

Integer-Sheet-Pump Quantization for Hexahedral Meshing


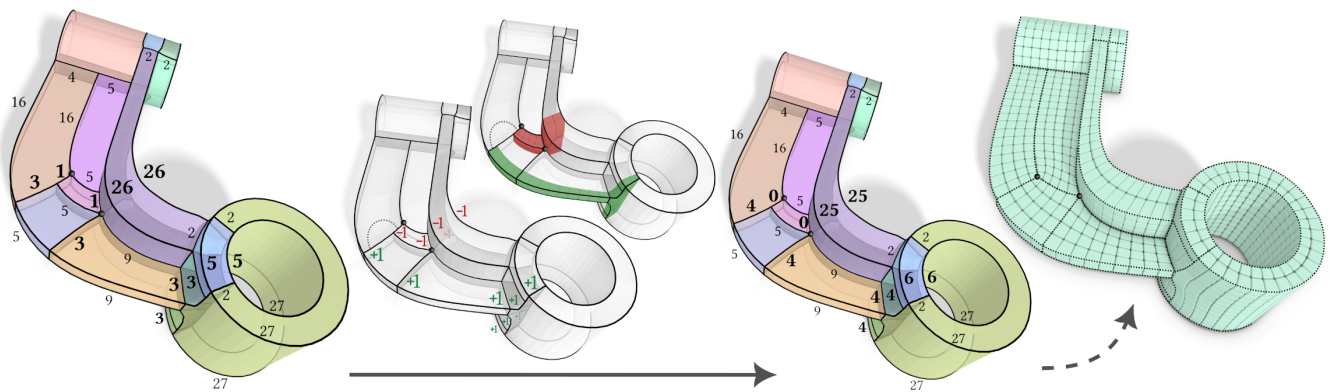
H. Brückler¹  D. Bommes²  M. Campen¹ ¹Osnabrück University, Germany²University of Bern, Switzerland

Figure 1: Our method solves the quantization problem of hexahedral meshing – deciding about the numbers of hexahedra between critical elements – in a very quick yet high-quality manner. Its central ingredient are integer-sheet inflation/deflation operators (center), executed on a cubical cell decomposition (also called T-mesh, left) of the object to be meshed, before actually materializing the hexahedral mesh (right).

Abstract

Several state-of-the-art algorithms for semi-structured hexahedral meshing involve a so called quantization step to decide on the integer DoFs of the meshing problem, corresponding to the number of hexahedral elements to embed into certain regions of the domain. Existing reliable methods for quantization are based on solving a sequence of integer quadratic programs (IQP). Solving these in a timely and predictable manner with general-purpose solvers is a challenge, even more so in the open-source field. We present here an alternative robust and efficient quantization scheme that is instead based on solving a series of continuous linear programs (LP), for which solver availability and efficiency are not an issue. In our formulation, such LPs are used to determine where inflation or deflation of virtual hexahedral sheets are favorable. We compare our method to two implementations of the former IQP formulation (using a commercial and an open-source MIP solver, respectively), finding that (a) the solutions found by our method are near-optimal or optimal in most cases, (b) these solutions are found within a much more predictable time frame, and (c) the state of the art run time is outperformed, in the case of using the open-source solver by orders of magnitude.

Keywords: T-mesh, hexahedral mesh, volume mesh, integer optimization, block decomposition, base complex, hex sheet

1. Introduction

Hexahedral mesh generation via global volumetric parametrization was shown to be a powerful and expressive approach, enabling high flexibility. With recent advances also several of the remaining robustness gaps have been closed. This is discussed further in Sec. 2.

An important component of this approach is that of *quantization*: State-of-the-art methods initially operate with relaxed continuous problems (smooth aligned frame-field generation, field-guided global parametrization), but at some point discrete integer decisions need to be made – due to the fact that the desired hexahedral mesh

has a discrete structure. The process of making these decisions (as well as the resulting integer choices themselves) are referred to as quantization.

The most recent method to reliably compute a quantization in the general volumetric context [BBC22] builds an auxiliary structure based on which this problem can be formulated efficiently: a cell complex forming a partition of the to-be-meshed object into parametrization-aligned blocks, as illustrated in Fig. 2. It may be non-conforming, containing T-junctions, and is also called T-mesh. The integer degrees of freedom are associated with the edges of this structure, subject to a number of constraints. Finding the optimal

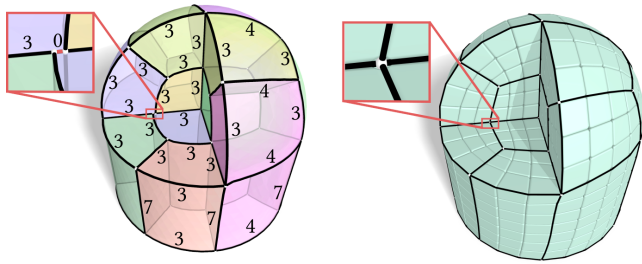


Figure 2: Left: Volumetric cubical cell decomposition of an example object. Concretely, this is the motorcycle complex, computed based on a global seamless map. This is a non-conforming complex with T-junctions (see blow-up). The integers denote a quantization expressed on this structure. Right: Hexahedral mesh implied by this quantization. Its (conforming) base complex is indicated in black.

(or just a valid) assignment of integer values is then delegated to a general-purpose Integer Quadratic Programming solver.

We observe that a commercial solver performs well on this task in the average case. However, there are also outliers where run time is quite impractically high. Furthermore, the required time is very unpredictable, as there is limited correlation between problem size and required time. For practical purposes, therefore additional early-stopping criteria for the solver need to be set (a time limit and/or optimality gap threshold), and choosing these can be challenging. Even more importantly, we observe that open source solvers show significantly lower performance on this task also in the average case.

1.1. Contribution

We propose a special-purpose strategy to efficiently compute near-optimal guaranteed-valid quantizations. Effectively: *A minor amount of quality is traded for a major gain in speed and predictability, while retaining full reliability.*

Key to this is that our method does not require a complex integer quadratic programming (IQP) solver, but relies only on much simpler continuous linear programs (LPs).

It builds on the same auxiliary T-mesh structure as the above state-of-the-art method. But the task is not delegated in its entirety to an external problem-agnostic solver. Instead, we exploit the geometric nature of the problem, devising operators that enable incremental transitions between consistent quantizations. These are applied in a strategic sequence so as to greedily work towards an optimal solution. Inspiration for this stems from related work addressing the generation of surface quad meshes [CBK15].

The benefit is a significantly reduced and more predictable run time, using only open source components, at a often marginal cost in terms of quantization quality.

1.2. Method Overview

Our method’s larger context is the hexahedral mesh generation pipeline as used in several recent works [BBC22, LB23, BC23]. It can serve as a drop-in replacement for the quantization computation step of this pipeline. Its input and output thus are as follows.

Input: A decomposition of a volumetric domain into cuboidal blocks, possibly with T-junctions, for instance derived from a field-guided parametrization as described by Liu et al. [LB23, §6.1]. This decomposition can be interpreted as a non-conforming cell complex, whose cells we will call *blocks*, its facets *patches*, its edges *arcs* and its vertices *nodes*. Some patches, arcs, and nodes may be marked as features, indicating that the intended hexahedral mesh is supposed to be aligned with these. Each arc has an assigned (continuous) target length, commonly derived from its parametric length.

Output: An assignment of integer lengths to the arcs, such that the difference between these integer lengths and the target lengths is low under some measure. This integer assignment can be interpreted as the number of hexahedral layers (or *sheets*) later conceptually passing through each arc as depicted in Fig. 2. The assignment is globally *consistent*, meaning the number of sheets passing through the opposite sides of a block are equal (i.e. any sheet going "into" the block at one end comes "out" of it at its other end), and *separating*, meaning all features (including singular edges and vertices) are pairwise separated by at least one sheet passing between them. See Fig. 3 for an illustration.

Procedure: First, we initialize the quantization by assigning zero length to all arcs. This solution is trivially consistent, but of extremely low quality and, more importantly, not separating.

Our *integer-sheet-pump* (ISP) then repeatedly performs minimal consistency-preserving modifications of this initial quantization. These modifications, termed *integer-sheet inflations/deflations*, can roughly be thought of as virtually inserting or collapsing sheets of hexahedra in the (not yet existing) hexahedral mesh (cf. Fig. 1). This is only a partial interpretation, though, as our integer-sheets more generally correspond to *linear combinations* of hexahedral sheets (cf. Fig. 5). Algebraically, an integer-sheet is a set of integer increments/decrements for the arc lengths, that when applied preserves quantization consistency. We iteratively apply the most favorable inflation/deflation based on how much closer this brings the consistent integer arc lengths to their (continuous) target lengths. This is repeated until no further favorable sheet operation can be found.

Because this quantization, while consistent and now of high quality, might still be nonseparating, additional inflations follow, driven by the goal of feature separation, until this is achieved.

Using the resulting valid (i.e. consistent and separating) quantization, the above mentioned pipeline can proceed to generate a corresponding hexahedral mesh.

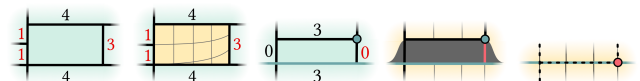


Figure 3: Left: inconsistent quantization, due to non-matching left and right patch sides. Center: non-separating quantization, the feature node and feature arc (both teal colored) have a distance of zero under the assigned integer lengths. This conceptually implies a hole in the mesh or a collapse of the features (right).

2. Related Work

A variety of approaches towards automatic hexahedral mesh generation have been explored, as discussed in a number of surveys [PCS*22, SRRGRN14, SJ08, Tau01, Bla00]. Methods based on global parametrization, especially on so-called integer-grid maps, are associated with particularly high flexibility and achievable quality, but also with challenges in robustness. Recent advances have pushed also their robustness to higher levels [LB23, BC23]. We focus on these parametrization-based methods in the following and refer to the above surveys for a broader overview.

Integer-Grid Maps The general algorithmic pipeline for hexahedral mesh generation based on integer-grid maps [NRP11] is based on ideas from the simpler 2D case of quadrilateral mesh generation [KNP07, BZK09, BCE*13]. An up-to-date overview is given in recent works by Brückler et al. [BC23, BBC22].

This algorithmic pipeline is based on the following principle: Compute a global *seamless* parametrization [NRP11, CSZZ19, MC19, Lev21], typically guided by a smooth frame field [RSL16, SVB17, LZC*18, CC19, PBS20, ZVC*20, LB23], and based on this then make all the necessary discrete decisions to turn it into an integer-grid map – which then implies a hexahedral mesh [LBK16].

The process of making the integer decisions is referred to as quantization. After initial efficient but fragile rounding-based attempts [NRP11, JHW*14, LLX*12], reliable solutions for this problem have recently been presented [BBC22, BC23].

Cell Decompositions The above reliable solutions are based on structured cell decompositions of the volume (also called T-meshes). They can be automatically constructed, for instance, by means of the motorcycle complex [BGMC22], aligned with the given seamless parametrization. Fig. 2 shows such a volumetric cell decomposition of an example object. The way it is constructed, it serves as a convenient auxiliary structure on which the integer degrees of freedom can be expressed and validity criteria can be formulated efficiently. Our method operates on this basis as well.

Distantly related are other types of cell decompositions that have been used in conjunction with hexahedral meshes, such as the base complex [GDC15, Tak19], multi-block decompositions [LPP*20], and variations of the motorcycle complex [GLA23]. In the 2D quadrilateral mesh setting, a decomposition structure called motorcycle graph [EGKT08] has been used for quantization purposes.

Quantization In the 2D case, targeting quadrilateral mesh generation, a number of different strategies have been described to reliably compute quantizations based on cell decompositions, mostly T-meshes constructed according to the motorcycle graph principle. Concretely, there are formulations as Integer Programs [LCK21a, LCK21b], as Minimum Flow problems [HWP23], or as series of Shortest Path problems [CBK15, LCBK19, CODH*24].

In the 3D setting, targeting hexahedral mesh generation, the only reliable solution so far is via an Integer Quadratic Program formulation [BBC22, BC23]. The special case of meshes without internal singularities (so-called polycube meshes) was addressed separately [PRR*22, CLS16]. As we demonstrate in Sec. 5, the IQP formulation in the 3D case is rather slow when using a state-of-the-art open

source solver. It is faster with commercial solvers, but then shows problematic worst-case behavior, leading to highly unpredictable time requirements and making it hard to set stopping criteria to yield at least near-optimal solutions. Our method offers a replacement for this, requiring only a simple LP solver.

The related problem of interval assignment has been addressed in a non-IQP manner using an incremental approach [Mit23] based on a heuristic analysis of the algebraic constraint system. Feasibility of the resulting integer assignment, however, cannot be guaranteed.

We make use of operations that, at the lowest level, can be interpreted as inserting or removing sheets of hexahedra, albeit in a virtual or implicit manner, before the hexahedral mesh even exists. On a conceptual level these have some relation to explicit sheet operators that are used in mesh postprocessing methods [SGW21, WGZC18, GPW*17, SDW*10, SSL10, PBSB08].

3. Background

We adopt the notation of [BC23], viewing the given T-mesh decomposition as cell complex $\mathcal{T} = B \cup P \cup A \cup N$ consisting of blocks (3-cells) $B = \{b_1, b_2, \dots\}$, patches (2-cells) $P = \{p_1, p_2, \dots\}$, arcs (1-cells) $A = \{a_1, a_2, \dots\}$, and nodes (0-cells) $N = \{n_1, n_2, \dots\}$. There may be subsets of nodes $\hat{N} \subset N$, arcs $\hat{A} \subset A$ and patches $\hat{P} \subset P$ marked as features. As in previous work, we assume the singularity structure encoded in \mathcal{T} is supposed to be preserved (i.e. should become the extraordinary edges in the hexahedral mesh), as is the boundary. Hence all boundary patches and all singular arcs are considered features as well. Let us remark that a few other approaches – in the 2D/surface setting – have also considered settings where dynamic alteration by addition [PNA*21] or removal [LCK21b] of singularities as part of the quantization process is allowed.

Via a map $\ell: A \rightarrow \mathbb{R}, a_i \mapsto \ell_i$ a target length ℓ is assigned to each arc. Via another map $q: A \rightarrow \mathbb{Z}, a_i \mapsto q_i$ the sought quantized integer lengths q are represented. As arcs can be indexed from 1 to $|A|$, both mappings can be represented as $|A|$ -dimensional vectors $\ell = (\ell_1, \ell_2, \dots, \ell_{|A|})$ and $q = (q_1, q_2, \dots, q_{|A|})$.

The goal of quantization in general is to minimize the difference between ℓ and q with respect to some distance function $d(\ell, q)$, while keeping the quantization *consistent* and *separating*. In a *consistent* quantization, the lengths of opposite patch sides are equal. This intuitively corresponds to equally many quad strips (or hex sheets) passing into and out of the patch. This is easily modelled via a system of equality constraints $\mathcal{A}q = \mathbf{0}$, with the consistency matrix \mathcal{A} containing entries from $\{-1, 0, 1\}$. Note that this per-patch length equality implies consistency also on the level of blocks.

In a *separating* quantization, all features are properly separated by at least one implied hex sheet passing between them, i.e. there is a non-zero distance in terms of the assigned integer arc lengths q . This can be modelled with inequality constraints $\mathcal{B}q \geq \mathbf{1}$ [BBC22], with the separation matrix \mathcal{B} having entries from $\{-1, 0, 1\}$ and $\mathbf{1}$ being a vector of ones. In essence, each such constraint sums the assigned integer arc lengths q along arc-paths between features, and thereby requires a non-zero distance in terms of quantized lengths.

Lastly, to prevent local inversions in the quantization, variables q can either be given a lower bound of 0 (as was done in previous

work on the 2D case [CBK15]) or, providing more freedom, three inequality constraints per block can be introduced to enforce non-negative block dimensions (as was done in previous work on the 3D case [BBC22]). In either case, this gives an additional $Cq \geq \mathbf{0}$ with the non-negativity matrix C being either the identity matrix or a matrix with $3|B|$ rows and entries from $\{0, 1\}$.

This yields the general *quantization problem*

$$\min d(q, \ell) \quad (1a)$$

$$\text{s.t. } \mathcal{A}q = \mathbf{0} \quad (\text{consistency}) \quad (1b)$$

$$\mathcal{B}q \geq \mathbf{1} \quad (\text{separation}) \quad (1c)$$

$$Cq \geq \mathbf{0} \quad (\text{non-negativity}). \quad (1d)$$

In previous work [BBC22, LB23, BC23] eq. (1) is solved via a sequence of IQPs, with the objective function set to the quadratic expression $d(q, \ell) = \sum_{a_i \in \mathcal{A}} |q_i - \ell_i|^2$. This is done as follows:

- 1) Set up \mathcal{A} and C , initialize \mathcal{B} empty (or with trivial constraints).
- 2) Solve (1) using a general-purpose solver for IQPs.
- 3) Search the arc graph for unseparated features. If any: append corresponding separation constraints to \mathcal{B} , goto step 2).

The separation matrix \mathcal{B} is built lazily for efficiency, always adding only the tightest violated constraints, because a very small subset proves to often already be sufficient to achieve full separation.

4. Greedy Incremental Quantization

Our central goal is replacing step 2) in the above algorithm with a tailored efficient alternative, not relying on IQPs. We design an algorithm that initializes the quantization to satisfy (1b)+(1d), then greedily works towards minimizing (1a) while preserving (1b)+(1d) and establishing satisfaction of (1c) in the process.

4.1. Concept

Starting from an initial trivial quantization $q^0 = \mathbf{0}$ we obviously have $\mathcal{A}q^0 = \mathbf{0}$ and $Cq^0 \geq \mathbf{0}$. We then wish to perform incremental additive integer-valued updates Δq preserving consistency, i.e. $\mathcal{A}\Delta q = \mathbf{0}$. We reject potential updates that would violate $C(q + \Delta q) \geq \mathbf{0}$. Regarding separation, however, we will need not only separation-preserving updates Δq , but also separation-establishing updates that enable incremental reduction of the number of separation violations.

Ideally we would like to find an update Δq that respects all these requirements while directly leading to the globally optimal solution, i.e. minimizing (1a). But this, of course, is just as hard as solving (1). Hence we instead commit to a greedy search approach. Among locally minimal updates q that are consistency- and nonnegativity-preserving, we select one that improves the objective (1a) as much as possible, or one that satisfies an additional constraint from (1c) while deteriorating the objective as little as possible. Choosing such small updates as the steps of our algorithm has two rationales: They can be constructed efficiently, and they enable fine-grained navigation towards the optimum.

These considerations lead to the high-level view of our algorithm outlined in Alg. 1. After initialization of q , the outermost loop alternately repeats the inner optimization loop and separation step until global separation is achieved. The inner optimization loop

goes through all arcs and tries to find minimal updates bringing arcs closer to their target length, starting with the arcs furthest away from their target length. In this way the arcs that most negatively impact the quadratic objective function are tackled first. OPTIMAL-INTEGERSSTEP determines the optimal multiplicity $t \in \mathbb{N}$ for the determined minimal consistent update $\Delta q: \arg \min_t |q + t\Delta q - \ell|^2$, rounded to the closest constraint-feasible integer (due to convexity). This is merely for efficiency; one could always use $t = 1$. If the found update improves the objective, we apply it and update all affected arc priorities accordingly. Lastly, after convergence of an optimization round we search for violated separation constraints [BBC22], returning if there are none. Otherwise we apply a minimal update that establishes separation for the lazily found violations and then reoptimize.

Both the arc-based and separation-driven updates should fulfill our previously stated requirements: local minimality, preservation of consistency and non-negativity. In the following we will focus on how to efficiently find exactly such updates.

Algorithm 1: INTEGER-SHEET-PUMP QUANTIZATION

```

Input: T-mesh with target lengths  $\ell$ 
Output: Final quantized integer arc lengths  $q$ 
//Initialization//
 $q = \mathbf{0}$ 
set up  $\mathcal{A}$  and  $C$ 
 $\mathcal{B} = \emptyset$ 
repeat
  //Greedy optimization//
  while still improving do
     $Q =$  queue of arcs  $a_i$  ordered by  $|q_i - \ell_i|$ 
    while  $Q$  not empty do
       $a_i = Q.\text{pop}()$ 
       $\Delta q = \text{MINIMALARCUPDATE}(a_i)$  // App. C
       $t = \text{OPTIMALINTEGERSSTEP}(\Delta q, q, \ell)$ 
      if  $|q + t\Delta q - \ell|^2 < |q - \ell|^2$  then
         $q = q + t\Delta q$ 
        foreach nonzero row  $i$  of  $\Delta q$  do
           $Q.\text{update}(a_i)$ 
  //Lazy separation//
   $\mathcal{B}' = \text{LAZYSEPARATIONCONSTRAINTS}(q)$  // [BBC22, Alg.1]
  if  $\mathcal{B}'$  empty then
    return  $q$ 
   $\mathcal{B} = \mathcal{B} \cup \mathcal{B}'$ 
   $q = q + \text{MINIMALSEPARATINGUPDATE}()$  // App. C

```

4.2. Minimal Consistent Updates

Regarding the construction of locally minimal updates, we can take inspiration from related work on the 2D quantization problem for surface quad mesh generation [CBK15] – at least on a conceptual level. In that work, the concept of generating vectors is used to represent minimal quantization updates. In this context, generating vectors are integer vectors Δq representing the smallest consistent building blocks that can form any consistent quantization by incremental addition, when starting from $q^0 = \mathbf{0}$.

The main difficulty lies in determining such favorable generating vectors. Generating vectors Δq fulfilling $\mathcal{A}\Delta q = \mathbf{0}$ turn out to correspond directly to certain dual paths in the 2D/surface T-mesh.

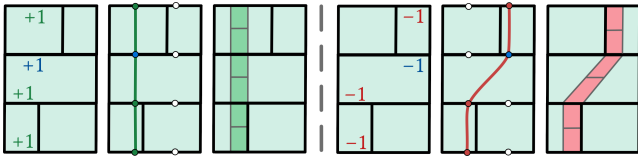


Figure 4: Two examples of positive (green) and negative (red) integer-strips Δq constrained to pass through different root arcs (blue), illustrated as dual paths through the T-mesh and as corresponding (inserted or collapsed) quad strips. Note that not all integer-strips can be simply interpreted as quad strips, see Fig. 5.

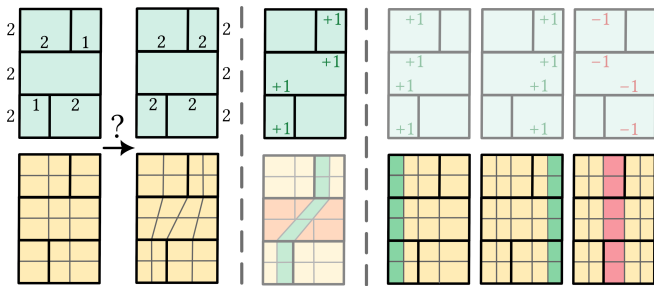


Figure 5: Top: quantizations. Bottom: corresponding quad meshes. The transition shown on the left can be achieved through modification of the quantization by the integer-strip shown top center. Note that it has no corresponding interpretation as a quad strip insertion in the mesh. Instead, it rather corresponds to a combination of multiple quad strip insertions and collapses, as shown on the right, and thus is able to capture more complex transitions.

More specifically, these paths have to always traverse patches end-to-end and have to be cycles or end at the boundary. The crossed arcs have a value of 1 in Δq , all others 0. Fig. 4 shows an example. Due to their path nature, a simple Dijkstra-type algorithm can be employed to find them. To this end, strictly positive weights are assigned to each arc a_i of the decomposition (and their dual arcs) that gauge how much closer or further $\Delta q_i = 1$ would bring that arc towards its target ℓ_i . Using this, a minimum-weight path constrained to pass through at least one root arc a^* will pass through as few (in a weighted sense) as possible other arcs. Hence, the corresponding vector Δq is a greedily chosen, locally minimal consistent update.

4.2.1. Sheet/Strip Metaphor

Intuitively, such a generating vector (or *integer-strip*) can be thought of as inserting a single additional strip of quads in the quad mesh that is structurally implied by the quantization. This is a limited interpretation, though, as these virtual insertions do not translate one-to-one to incremental updates of the implied quad mesh by classical quad strip insertion (or collapse) operations. Integer-strip insertions are more expressive than quad-strip insertions in the quad mesh, in the sense that a single integer-strip insertion would require multiple (and arbitrarily many) operations on the quad strips of a quad mesh, as demonstrated in Fig. 5.

This strip metaphor can be lifted to the volumetric setting: positive updates Δq correspond to the number of additional virtual

sheets crossing the arcs within the block decomposition, as shown in Fig. 1 (center). We call those *integer-sheets* and hence our algorithm, repeatedly inflating or deflating them, an *integer-sheet-pump* (ISP).

4.3. Formulation as a Linear Program

While the determination of integer-strips is equivalent to finding paths in a graph, this concept does not extend to the volumetric setting. Instead, determining an integer-sheet in the cell complex is equivalent to finding a discrete dual *surface* within a polyhedral mesh. This can be phrased as a linear program [Gra08].

Essentially, integer-sheets (and also integer-strips in 2D) rooted at an arc a_j are solutions to a linear program of the following form:

$$\min_{\Delta q \geq 0} \sum_{a_i \in A} \Delta q_i w_i \tag{2a}$$

$$\text{s.t. } \mathcal{A} \Delta q = \mathbf{0} \tag{2b}$$

$$\Delta q_j \geq 1 \tag{2c}$$

with w_i being the previously mentioned unfavorability weights of arcs. These weights (detailed in eq. (4)) represent how unfavorable it is for a strip/sheet to cross a given arc, and the objective function therefore tries to minimize the total unfavorability of the strip/sheet. j is the index of an arc a_j which the strip/sheet is constrained to pass through. Without (2c), $\Delta q = \mathbf{0}$ would be the trivially optimal solution; the constraint effectively pumps up the otherwise flat sheet. In the 2D case of integer-strips, \mathcal{A} has a structure that allows the LP to be easily solved via Dijkstra’s algorithm. Namely, \mathcal{A} has at most two nonzero entries per column – a result of each arc being incident to at most two patches. In 3D arcs are generally incident to an arbitrary number of patches and thus there are commonly more than two entries per column, preventing this simple approach.

4.3.1. Integer Solutions

The above continuous LP always has a rational solution and one such solution will reliably be retrieved by a standard simplex algorithm. This follows from $\mathcal{A}, \mathcal{B}, \mathcal{C}$ as well as the constraint right-hand sides only having integer entries [PS98]. This rational solution turns out to even be an integer solution in the vast majority of cases. Intuitively, this is due to the objective function trying to push all variables down to their lower bounds – which is an integer 1 for the root arc – while the consistency constraints propagate this value across patches. Only due to rather intricate interplay of T-junctions, non-integer but rational solutions occasionally occur.

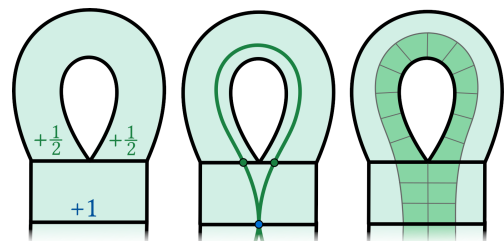


Figure 6: 2D example of a T-mesh where constraining the blue arc to value 1 will yield a consistent solution with half-integers. Multiplication by 2 yields an integer-sheet in this case (right).

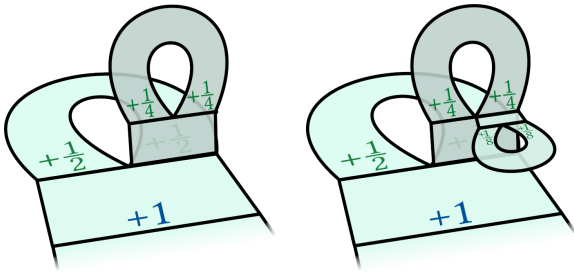


Figure 7: 3D example analogous to Fig. 6. For clarity only some patches are shown, no blocks; imagine an extruded version of this for a complete example. The third dimension allows nesting the U-turn pattern, here illustrating that denominators 2, 4, 8 can result.

In 2D only one structural non-integer configuration exists [CBK15], displayed in Fig. 6. In this case, going from the rational LP solution to the respective integer-strip simply requires multiplication by factor 2. In 3D, by contrast, non-integer rational solutions with more varied denominators can occur. This is illustrated in Fig. 7. By further nesting this structural pattern, examples with arbitrarily high denominators could be constructed, although this is unlikely to occur in practice. In any case, an integer solution can be obtained through multiplication with the *least common denominator*.

Note, however, that such an integer solution obtained through multiplication might not actually be locally minimal anymore. We could resort to solving the LP as an actual ILP (with integer-valued variables) in such cases. However, we determined that the lcd differs from 1 quite rarely, and in these cases it is by far most often just 2; see Fig. 8. It hence appears reasonable to accept this minor local suboptimality for the sake of efficiency and simplicity within the anyway approximative algorithm.

4.3.2. Weights

To determine integer-sheets Δq whose inflation ($q + \Delta q$) is favorable, the weights $w_i > 0$ per arc need to be chosen to reflect unfavorability of arc inflation. To determine integer-sheets whose deflation ($q - \Delta q$) is favorable, they need to be chosen to reflect unfavorability of arc deflation.

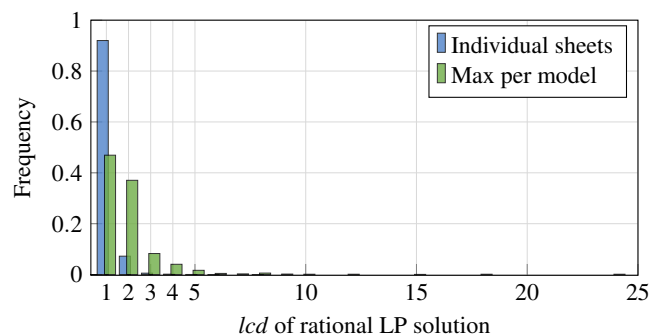


Figure 8: Distribution of least common denominators of all the rational LP solutions computed over all models in our test dataset (cf. Sec. 5). In 92% of the cases it is 1, i.e. no multiplication is required to yield an integer-sheet.

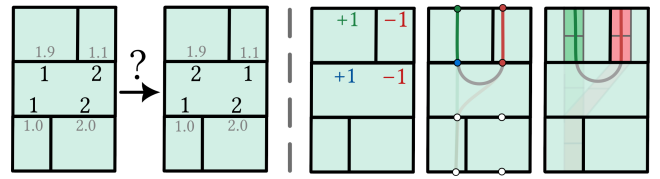


Figure 9: The transition depicted on the left is desired, it brings the integer lengths closer to the target values (grey). Via sequential inflation or deflation of integer-sheets, however, this transition cannot be achieved in a monotonic manner. But a mixed-sign sheet, which can be interpreted as a sum of multiple sheets, here one inflating (green) and one deflating (red), enables the transition atomically.

($-\Delta q$) is favorable, they need to be chosen to reflect unfavorability of arc deflation.

A weighting scheme that fulfills these requirements is the one also used for the 2D Dijkstra formulation [CBK15]. For each arc $a_i \in A$ the weights are built from the signed difference ε_i of each arc's current quantized length from its target length:

$$\varepsilon_i = \begin{cases} \ell_i - q_i, & \text{if inflating} \\ q_i - \ell_i, & \text{if deflating} \end{cases} \quad (3)$$

Based on this, a 3-tier hierarchical weighting is employed:

$$w_i = \begin{cases} 1/(\varepsilon_i + 1), & \text{if } 1 \leq \varepsilon_i \\ |A|/(\varepsilon_i + 1), & \text{if } 0 \leq \varepsilon_i < 1 \\ |A|^2(1 - \varepsilon_i), & \text{if } \varepsilon_i < 0 \end{cases} \quad (4)$$

Arcs in the highest tier would be shifted towards their target length and consequently have very low unfavorability weights between 0 and $1/2$. Arcs in the second tier would be shifted across their target length, so an inflation would be either slightly favorable or slightly unfavorable. These are assigned weights between $|A|$ and $|A|/2$, which by construction is strictly greater than the combined weights in tier 1. The third tier concerns arcs that would be shifted even further away from their target, and are assigned weights at $|A|^2$ and above – larger than the last two tiers combined. This hierarchical weighting ensures that arcs for which a shift in length would be unfavorable will only be shifted if there is no way to shift instead any number of other arcs for which the shift would be favorable.

4.3.3. Mixed-Sign Sheets

So far we have silently assumed the variables in our LP formulation to be non-negative. This corresponds to either an integer-sheet inflation or a deflation (when the negated result is added).

It is not hard to see that there are cases for which both simple sheet inflations and deflations fail to escape local minima of the global objective function $d(q, \ell) = |q - \ell|^2$. One such case in a 2D setting is shown in Fig. 9. Here, the global optimum is not attainable with a greedy algorithm that only considers Δq with non-negative (or non-positive) entries and requires monotonic decrease of $d(q, \ell)$. This is a limitation also affecting the above mentioned 2D quantization algorithm [CBK15].

Instead of compromising the greedy nature of the approach to

overcome this, we suggest here a way of modelling mixed-sign updates $\Delta q = \Delta^+q - \Delta^-q$, i.e. mixed inflation and deflation, with the same formalism via two non-negative vectors Δ^+q, Δ^-q . To this end we extend LP (2) as follows:

$$\min_{\Delta^+q, \Delta^-q \geq 0} \sum_{a_i \in A} \Delta^+q_i w_i^+ + \Delta^-q_i w_i^- \quad (5a)$$

$$\text{s.t. } \mathcal{A}(\Delta^+q - \Delta^-q) = [\mathcal{A} \mid -\mathcal{A}] \begin{bmatrix} \Delta^+q \\ \Delta^-q \end{bmatrix} = \mathbf{0} \quad (5b)$$

$$\begin{cases} \Delta^+q_j \geq 1 \wedge \Delta^-q_j = 0 & \text{if inflating } a_j \\ \Delta^+q_j = 0 \wedge \Delta^-q_j \geq 1 & \text{if deflating } a_j \end{cases} \quad (5c)$$

where w_i^+ and w_i^- are the inflation/deflation weights computed via eqs. (3) and (4), and a_j is an arc constrained to be either inflated or deflated. With this the transition in Fig. 9 can be achieved in a single greedy step.

4.3.4. Enforcing Separation and Non-Negativity

Of our constraints (eqs. (1b)–(1d)) only consistency has been taken into account so far. Incorporating the other two is straightforward through the relation $q = q_{\text{pre}} + \Delta q$ where q_{pre} is the current quantization before applying the update. Using this substitution, separation and non-negativity constraints can be added to the LP as:

$$\mathcal{B}\Delta q \geq \mathbf{1} - \mathcal{B}q_{\text{pre}} \quad (6a)$$

$$\mathcal{C}\Delta q \geq -\mathcal{C}q_{\text{pre}} \quad (6b)$$

These constraints operate in two ways during our greedy optimization. If the current quantization q_{pre} is already separating (and non-negative), then these constraints merely provide bounds for the integer-sheet-pump driven by single arc inflations or deflations (via (5c)). When demanding arc deflation, the LP may even become infeasible, signalling that forced deflation of a certain arc would be incompatible with separation or non-negativity. This mode is used in MINIMALARCUPDATE from Alg. 1. On the other hand, if the current quantization is currently non-separating – which may be the case initially due to separation constraints being added lazily – no arc-driven pumping via (5c) is needed; the sheet is *self-inflating* via (6a). This mode is used in MINIMALSEPARATINGUPDATE from Alg. 1. Both algorithms are listed in App. C for reference.

A challenge that presents itself here are the occasional non-integer, rational solutions to the LP. If one avoids these by re-solving the LP with integer constraints as an ILP, there is no problem. Otherwise, if an integer-sheet is obtained via lcd-multiplication, additional care needs to be taken. The rational quantization update by construction would stay within all bounds but a multiple of it might not. Note, however, that this can only be the case when negative values are involved. Hence, for non-integer updates we encounter that become infeasible through multiplication, we solve restricted LPs instead (with $\Delta^-q = \mathbf{0}$ and negative values suppressed in \mathcal{B}). This yields a solution from a more constrained yet feasible space, which can always be scaled as necessary to obtain an integer solution that establishes separation. Details on this fallback are given in App. B.

4.4. Implementation Considerations

The complete LP formulation, including all described extensions, is stated in App. A. In that formulation, there are two variables

per arc, two equality constraints per patch, and a varying number of inequality constraints. We observe that often not all variables are mutually dependent via constraints. This allows partitioning the problem into multiple independent subproblems, which can be solved more efficiently. To this end, we determine the clusters of variables that are transitively dependent via constraints. For each cluster then a separate LP with those constraints that involve the respective variables is formed. On average this resulted in around 5 subproblems in our experiments.

Furthermore, we can trivially reduce the problem size of these subproblems, by eliminating one variable and one constraint for each equality constraint that involves only two variables, i.e. for each pair of opposite patch sides consisting of only one arc each. This situation occurs quite frequently in the T-meshes studied here: only around 15% of consistency constraints involved more than two variables in our experiments.

5. Results

We have implemented our method as a modification of the IQP based software released by Brückler et al. [BBC22]. Our integer-sheet-pump (ISP) implementation is available in the same code repository, github.com/HendrikBrueckler/QGP3D. It employs CLP from the COIN-OR project [LH03] as an open-source solver for linear programs. Experiments were conducted on a modern 32-core server CPU. While multithreaded performance might be an interesting subject to investigate, our experiments solely focus on single-threaded performance, allocating exactly one thread per experiment instance.

An experiment instance here is an input model equipped with a seamless parametrization (based on which an aligned T-mesh can be computed) and a scaling factor. Target lengths of arcs are given by their length under the seamless parametrization, scaled by aforementioned global factor. This factor effectively controls the resolution of the resulting hexahedral mesh. As input we use the same datasets as employed in [BC23], and per model the T-mesh is computed via the motorcycle complex [BGMC22]. Each model we use in combination with 8 different scaling factors, chosen per model such that 0%, 1%, 5%, 10%, 17%, 25%, 50% or 100% of arcs have a target length below 0.5, respectively, yielding a total of 1568 test instances. Spreading out the scaling factors like this, we aim to cover the full range of potential application scenarios ranging from the generation of maximally coarse block layouts to dense hexahedral meshes, where the optimization problems might behave quite differently.

We compare our ISP method to two implementations of the IQP formulation [BBC22] based on different general purpose solvers as back-ends. Our main point of comparison is using the open-source solver BONMIN [BBC*08]. Additionally, we report timings for using the commercial solver GUROBI [Gur24].

5.1. Quantization Evaluation

Comparison methodology In this part of our evaluation we want to focus on three key aspects to gauge our method's performance against IQP formulations based on general-purpose solvers. Those are execution speed, predictability of run time and optimality of results obtained within certain time frames.

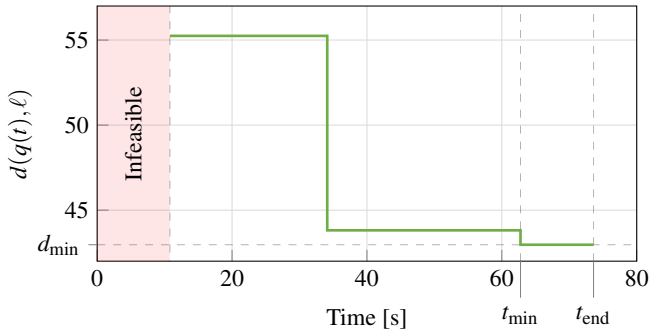


Figure 10: Generic example of solver behavior over time. Branch-and-bound IQP solvers as well as our ISP method are (after an initial phase) anytime algorithms, i.e. can be interrupted to return intermediate results (of increasing quality over time).

All three aspects are not as straightforward to compare between implementations as one may expect. Often, general-purpose integer solvers may find a good or even optimal solution after a moderate amount of time and then spend a disproportionately larger amount of time on closing the optimality gap – not necessarily by improving the solution but by incrementally determining that no better solution exists. In the worst case this involves exploring an exponential amount of promising but eventually worse alternative solutions. When comparing execution times, it is not immediately obvious which time to use for comparison – the one where a (in some sense) good solution was found, the one where the optimal solution was found, or the final time when the solver terminates.

Hence, in the following we will compare not a single time and a single objective value but rather entire time curves of the objective values $d(q(t), \ell)$. An example of such a time curve is shown in Fig. 10. Until a solution that is feasible with respect to eqs. (1b)–(1d) is encountered, the objective value can be considered undefined or – more practically – infinite. The second particular point of interest is the final drop of the curve, i.e. the time of acquisition of the final solution and its objective value, which we will refer to as t_{\min} and d_{\min} , respectively. Lastly, the final termination time we will denote by t_{end} . These curves and their key indicators form the baseline for the following evaluations.

Optimality over time For this evaluation we aim to find each method’s “average time curve” of the objective value, over the entire set of test instances. Because both the time and objective scale may differ by orders of magnitude, we need to normalize both axes. For the objective value we normalize via division by the (quasi-)optimum, the final objective d_{\min} of the commercial IQP solver version (with only a generous time limit of 5h). Because the result will be in the range $[1, \infty)$ we take the inverse before averaging the resulting curves. This inverse then represents a relative optimality between 0 (infeasible) and 1 (quasi-optimal). For the time axis, to make comparison between our method and others clearest, we normalize via division by t_{\min} of our method. The result of this normalized averaging is displayed in Fig. 11. Evaluation of the plot suggests that for the average problem instance our algorithm is most likely to achieve near-optimal results at 98% optimality, and perform

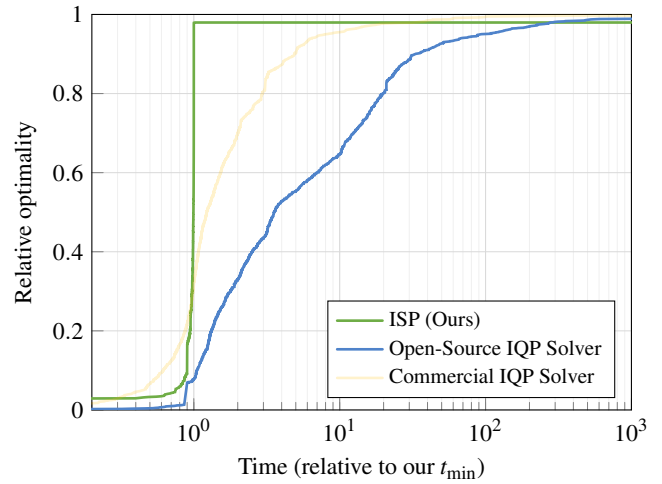


Figure 11: Comparison of relative optimality over time on a per-model basis. Plots of all test instances were normalized along the time axis, relative to t_{\min} of our algorithm, then averaged. Our method achieves its average final optimality of $\approx 98\%$ in about a hundredth of the time the open-source IQP solver needs to reach similar optimality levels on average and in about a tenth of the time the commercial IQP solver requires. Past these times both IQP solvers achieve minor further improvement on average, albeit at diminishing rates.

faster than both the commercial and open-source IQP methods by one or two orders of magnitude, respectively.

To demonstrate the performance of our method when processing a large dataset sequentially, we measured a second time series in which all experiment instances are processed sequentially and both time and relative optimality are accumulated over the course of the experiment. In this case, running the IQP solvers without an early termination criterion is impractical, as they would sometimes spend huge amounts of time to marginally improve or to conclude optimality before advancing to the next test instance. For fair comparison, we therefore test our method against the IQP approach with early termination. Concretely, in one experiment we allow termination once the solver’s optimality gap drops below 30%, in a second experiment we instead set an upper time limit, enabling early termination after 20 seconds of processing time per model. The resulting cumulative optimality curve is plotted in Fig. 12. Remarkably, the gap-based stopping criterion, despite allowing a rather high suboptimality of 30%, proves to be very ineffective at reducing the overall execution time of the IQP solver. On the other hand the time-limited IQP solver performs objectively worse than our approach in terms of both speed and optimality. We conclude that our method in this scenario again performs clearly better in terms of speed and comparably in terms of optimality, while not requiring the tuning of meta-parameters like early termination criteria to achieve this.

Predictability of t_{end} Generally, a desirable property of an algorithm is allowing to roughly estimate the necessary execution time to produce useful output based on the complexity of the input. For the quantization scenarios relevant here, a simple measure representing

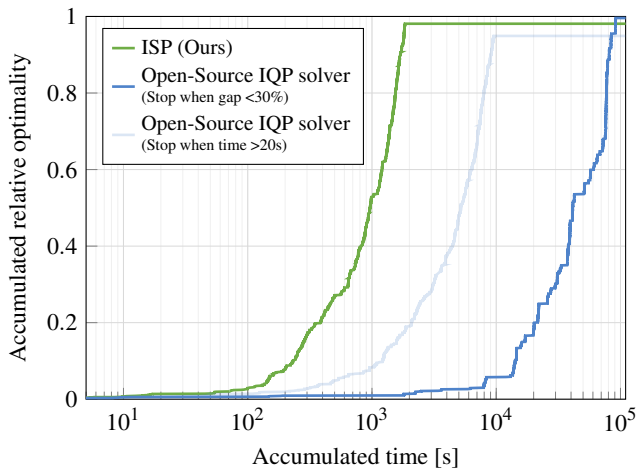


Figure 12: Comparison of relative optimality over time for sequential processing of the whole dataset. The y-axis represents the normalized sum over model-wise relative optimalities. Our method finishes processing all 1568 instances in just 30 minutes. Despite the large allowable optimality gap of 30%, the total processing time of the gap-limited IQP solver exceeds ours by two orders of magnitude, showing the ineffectivity of this early stopping criterion. The time-limited IQP solver performs about an order of magnitude faster than its gap-limited counterpart but drops to 95% overall relative optimality, thereby being surpassed by our algorithm in terms of both speed and quality.

input complexity is the number of arcs in the input cell decompositions, because these are directly associated with the variables of interest.

Fig. 13 shows the distribution of algorithm termination times t_{end} . For the IQP solvers a 10% optimality gap was allowed as early termination condition. Overall, the qualitative differences in run time are reflected here once more, with our method in the lowest tier of termination times. More interestingly, though, we see wildly diverging termination times for models above ≈ 300 arcs in case of the open-source IQP solver or above ≈ 800 arcs for the commercial solver, in some cases spanning a large range of run times even for the same, but differently scaled input model. Due to the time limit that was necessary for practicality, it is unclear how much further this divergence actually goes for large inputs. For our algorithm, run times seem to scale much more predictably with input size.

Optimality of d_{min} We reported above already the average accuracy of our method when compared to a (quasi-)optimal benchmark. Fig. 14 shows a more detailed view of the distribution of final objective function values d_{min} . For the IQP approach we consider two values per instance: the final objective value and the (possibly intermediate) objective value at $10\times$ the time t_{end} required for our algorithm to terminate.

Halting the IQP solver after $10\times$ the time required, results in an equally good objective value for 71% of instances, in a better value in about 12% of cases and a worse value in 17% of cases, including 7% where no feasible solution had been found at all. Our

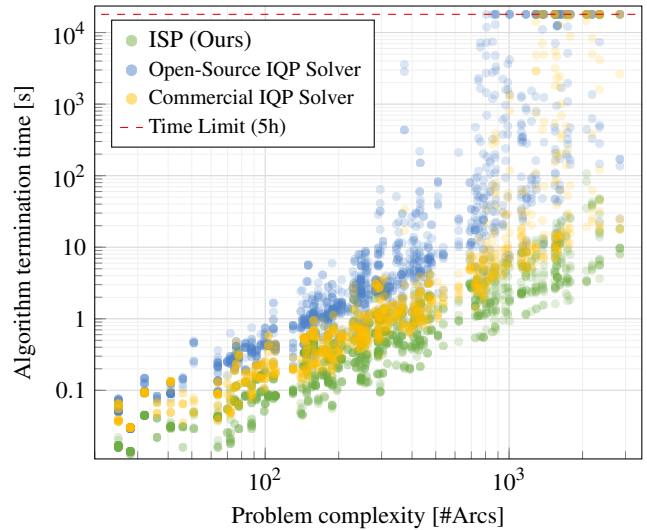


Figure 13: Comparison of run time over input size, as represented by the number of arc variables. Run time variance gives an estimate of the predictability of algorithm run times. IQP solvers, due to large gaps between typical and worst-case performance, are typically lacking in this regard – as becomes evident also here, with variances of up to four orders of magnitude for similarly complex inputs. Run times of our algorithm on similar inputs rarely differ by more than one order of magnitude and never by more than two.

final solution in turn was on par with the final solution of the IQP approach in 84% of cases, worse in 16% of cases and better in 2% of cases. Such latter cases, where our greedy algorithm outperforms the IQP algorithm in terms of optimality, can occur for two reasons: first, the time limit of 5h for the IQP solver, and second, the fact that the lazily added separation constraints [BBC22] are over-zealous (sufficient but not necessary); depending on the path of optimization, different parts of the feasible space are excluded.

Significance of mixed-sign sheets Finally, let us also evaluate the significance of our proposed extension to mixed-sign sheets. As demonstrated in Fig. 9 we have identified cases, where considering only uniform-sign updates leads to getting stuck in certain local minima – even in 2D cases. The inset of Fig. 14 demonstrates the performance of our algorithm with and without mixed-sign sheets enabled. Enabling mixed-sign sheets yields significantly better final objective values in about a quarter of cases, with best-case improvements by factors of 2 and above. Let us remark that these mixed-sign sheets are a general conceptual improvement also over the greedy 2D quantization algorithm [CBK15]; such mixed-sign updates are not amenable to that algorithm’s Dijkstra-based approach.

5.2. Hexahedral Meshing

Hexahedral meshes implied by the computed quantizations were produced using the ALGOHEX library [LB23]. Note that the T-mesh is used only as an auxiliary structure to derive and represent suitable integer spacings between critical entities (singularities, boundaries, and features), the position of T-mesh walls does not carry over to the

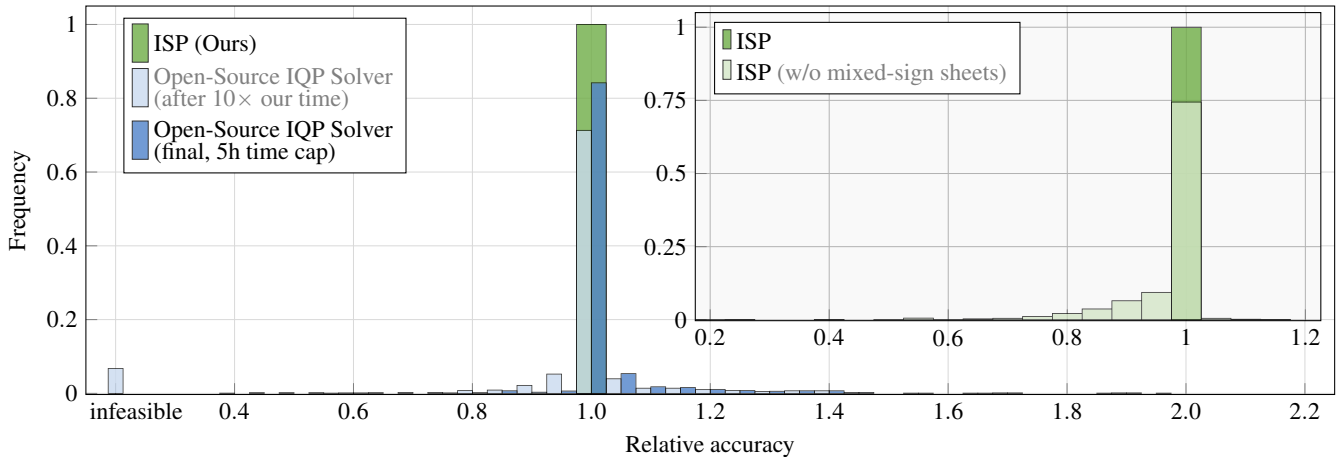


Figure 14: Comparison of other methods' accuracy (in terms of achieved objective value) relative to our method, over the test instances. It can be seen that even after granting the IQP solver $10\times$ the time per instance that our method takes, it has not found an even just feasible quantization for about 7% of the instances, and the quantization found for the others is slightly worse than ours on average. Furthermore, our method's result is close to optimal or optimal in the vast majority of cases: The final solution returned by the IQP solver after nearly unlimited time (a generous time cap of 5h) is better by 2.6% on average. Regarding the few outliers, see Fig. 15. The inset illustrates the benefit of enabling mixed-sign sheets. Notice how in some cases it is crucial in order to escape bad local minima.

resulting hexahedral mesh. Instead the quantization on the T-mesh is used only to determine exactly the integer degrees of freedom of a global integer-grid map subsequently optimized for low distortion. The resulting hexahedral sheets extracted from it do not necessarily align with the precursor T-mesh walls, only the marked features and singularities are matched by the mesh as intended.

As a consequence, differences in optimality of a T-mesh quantization do not directly translate into equivalently large differences in hex mesh quality, but rather inaccuracies in achieving exactly the desired target resolution of the hex mesh – especially when the differences in quantization accuracy are relatively small, like with the average 2% suboptimality of our method's results. Fig. 16 demonstrates to what extent typical differences between our method's quantization and the optimal quantization are reflected in the resulting hex meshes – including a note on the relative speedup for reference. As conjectured before, there are no noticeable differences, not even if the quantization accuracy differs by 10% or 20%. Taking a look at the so-called base complex of the generated hex meshes, one can find small differences in the layout, mostly due to different relative alignment of singularities in the mesh structure.

5.3. Limitations

There are however, in contrast to the cases discussed above, situations where the exact alignment of singularities in the mesh play a role also for hexahedral element quality. Specifically if there are singularities that in the T-mesh are barely misaligned and confined by other critical entities surrounding them, it can be relevant that the quantization realigns the singularities by very strategically placed 0s. Otherwise there may be very flat or misshaped hexahedral sheets in the output that need to squeeze between singularities and boundaries in geometrically unfavorable ways. Fig. 15 shows one of the rare examples where our quantization process ends up in a rather

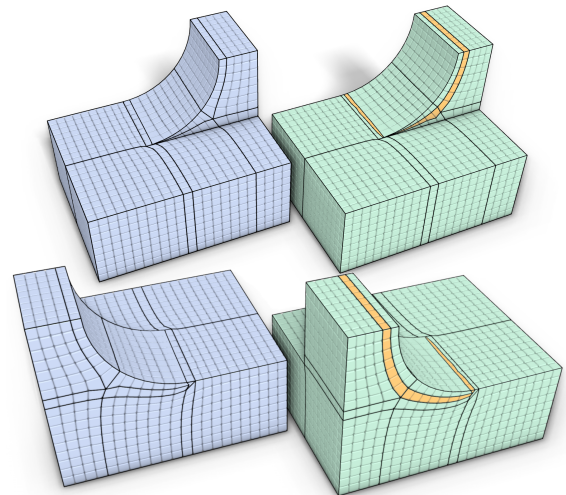


Figure 15: Left: result using the IQP solver. Right: result using our ISP method. The main difference is an additional sheet (orange). This causes a quantization objective higher by 57%. In essence, one more layer of hexahedra squeezes into the sharp tip near the base of the ramp, or in terms of the quantization: more very short arcs of target length ≈ 0 were quantized to length 1.

suboptimal local minimum that essentially contains an additional sheet – that cannot monotonically be removed using our operators. As it is very flat near the base of the ramp (squeezing between the boundary and a prescribed singularity), it has a strong impact on the objective value. In part this is also due to the prescribed singularity structure being quite badly located near the base of the ramp in this input example from the dataset.

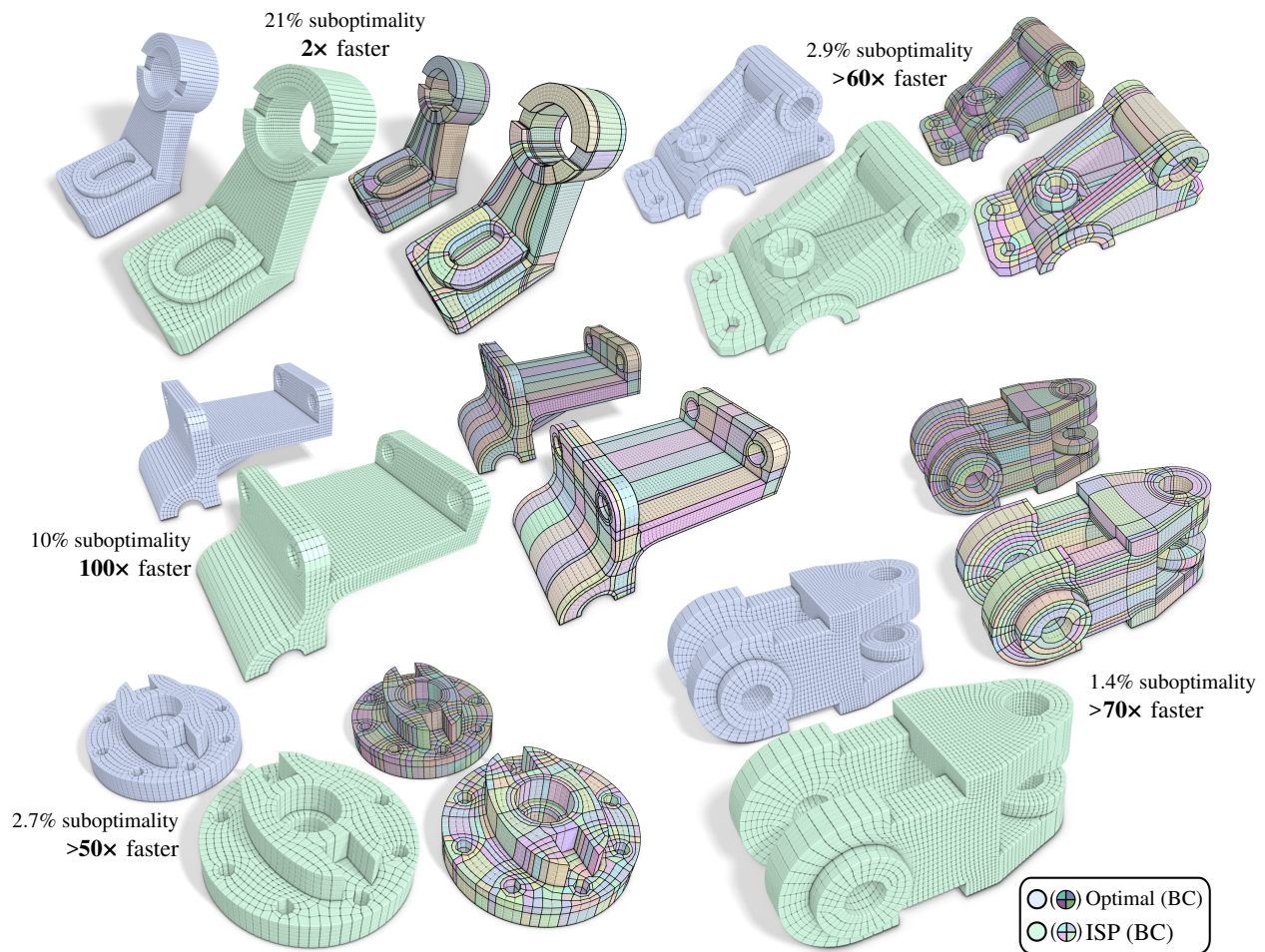


Figure 16: A selection of hex meshes produced from optimal quantizations (blue) or from our ISP quantizations (green). Beside the plain hex meshes a highlighting of the structure of the respective base complex (BC) is shown, which allows an easier visual inspection of singularity alignment and element count. Times (relative to the commercial IQP solver) and suboptimality numbers refer to the quantization phase.

6. Conclusion

Taking inspiration from previous work on 2D quantization for quad meshing, we have presented a novel greedy LP-based formulation, that allowed us to propose a more general approach suitable for 3D quantization for hexahedral meshing. Focusing on aspects of open-source availability, usability, and speed we have demonstrated clear advantages of using our tailored approach for volumetric quantization over the previous state of the art which relies on hard-to-configure, expensive, or slow general-purpose IQP solvers.

Building on our formulation, it will be interesting to investigate variants supporting also structural modification of the singularity graph by singularity recombination or insertion. Such an approach may benefit also cases like the one shown in Fig. 15. Also additional types of integer-sheet operators are worth investigating for such cases, to help escaping the occasional local minimum of low quality.

Another interesting avenue for future research is the design and use of objective functions in the quantization phase that are more

directly tied to the resulting mesh and its quality rather than focusing somewhat indirectly on a per-arc fitting of sizing.

Acknowledgements

The authors thank Martin Heistermann for for suggesting the specific nesting example shown in Fig. 7 and for insightful exchanges concerning related work. D. Bommes has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (AlgoHex, grant agreement No 853343).

References

- [BBC*08] BONAMI P., BIEGLER L. T., CONN A. R., CORNUÉJOLS G., GROSSMANN I. E., LAIRD C. D., LEE J., LODI A., MARGOT F., SAWAYA N., WÄCHTER A.: An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization* 5, 2 (2008). 7
- [BBC22] BRÜCKLER H., BOMMES D., CAMPEN M.: Volume parametrization quantization for hexahedral meshing. *ACM Trans. Graph.* 41, 4 (2022). 1, 2, 3, 4, 7, 9, 13

- [BC23] BRÜCKLER H., CAMPEN M.: Collapsing embedded cell complexes for safer hexahedral meshing. *ACM Trans. Graph.* 42, 6 (2023). 2, 3, 4, 7
- [BCE*13] BOMMES D., CAMPEN M., EBKE H.-C., ALLIEZ P., KOBBELT L.: Integer-grid maps for reliable quad meshing. *ACM Trans. Graph.* 32, 4 (2013), 98:1–98:12. 3
- [BGMC22] BRÜCKLER H., GUPTA O., MANDAD M., CAMPEN M.: The 3D Motorcycle Complex for Structured Volume Decomposition. *Computer Graphics Forum* 41, 2 (2022). 3, 7
- [Bla00] BLACKER T.: Meeting the challenge for automated conformal hexahedral meshing. In *Proc. Int. Meshing Roundtable* (2000), pp. 11–20. 3
- [BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quad-rangulation. *ACM Trans. Graph.* 28, 3 (2009). 3
- [CBK15] CAMPEN M., BOMMES D., KOBBELT L.: Quantized global parametrization. *ACM Trans. Graph.* 34, 6 (2015). 2, 3, 4, 6, 9
- [CC19] CORMAN E., CRANE K.: Symmetric moving frames. *ACM Trans. Graph.* 38, 4 (2019). 3
- [CLS16] CHERCHI G., LIVESU M., SCATENI R.: Polycube simplification for coarse layouts of surfaces and volumes. *Computer Graphics Forum* 35, 5 (2016), 11–20. 3
- [CODH*24] COUDERT-OSMONT Y., DESOBRY D., HEISTERMANN M., BOMMES D., RAY N., SOKOLOV D.: Quad mesh quantization without a t-mesh. *Computer Graphics Forum* 43, 1 (2024). 3
- [CSZZ19] CAMPEN M., SHEN H., ZHOU J., ZORIN D.: Seamless parametrization with arbitrary cones for arbitrary genus. *ACM Trans. Graph.* 39, 1 (2019). 3
- [EGKT08] EPPSTEIN D., GOODRICH M. T., KIM E., TAMSTORF R.: Motorcycle graphs: canonical quad mesh partitioning. *Computer Graphics Forum* 27, 5 (2008), 1477–1486. 3
- [GDC15] GAO X., DENG Z., CHEN G.: Hexahedral mesh re-parameterization from aligned base-complex. *ACM Trans. Graph.* 34, 4 (2015). 3
- [GLA23] GUNPINAR E., LIVESU M., ATTENE M.: Exploration of 3d motorcycle complexes from hexahedral meshes. *Computers & Graphics* (2023). 3
- [GPW*17] GAO X., PANOZZO D., WANG W., DENG Z., CHEN G.: Robust structure simplification for hex re-meshing. *ACM Trans. Graph.* 36, 6 (2017). 3
- [Gra08] GRADY L.: Minimal surfaces extend shortest path segmentation methods to 3d. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 2 (2008), 321–334. 5
- [Gur24] GUROBI OPTIMIZATION, LLC: Gurobi Optimizer, 2024. URL: <https://www.gurobi.com>. 7
- [HWB23] HEISTERMANN M., WARNETT J., BOMMES D.: Min-deviation-flow in bi-directed graphs for t-mesh quantization. *ACM Trans. Graph.* 42, 4 (2023). 3
- [JHW*14] JIANG T., HUANG J., WANG Y., TONG Y., BAO H.: Frame field singularity correction for automatic hexahedralization. *IEEE Trans. Vis. Comput. Graph.* 20, 8 (2014). 3
- [KNP07] KÄLBERER F., NIESER M., POLTHIER K.: QuadCover - surface parameterization using branched coverings. *Computer Graphics Forum* 26, 3 (2007), 375–384. 3
- [LB23] LIU H., BOMMES D.: Locally meshable frame fields. *ACM Trans. Graph.* 42, 4 (2023). 2, 3, 4, 9
- [LBK16] LYON M., BOMMES D., KOBBELT L.: Hexex: Robust hexahedral mesh extraction. *ACM Trans. Graph.* 35, 4 (2016). 3
- [LCBK19] LYON M., CAMPEN M., BOMMES D., KOBBELT L.: Parametrization quantization with free boundaries for trimmed quad meshing. *ACM Trans. Graph.* 38, 4 (2019). 3
- [LCK21a] LYON M., CAMPEN M., KOBBELT L.: Quad layouts via constrained t-mesh quantization. *Computer Graphics Forum* 40, 2 (2021). 3
- [LCK21b] LYON M., CAMPEN M., KOBBELT L.: Simpler quad layouts using relaxed singularities. *Computer Graphics Forum* 40, 5 (2021), 169–179. 3
- [Lev21] LEVI Z.: Direct seamless parametrization. *ACM Trans. Graph.* 40, 1 (2021). 3
- [LH03] LOUGEE-HEIMER R.: The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development* 47, 1 (2003), 57–66. 7
- [LLX*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex meshing using singularity-restricted field. *ACM Trans. Graph.* 31, 6 (2012), 177. 3
- [LPP*20] LIVESU M., PIETRONI N., PUPPO E., SHEFFER A., CIGNONI P.: Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing. *ACM Trans. Graph.* 39, 4 (2020). 3
- [LZC*18] LIU H., ZHANG P., CHIEN E., SOLOMON J., BOMMES D.: Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.* 37, 4 (2018). 3
- [MC19] MANDAD M., CAMPEN M.: Exact constraint satisfaction for truly seamless parametrization. *Computer Graphics Forum* 38, 2 (2019), 135–145. 3
- [Mit23] MITCHELL S. A.: Incremental interval assignment by integer linear algebra with improvements. *Computer-Aided Design* 158 (2023). 3
- [NRP11] NIESER M., REITEBUCH U., POLTHIER K.: Cubecover – parameterization of 3d volumes. *Computer Graphics Forum* 30, 5 (2011), 1397–1406. 3
- [PBS20] PALMER D., BOMMES D., SOLOMON J.: Algebraic representations for volumetric frame fields. *ACM Trans. Graph.* 39, 2 (2020). 3
- [PBSB08] PARRISH M., BORDEN M., STATEN M., BENZLEY S.: A selective approach to conformal refinement of unstructured hexahedral finite element meshes. In *Proc. 16th Int. Meshing Roundtable* (2008), Springer, pp. 251–268. 3
- [PCS*22] PIETRONI N., CAMPEN M., SHEFFER A., CHERCHI G., BOMMES D., GAO X., SCATENI R., LEDOUX F., REMACLE J., LIVESU M.: Hex-mesh generation and processing: A survey. *ACM Trans. Graph.* 42, 2 (2022). 3
- [PNA*21] PIETRONI N., NUVOLE S., ALDERIGHI T., CIGNONI P., TARINI M., ET AL.: Reliable feature-line driven quad-remeshing. *ACM Trans. Graph.* 40, 4 (2021). 3
- [PRR*22] PROTAIS F., REBEROL M., RAY N., CORMAN E., LEDOUX F., SOKOLOV D.: Robust quantization for polycube maps. *Computer-Aided Design* 150 (2022), 103321. 3
- [PS98] PAPADIMITRIOU C. H., STEIGLITZ K.: *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998. 5
- [RSL16] RAY N., SOKOLOV D., LÉVY B.: Practical 3d frame field generation. *ACM Trans. Graph.* 35, 6 (2016). 3
- [SDW*10] SHEPHERD J. F., DEWEY M. W., WOODBURY A. C., BENZLEY S. E., STATEN M. L., OWEN S. J.: Adaptive mesh coarsening for quadrilateral and hexahedral meshes. *Finite Elements in Analysis and Design* 46, 1-2 (2010). 3
- [SGW21] SHEN C., GAO S., WANG R.: Topological operations for editing the singularity on a hex mesh. *Engineering with Computers* 37, 2 (2021), 1357–1375. 3
- [SJ08] SHEPHERD J. F., JOHNSON C. R.: Hexahedral mesh generation constraints. *Engineering with Computers* 24, 3 (2008), 195–213. 3
- [SRRGRN14] SARRATE RAMOS J., RUIZ-GIRONÉS E., ROCA NAVARRO F. J.: Unstructured and semi-structured hexahedral mesh generation methods. *Computational Technology Reviews* 10 (2014), 35–64. 3

- [SSLS10] STATEN M. L., SHEPHERD J. F., LEDOUX F., SHIMADA K.: Hexahedral mesh matching: Converting non-conforming hexahedral-to-hexahedral interfaces into conforming interfaces. *Int J Num Methods Engineering* 82, 12 (2010), 1475–1509. 3
- [SVB17] SOLOMON J., VAXMAN A., BOMMÈS D.: Boundary element octahedral fields in volumes. *ACM Trans. Graph.* 36, 4 (2017), 1. 3
- [Tak19] TAKAYAMA K.: Dual Sheet Meshing: An Interactive Approach to Robust Hexahedralization. *Computer Graphics Forum* 38, 2 (2019), 37–48. 3
- [Tau01] TAUTGES T. J.: The generation of hexahedral meshes for assembly geometry: survey and progress. *Int J Num Methods Engineering* 50, 12 (2001), 2617–2642. 3
- [WGZC18] WANG R., GAO S., ZHENG Z., CHEN J.: Hex mesh topological improvement based on frame field and sheet adjustment. *Computer-Aided Design* 103 (2018), 103–117. 3
- [ZVC*20] ZHANG P., VEKHTER J., CHIEN E., BOMMÈS D., VOUGA E., SOLOMON J.: Octahedral frames for feature-aligned cross fields. *ACM Trans. Graph.* 39, 3 (2020). 3

Appendix A: Complete LP Formulation

With all extensions described in Sec. 4.3 the final LP is:

$$\min_{\Delta q, \Delta q \geq 0} \sum_{a_i \in \mathcal{A}} \Delta q_i w_i^+ + \Delta q_i w_i^- \quad (7a)$$

$$\text{s.t. } [\mathcal{A} \mid -\mathcal{A}] \begin{bmatrix} \Delta q \\ \Delta q \end{bmatrix} = \mathbf{0} \quad (7b)$$

$$[\mathcal{B} \mid -\mathcal{B}] \begin{bmatrix} \Delta q \\ \Delta q \end{bmatrix} \geq \mathbf{1} - \mathcal{B}q_{\text{pre}} \quad (7c)$$

$$[\mathcal{C} \mid -\mathcal{C}] \begin{bmatrix} \Delta q \\ \Delta q \end{bmatrix} \geq -\mathcal{C}q_{\text{pre}} \quad (7d)$$

$$\begin{cases} \Delta q_j \geq 1 \wedge \Delta q_j = 0 & (7e) \\ \Delta q_j = 0 \wedge \Delta q_j \geq 1 & (7f) \\ \Delta q = \mathbf{0} & (7g) \end{cases}$$

Defining $\epsilon_i^\pm = \pm(\ell_i - q_i)$, the weights w_i^\pm are given by

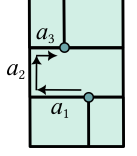
$$w_i^\pm = \begin{cases} 1/(1 + \epsilon_i^\pm), & \text{if } 1 \leq \epsilon_i^\pm \\ |\mathcal{A}|/(1 + \epsilon_i^\pm), & \text{if } 0 \leq \epsilon_i^\pm < 1 \\ |\mathcal{A}|^2(1 - \epsilon_i^\pm), & \text{if } \epsilon_i^\pm < 0 \end{cases}$$

This LP is invoked in multiple ways by Algs. 2 and 3.

Appendix B: Failsafe Feature Separation

In the LP (7) all equality constraints are homogeneous. If in addition all inequality constraints had non-negative coefficients only, any positive multiple of a solution would be a solution as well. This is of relevance for the rare cases in which we need further quantization updates in order to establish separation, but the LP yields a non-integer solution that needs to be scaled up. By using a non-deflating mode in this case, i.e. constraint (7g) is activated, all relevant coefficients of eq. (7d) are non-negative. The separation constraints (7c), however, depending on how they are formulated may contain some negative coefficients in matrix \mathcal{B} . Alg. 3 therefore, if necessary, switches to a non-negative matrix \mathcal{B}^+ (defined below) to ensure separation can always be achieved.

The separation constraints arise from a graph search on the T-mesh [BBC22, Alg.1]. A path between two features that implies zero quantized distance between these serves as a witness of feature non-separation. Based on the discovered path, an additional constraint is devised that enforces separation. Such a path, by construction, always has one axis along which it expands monotonically – here the axis of arc a_2 . Along the other axes the path may wind back and forth, as seen here for a_1 and a_3 . Separation along one axis is sufficient, i.e. the desired constraint is $q_2 > 0 \vee q_1 - q_3 \neq 0$. This can be turned into a linear constraint (for separation matrix \mathcal{B}) in a conservative manner in different ways. By default, we use $q_2 + (q_1 - q_3) > 0$ [BBC22]. When building matrix \mathcal{B}^+ we instead turn it into $q_2 > 0$, i.e. separation specifically along the monotone axis is asked for, leading to only non-negative coefficients.



Appendix C: Subroutine Pseudocode

Algorithm 2: MINIMALARCUPDATE

Input: T-mesh with target lengths ℓ and current quantization q_{pre} ; matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}$; root arc a_j

Output: Integer quantization update Δq improving q_j

```

 $\Delta q = 0$ 
if  $q_{\text{pre},j} < \ell_j$  then
  |  $\Delta q = \text{solve eq. (7a), s.t. eqs. (7b)–(7d) and (7e)}$ 
else
  | if eqs. (7c) or (7d) incompatible with eq. (7f) then
  | | return  $\mathbf{0}$ 
  |  $\Delta q = \text{solve eq. (7a), s.t. eqs. (7b)–(7d) and (7f)}$ 
if  $\Delta q$  not integer then
  |  $\Delta q = \text{lcd}(\Delta q) \cdot \Delta q$ 
  | if  $\Delta q$  violates eqs. (7c) or (7d) then
  | | return  $\mathbf{0}$ 
return  $\Delta q$ 

```

Algorithm 3: MINIMALSEPARATINGUPDATE

Input: T-mesh with target lengths ℓ and current quantization q_{pre} ; matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}$; root arc a_j

Output: Integer quantization update Δq enforcing separation

```

//Default mode//
 $\Delta q = \text{solve eq. (7a), s.t. eqs. (7b)–(7d)}$ 
if  $\Delta q$  is not integer then
  |  $\Delta q = \text{lcd}(\Delta q) \cdot \Delta q$ 
  | if  $\Delta q$  violates eqs. (7c) or (7d) then
  | | //Non-deflating mode//
  | |  $\Delta q = \text{solve eq. (7a), s.t. eqs. (7b)–(7d) and (7g)}$ 
  | | if  $\Delta q$  is not integer then
  | | |  $\Delta q = \text{lcd}(\Delta q) \cdot \Delta q$ 
  | | | if  $\Delta q$  violates eq. (7c) then
  | | | | //Failsafe mode (App. B)//
  | | | |  $\mathcal{B} = \mathcal{B}^+$ 
  | | | |  $\Delta q = \text{solve eq. (7a), s.t. eqs. (7b)–(7d) and (7g)}$ 
  | | | | if  $\Delta q$  is not integer then
  | | | | |  $\Delta q = \text{lcd}(\Delta q) \cdot \Delta q$ 
return  $\Delta q$ 

```
