurations exist for which no inversion free solution is possible when boundary vertices are fixed (Figure 11). In our experiments, however, we are able to generate inversion free outputs even from heavily corrupted inputs, if and when the boundary preservation constraints were sufficiently relaxed. In practice, our method can fail on models with extreme grading, as massive differences (a factor of 32 or more) between the current and target edge lengths can cause ill-conditioned matrices. We can also fail when presented with unreasonable target edge lengths, for instance when randomly and dramatically displacing all interior vertices by over 400% of the average interior edge length, and then using the distorted lengths as optimization targets.

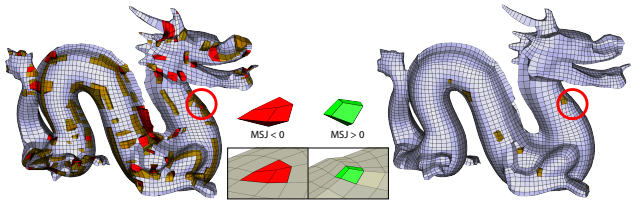


Figure 11: *The initial dragon mesh contains hexahedra with three surface faces. In the zoomed-in mesh configuration no inversion-free solution is possible without slightly relaxing the boundary vertices.*

5 Conclusions

We have presented a novel method for improving the quality of hexahedral meshes. Our entire framework is compactly described by the 11 lines of pseudo-code in Algorithm 1. We demonstrate a significant improvement on the state-of-the-art in terms of both worst and average hex element quality, and untangle poor quality meshes on which previous approaches fail.

Our approach was inspired by local-global optimization schemes (such as [Aigerman and Lipman 2013] and [Sorkine and Alexa 2007]) that repeatedly optimize local cells, then combine local solutions into a global one. In selecting cell resolution one must allow enough degrees of freedom to avoid overconstrained local solutions, yet take adjacent geometry into account for local solutions to be as compatible as possible in the global stage. For hex meshes, our edge-cone cell decomposition provides this balance, and has a clear easily computed local optimum (axis maximally orthogonal to base). The alternative of using the eight corner tets, or entire hexes, leads to incompatible local solutions; while optimizing individual vertex positions requires complex Newton-type optimization and often leads to overconstrained situations. Intuitively, we see hex-mesh cones as the counterpart of the overlapping umbrella cells on surfaces in ARAP deformation. By recasting the problem of hex mesh quality improvement as an optimization problem on the cone of tetrahedra surrounding every directed edge of the hex mesh, we are able to convert a complex constrained non-linear optimization problem into a series of tractable, simple, convex optimizations. Our choice, validated by our extensive reported results, is the core innovation of our paper. We hope the theory behind it will interest future researchers.

Future Work. Practical improvements we would like to investigate include optimization speedup, through the use parallel or multigrid solvers, and increased numerical stability in the presence of heavily graded meshes. Our work also motivates several interesting theoretical questions: our convergence rate as well as the robustness of the alternative methods we tested appear to be directly linked to the initial surface mesh configuration and the degree of input mesh regularity, raising the question of how to quantify ”easy” or ”hard” to optimize initial meshes.

Acknowledgments

The authors thank NSERC and GRAND NCE for their ongoing financial support and M. Bessmeltsev for help with figures. We thank AIM@SHAPE-VISIONAIR (bust, dancing children) and Stanford (bunny, armadillo, dragon) for models.

References

AIGERMAN, N., AND LIPMAN, Y. 2013. Injective and bounded distortion mappings in 3d. *ACM Trans. Graph. 32*, 4.

BLACKER, T. 2000. Meeting the challenge for automated conformal hexahedral meshing. In *Proc. International Meshing Roundtable*.

BREWER, M. L., DIACHIN, L. F., KNUPP, P. M., LEURENT, T., AND MELANDER, D. J. 2003. The mesquite mesh quality improvement toolkit. In *Proc. International Meshing Roundtable*.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: Measuring Error on Simplified Surfaces. *Comput. Graph. Forum 17*, 2.

ERICKSON, J. 2014. Efficiently hex-meshing things with topology. *Discrete and Computational Geometry 52*, 3, 427–449.

FREITAG DIACHIN, L., KNUPP, P., MUNSON, T., AND SHONTZ, S. 2006. A comparison of two optimization methods for mesh quality improvement. *Engineering with Computers 22*, 2, 61–74.

FREY, P. J., AND GEORGE, P. 2007. *Mesh Generation: Application to Finite Elements*.

GREGSON, J., SHEFFER, A., AND ZHANG, E. 2011. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum (Proc. SGP 2011)*.

GUROBI OPTIMIZATION, 2013. http://www.gurobi.com/.

HUANG, J., JIANG, T., SHI, Z., TONG, Y., BAO, H., AND DESBRUN, M. 2014. L_1 based construction of polycube maps from complex shapes. *ACM Trans. Graph. 33*, 3 (June), 25:1–25:11.

KNUPP, P. M. 2001. Hexahedral and tetrahedral mesh untangling. *Engineering with Computers 17*, 3, 261–268.

KNUPP, P. M. 2003. A method for hexahedral mesh shape optimization. *Intl. Journal for Numerical Methods in Engineering 58*, 2, 319–332.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*.

LI, Y., LIU, Y., XU, W., WANG, W., AND GUO, B. 2012. All-hex meshing using singularity-restricted field. *ACM Trans. Graph. 31*, 6.

LIVESU, M., VINING, N., SHEFFER, A., GREGSON, J., AND SCATENI, R. 2013. Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Trans. Graph. 32*, 6.

MARECHAL, L. 2009. Advances in octree-based all-hexahedral mesh generation: handling sharp features. In *Proc. International Meshing Roundtable*.

MIYOSHI, K., AND BLACKER, T. 2000. Hexahedral mesh generation using multi-axis cooper algorithm. In *Proc. International Meshing Roundtable*, 89–97.

NIESER, M., REITEBUCH, U., AND POLTHIER, K. 2011. CubeCover - Parameterization of 3D Volumes. *Computer Graphics Forum*.

NOCEDAL, J., AND WRIGHT, S. 2006. *Numerical Optimization*. Springer-Verlag, New York.

OWEN, S. 2009. A survey of unstructured mesh generation technology. http://www.andrew.cmu.edu/user/sowen/survey/hexsurv.html.

PAILLÉ, G.-P., POULIN, P., AND LÉVY, B. 2013. Fitting polynomial volumes to surface meshes with Voronoï squared distance minimization. *Computer Graphics Forum 32*, 5, 103–112.

PÉBAY, P. P., THOMPSON, D., SHEPHERD, J., KNUPP, P., LISLE, C., MAGNOTTA, V. A., AND GROSLAND, N. M. 2007. New applications of the verdict library for standardized mesh verification pre, post, and end-to-end processing. In *Proc. International Meshing Roundtable*. 535–552.

RUIZ-GIRONÉS, E., ROCA, X., SARRATE, J., MONTENEGRO, R., AND ESCOBAR, J. 2014. Simultaneous untangling and smoothing of quadrilateral and hexahedral meshes using an object-oriented framework. *Advances in Engineering Software*.

RUIZ-GIRONÉS, E., ROCA, X., AND SARRATE, J. 2014. Optimizing mesh distortion by hierarchical iteration relocation of the nodes on the CAD entities. In *Proc. International Meshing Roundtable*.

SASTRY, S. P., AND SHONTZ, S. M. 2009. A comparison of gradient-and hessian-based optimization methods for tetrahedral mesh quality improvement. In *Proc. International Meshing Roundtable*.

SASTRY, S., AND SHONTZ, S. 2014. A parallel log-barrier method for mesh quality improvement and untangling. *Engineering with Computers 30*, 4, 503–515.

SCHNEIDERS, R. 1996. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers 12*, 168–177.

SCHÜLLER, C., KAVAN, L., PANOZZO, D., AND SORKINE-HORNUNG, O. 2013. Locally injective mappings. *Computer Graphics Forum (Proc. SGP) 32*, 5, 125–135.

SHEPHERD, J. F., AND JOHNSON, C. R. 2008. Hexahedral mesh generation constraints. *Eng. with Computers 24*, 3, 195–213.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. SGP*, 109–116.

SUN, L., ZHAO, G., AND MA, X. 2012. Quality improvement methods for hexahedral element meshes adaptively generated using grid-based algorithm. *Intl. Journal for Numerical Methods in Engineering 89*, 6, 726–761.

TAUTGES, T. J., BLACKER, T., AND MITCHELL, S. A. 1996. The whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes. *Intl. Journal for Numerical Methods in Engineering 39*, 19, 3327–3349.

VARTZIOTIS, D., AND HIMPEL, B. 2014. Efficient and global optimization-based smoothing methods for mixed-volume meshes. In *Proc. International Meshing Roundtable*. 293–311.

VARTZIOTIS, D., AND PAPADRAKAKIS, M. 2013. Improved GETMe by adaptive mesh smoothing. *Computer Assisted Methods in Engineering and Science*, 20, 55–71.

WILSON, T., SARRATE, J., ROCA, X., MONTENEGRO, R., AND ESCOBAR, J. 2012. Untangling and smoothing of quadrilateral and hexahedral meshes. In *Eighth Intl. Conference on Engineering Computational Technology*.

WILSON, T., 2011. Simultaneous untangling and smoothing of hexahedral meshes.

ZHANG, Y., BAJAJ, C., AND XU, G. 2009. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Communications in Numerical Methods in Engineering 25*, 1, 1–18.