Interactively Controlled Quad Remeshing of High Resolution 3D Models

Hans-Christian Ebke[1] Patrick Schmidt[1] Marcel Campen[2] Leif Kobbelt[1]
[1]RWTH Aachen University [2]New York University

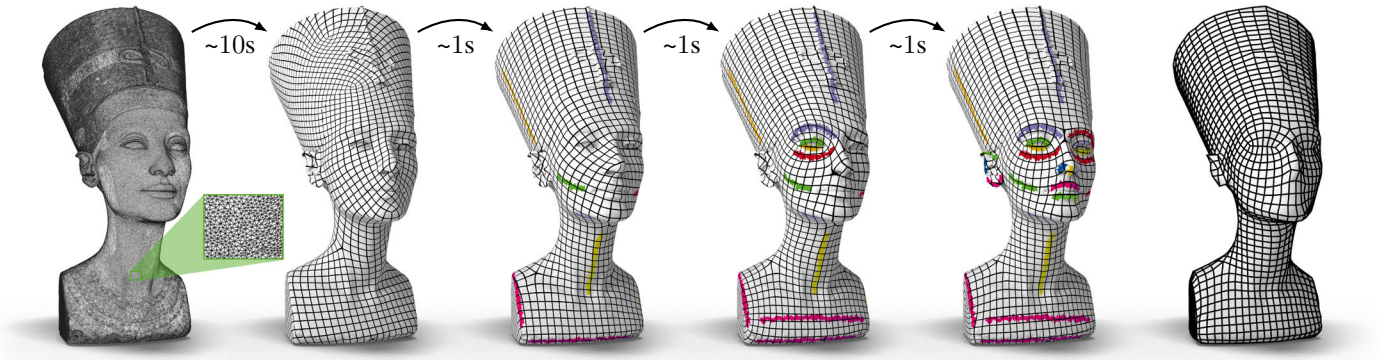


Figure 1: *Steps of a workflow based on our method. Starting from a high resolution input mesh (2M faces), a hierarchichal representation and an unconstrained integer grid parametrization are computed in a one-off initialization step (taking 10s). Subsequently, the user can iteratively add or change constraints to adjust the edge-flow and other mesh properties, while being provided with near-instant feedback at interactive rates (each update taking around 1s). Eventually, a quad mesh can be extracted from the parametrization. Without our framework, just using the underlying method [Bommes et al. 2009; Ebke et al. 2014] in its original form, every single update step takes 5 minutes.*

Abstract

Parametrization based methods have recently become very popular for the generation of high quality quad meshes. In contrast to previous approaches, they allow for intuitive user control in order to accommodate all kinds of application driven constraints and design intentions. A major obstacle in practice, however, are the relatively long computations that lead to response times of several minutes already for input models of moderate complexity. In this paper we introduce a novel strategy to handle highly complex input meshes with up to several millions of triangles such that quad meshes can still be created and edited within an interactive workflow. Our method is based on representing the input model on different levels of resolution with a mechanism to propagate parametrizations from coarser to finer levels. The major challenge is to guarantee consistent parametrizations even in the presence of charts, transition functions, and singularities. Moreover, the remaining degrees of freedom on coarser levels of resolution have to be chosen carefully in order to still achieve low distortion parametrizations. We demonstrate a prototypic system where the user can interactively edit quad meshes with powerful high-level operations such as guiding constraints, singularity repositioning, and singularity connections.

Keywords: integer grid maps, quad meshing, interactive

Concepts: •**Computing methodologies** → **Mesh models; Mesh geometry models;**

SA '16 Technical Papers, December 05–08, 2016, Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2982413

1 Introduction

The problem of generating a quad mesh for a given surface has a long history and has been tackled by many different computational approaches over the decades. In recent years, a field-guided parametrization-based strategy, suggested, e.g., by Knupp [1995] and later elaborated in detail [Ray et al. 2006; Kälberer et al. 2007; Bommes et al. 2009], has gained popularity. It is based on generating a parametrization of the input surface that maps the grid of integer iso-lines in $\mathbb{R}^2$ onto the surface in such a way that it induces a quad mesh. Bommes et al. [2013] coined the term *integer grid map* (IGM) for such parametrizations.

This approach's popularity is due to its high result quality paired with the fact that the result can be influenced by several types of constraints that are intuitive even for novice users. Thus, at least in theory, IGM based quad meshing allows for a workflow where the design space is explored by iteratively adding, modifying, or deleting constraints (cf. Figure 1). In practice this kind of workflow quickly breaks down due to the run time complexity of the underlying vector field generation and surface parametrization methods. Inacceptable delays well over 10 seconds can already occur on simple input models triangulated with as few as some ten-thousand triangles. The generic framework we present here addresses this common shortcoming, lifting the family of field-guided quad meshing methods into the realm of interactive methods.

At its core, our framework exploits the fact that the run time of the process of IGM computation depends first and foremost on the complexity of the input model's triangulation. A straightforward way to speed up the process to interactive rates thus is to heavily decimate the input and simply operate on this simpler version. This naive approach, however, has two crucial shortcomings:

- Obviously, the IGM computed on the decimated mesh will differ, and it might differ greatly. In particular, the IGM does not change gracefully, in a manner continuous in the geometric change induced by the decimation; the IGM has discrete and combinatorial properties (the singularities and the integer transitions, defining the global mesh structure), and these change abruptly and globally under mesh decimation. This

is particularly true for common decimation objectives such as Hausdorff distance minimization, as lower Hausdorff distance does not imply more similar IGM optimization results.

- The resulting IGM is defined not on the original mesh, but on a decimated version. Depending on the application context, an IGM on the original mesh may be required by subsequent processing steps. In the case of mere quad mesh extraction one could work with the coarse IGM, but the resulting quad mesh will have lower approximation quality (cf. Figure 2), alignment to features may be lacking, and user constraints may not be met correctly. One might consider transferring the IGM to the original mesh, but standard upsampling or prolongation operators are not applicable due to the specific discrete structure (cuts, transitions, and singularities in the IGM).

1.1 Contribution

We present a framework to enable the computation of IGMs on highly complex meshes at a run time quick enough to enable a truly interactive work flow. In order to avoid the problems of the above naive approach, we make the following, threefold contribution:

- A new decimation objective, aiming for Gaussian curvature preservation, that we use for the incremental decimation of the input mesh. As we show, this approach is highly beneficial in our scenario, keeping the IGM (in particular its guiding field) computed on the decimated mesh close to the ground truth.
- A novel coarse-to-fine mapping approach that allows us to map a global, chart-based parametrization (in particular an IGM) computed on a decimated mesh onto the original mesh in linear time. It is aware of charts, transitions, and singularities, and maintains their consistency throughout. This allows us to hide the coarse mesh as a proxy in the background, completely transparent to the user and downstream applications.
- A framework building on the above techniques to interactively explore the integer grid map design space. It is based on a specific three-tier hierarchy of the input model, custom-tailored for the IGM use case in order to be able to offer all of the usual design tools to the user. Thanks to the efficiency and low complexity of our coarse-to-fine mapping method, interactive rates are achieved even on meshes with millions of triangles.

Typical round-trip times (i.e. delays between adding a user constraint and completing the re-generation of the IGM and its visualization on the input mesh) achieved in our experiments (using [Ebke et al. 2014; Bommes et al. 2009] as underlying IGM computation method) were below 1 second on complex meshes of up to about 350k triangles and below 5 seconds on highly complex meshes of about 4 million triangles. The same IGM method used without our framework takes over 1 minute and over 20 minutes, respectively, for a single update using identical parameters and constraints.

Note that we do not present a new parametrization objective or rounding strategy for IGM computation, nor do we propose concrete UI metaphors to let the user specify the constraints suitable for our case. Our method can rather be understood as a framework, built around and oblivious to the concrete parametrization optimization method and the user interface (cf. Figure 3).

2 Related Work

Field-guided Parametrization

The aforementioned quad remeshing methods derive a great part of their user control from the orientation fields that guide their parametrization process. There is a sizeable amount of methods to choose from for the generation and modification of such fields [Vaxman et al. 2016]. They can loosely be categorized into those that generate 4-symmetric cross fields [Palacios and Zhang 2007; Ray et al. 2008; Knöppel et al. 2013; Liu et al. 2016] and methods that exploit more degrees of freedom by generating non-orthogonal fields [Panozzo et al. 2014; Diamanti et al. 2014; Jiang et al. 2015].

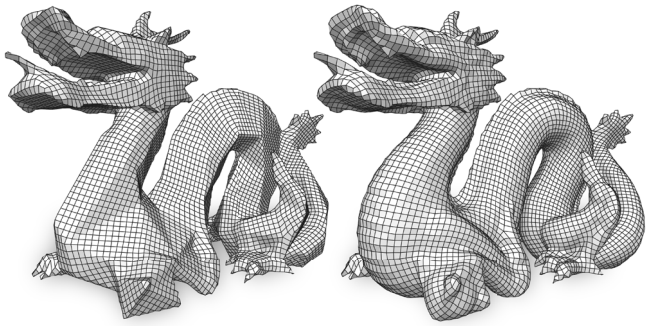


Figure 2: *Left: a quad mesh extracted directly from an IGM on a decimated mesh. Right: after mapping the same IGM to the original input mesh using our coarse-to-fine mapping technique, a quad mesh without the coarse discretization artifacts can be extracted.*

Using a Poisson formulation, parametrizations whose isolines strive to follow those fields' directions can be computed, as has been demonstrated in various contexts [Ray et al. 2006; Bommes et al. 2009; Pietroni et al. 2011; Nieser et al. 2012; Bommes et al. 2013; Marcias et al. 2013; Ebke et al. 2014; Campen and Kobbelt 2014; Li et al. 2015]. Using additional rounding techniques [Campen et al. 2015; Bommes et al. 2013; Bommes et al. 2009] these parametrizations can be turned into IGMs.

The framework we present in this paper can be used to speed up any combination of guiding field and IGM computation method. In fact, as it does not rely on the map's integer properties, it also works for general so-called seamless parametrizations, e.g. [Ray et al. 2010; Myles and Zorin 2012; Myles and Zorin 2013], which broadens the range of application scenarios beyond quad meshing.

Multi-Resolution

A common strategy to speed up computations on densely tessellated surfaces is the use of multi-resolution hierarchies or multigrid approaches.

Multi-level mesh hierarchies can be obtained through incremental decimation [Hoppe 1996]. Several approaches exploiting these for the purpose of parametrization have been proposed, mainly in the context of texture mapping [Sander et al. 2001; Sander et al. 2002; Cohen et al. 1998; Hoppe 1999; Lee et al. 1998; Hormann et al. 1999]. However, they either map the parametrization in fine-to-coarse direction only or they are tailored to disk-topology parametrization. The only line of works we are aware of which consider (in some sense) transferring chart transition functions in coarse-to-fine direction [Khodakovsky et al. 2003; Pietroni et al. 2010] relies on a dual setting (cuts are across rather than along edges) such that it is not applicable to IGMs, with their specific chart transitions and their singularities. Daniels et al. [2011] mention the related possibility of mapping triangles of a coarse level to corresponding "approximate geodesic triangles" on a fine level, without elaborating on the robust automatic determination of these, though.

Bommes et al. [2013] use a two-level hierarchy for IGM computation. Only the parametrization is computed on the coarse level; the

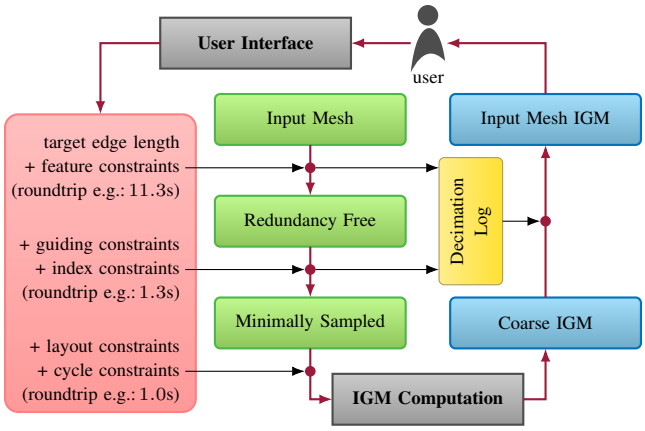


Figure 3: *An overview of the presented framework. A three-tier mesh hierarchy (green) is generated. Decimation records are kept in a log. IGM computation is performed on the coarsest level, and the coarse IGM (blue) is transformed into a fine IGM (blue) on the original mesh using the decimation log. This IGM serves as feedback based upon which the user adds, removes, or changes some constraints by means of a suitable UI (top). The different types of constraints are fed into the pipeline at different decimation stages (left), thus the next update iteration might be able to start at a low level of the hierarchy, significantly saving re-decimation time. This is exemplified with roundtrip timings of the scenario in Figure 1.*

cross field must be computed on the complex input mesh, because the special kind of decimation applied relies on a priori knowledge of the field's singularities. This in particular is a drawback because the cross field optimization problem is a large mixed integer problem, with the number of integer variables linear in the input size. Update rates are thus typically not interactive. Furthermore, the final IGM is defined on the extremely coarse mesh only; a pointwise map to the input mesh is available which, however, does not define a per-face-linear IGM on the input mesh.

Jakob et al. [2015] demonstrated the use of a multi-level strategy in a highly interactive quad(-dominant) meshing scenario. The approach is tailored to local parametrization operations, akin to [Ray et al. 2006], and the coarse-to-fine mapping relies on the absence of global parametrization consistency conditions. It does thus not directly extend to the computation of global seamless parametrizations such as IGMs (the consequent differences to our approach are demonstrated in Figures 9 and 11). In particular, this local approach implies that non-local constraints such as layout or cycle constraints (cf. Section 4) cannot be taken into account.

General (unstructured) multigrid approaches [Aksoylu et al. 2005; Ray and Lévy 2003] are likewise not readily applicable. First of all, it is far from obvious how an IGM, with its discrete transitions and singularities, can properly be mapped between levels using standard prolongation and restriction operators. A further general issue is the fact that IGM computation inherently is a mixed integer problem with discrete degrees of freedom. This makes the problem not well-suited for these approaches, because the solutions behave discontinuously across the levels; in particular, thus all the usual convergence results [Wesseling 2004] do not apply in this case.

3 Method Overview

Figure 3 provides a bird's-eye view over our framework. The initial starting point is the high resolution input mesh. Using the decimation technique, and keeping record of every decimation operation in the *decimation log*, we compute two additional, increasingly coarser meshes of very different nature: the intermediate *redundancy free* mesh, and the coarse *minimally sampled* mesh. The reason for this three-tier hierarchy is as follows.

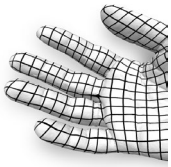
Three-Tier Hierarchy The cost of the IGM computation on the coarse mesh can be made almost arbitrarily small (by an appropriate choice of coarseness). The cost of coarse-to-fine mapping with our efficient strategy is also low. The entire process thus is dominated by the decimation step (in Figure 1 it takes about 9s, whereas IGM computation and coarse-to-fine mapping combined take less than 1s). If this decimation only needs to be done once in the beginning, this certainly is not an issue. However, as this initial decimation is unaware of the constraints (cf. Section 4) the user might be adding, the coarse mesh can become ill-suited at a certain point during interaction, lacking degrees of freedom (i.e. vertices) in regions where the user wants to exert fine-grained control.

A (relatively) costly re-decimation, which preserves the required degrees of freedom, is then necessary. To mitigate this cost, we introduce an intermediate level into the hierarchy: it is a uniformly sampled mesh with a resolution tailored to and sufficient for the two most important forms of user interaction (guiding the quad orientation, i.e. the *edge flow*, and controlling irregular vertices). Upon change of the pertinent constraints through the user, re-decimation can then be performed starting from the intermediate level, often reducing the delay by an order of magnitude or more.

The minimally sampled decimation level is a very coarse, non-uniform subsampling that takes Gaussian curvature, guiding and index constraints into account. Figure 5 displays the hierarchy of an example model. In Section 4 the constraint classes are described in detail, and in Section 5 the decimation process is treated.

Parametrization The IGM computation is performed entirely on the minimally sampled mesh, using any IGM computation method. From the perspective of our framework, the parametrization method can be considered a black box. It is merely required that a seamless parametrization is provided for the subsequent steps.

Coarse-to-Fine Mapping Once a parametrization is available on the minimally sampled mesh it is propagated to the high resolution input mesh using the information recorded in the decimation log while maintaining consistency of the transition functions, singularities, and the layout of the induced quad mesh.



Iterative Modification The parametrized input mesh reflecting the set of provided constraints serves as feedback for the user who can then add, remove, or modify constraints and reiterate the process. Additional run time is saved by not starting over from scratch but, depending on the types of changed constraints, starting re-decimation at the level where they take effect.

4 Design Constraints

We start by examining the five common classes of design constraints offered by state-of-the-art IGM computation methods, and determine what kind of requirements they impose on the decimation strategy to be introduced in Section 5.

Hard Feature Constraints are defined via a set of edges on the input mesh that represent sharp feature curves which should be

faithfully reproduced through edges in the final quad mesh. Such edges can be user-defined, detected by thresholding dihedral angles or by considering larger neighborhoods [Hubeli and Gross 2001].

IGM computation methods enforce this constraint by requiring these edges to lie on integer iso-lines in the parameter domain [Bommes et al. 2009]. In our setting we need to ensure that the coarse mesh still contains these edges (possibly in a sub-sampled form) in order to be able to adequately express these constraints when computing the IGM on the coarse mesh.

Soft Guiding Constraints are defined as connected regions on the surface with an associated smooth direction field. The IGM's iso-lines (and thus the edge flow of the resulting quad mesh) within such a region should be aligned along the given direction field. Such constraint regions can be determined from brush-stroke user interface metaphors, or they can be derived from soft-feature detection methods [Nieser et al. 2012; Gelfand and Guibas 2004; Campen et al. 2016].

Our decimation strategy has to ensure that every such region has a proper counterpart in the coarse mesh and the support of its influence on the parametrization of the entire surface is maintained.

Index Constraints are prescribed indices [Ray et al. 2008] of singularities in the cross field which drives the parametrization process. These indices directly translate into valences of irregular vertices (or the local absence of irregular vertices) in the quad mesh implied by the resulting IGM. They are given as a discrete scalar value on vertices [Bommes et al. 2009]. The most common application scenario for index constraints is in the implementation of a drag-and-drop UI metaphor to edit the irregular vertex configuration through move, merge, and split operations.

This type of constraint can only be enforced by the IGM computation if the vertices with prescribed index have a counterpart in the coarse mesh, i.e. if they did not get removed during decimation.

Layout Constraints affect the topological structure of the IGM or, in terms of a quad mesh, its connectivity. For instance, the user can specify that a pair of singularities is connected by an isoline of the IGM (a sequence of edges of the implied quad mesh) [Myles et al. 2010], without prescribing the concrete path of the isoline (which, if desired, can be done using feature constraints instead).

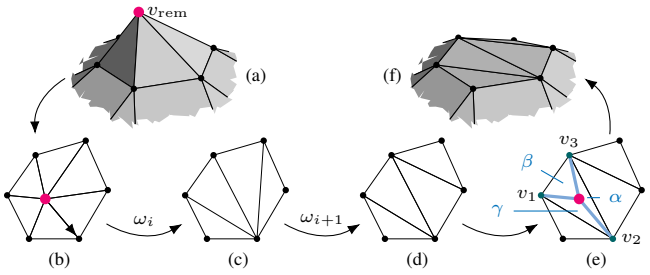


Figure 4: *Illustration of the atomic sub-operations performed in a decimation step. The local 1-ring (a) is parametrized (b) and a half edge collapse is performed (c). Next, edge flips are performed to establish a local Delaunay triangulation (d). For coarse-to-fine mapping purposes, barycentric coordinates of the removed vertex in the local domain are determined and stored (e). This is similar in spirit to the MAPS approach [Lee et al. 1998], though more information needs to be stored and the coarse-to-fine mapping is more complex due to handling of singularities and transition functions.*

Cycle Constraints require isolines to form closed, cyclic curves in specified regions (similar to the above layout constraints without prescribing concrete curves), preventing the local occurrence of undesirable helices [Bommes et al. 2011]. Such constraints affect the guiding field computation in form of holonomy constraints [Ray et al. 2008] and the parametrization in form of cyclic layout constraints (not necessarily involving a singularity) [Campen et al. 2016].

Being of topological nature, layout constraints and cycle constraints do not restrict the space of acceptable decimations. Nevertheless, they are expressed in terms of mesh entities (dual edge paths) and thus must be translated to the coarse level.

5 Decimation Method

To obtain coarse versions of the input mesh, we make use of the generic incremental decimation framework [Kobbelt et al. 1998]:

1: **Input:** triangle mesh $M^0 = M^{\text{fine}}$
2: $i \leftarrow 0$
3: **while** target complexity not reached **do**
4: $\quad v_{\text{rem}} \leftarrow \arg\min_v E^i(v)$
5: $\quad M^{i+1} \leftarrow \text{DECIMATE}(M^i, v_{\text{rem}})$
6: $\quad i \leftarrow i + 1$

By checking the link conditions [Dey et al. 1998] before choosing a vertex for removal, we can ensure that the surface of the decimated mesh is topologically equivalent to the original.

As the atomic operation DECIMATE we choose the common vertex removal operation [Lee et al. 1998] that retriangulates the 1-ring (determined as the Delaunay triangulation within the discrete geodesic polar parametrization of the 1-ring [Welch and Witkin 1994]), as illustrated in Figure 4. However, we express this operation as a half edge collapse followed by a sequence of edge flip operations instead of vertex removal and retriangulation. This is crucial for our purpose, because it enables us to consistently keep track of transitions when performing the coarse-to-fine-mapping after the IGM computation on the coarse mesh, cf. Section 6.

Regarding the vertex order taken by the incremental decimation approach (dictated by the cost function E chosen in line 4), our goal is to produce a mesh with a very low number of vertices in order to heavily reduce the run time of the IGM computation, while at the same time ensuring that the IGM computed on the coarse mesh is reasonably close to what it would look like if it was computed on the fine input mesh. We will see in the following that common cost function choices, e.g. the well-known error quadric metric or objectives promoting uniform edge lengths, are not well-suited for our specific purpose. In particular, note that geometric shape approximation in terms of Hausdorff distance is not a direct goal of decimation in our context.

In Section 5.1 we introduce two cost functions E, in Section 5.2 we detail how the user-specified design constraints are taken into account during decimation, and in Section 5.3 we introduce two techniques to further accelerate interactive updates to the IGM.

5.1 Decimation Objective

Modern IGM parametrization algorithms rely on cross or frame field generation methods (cf. Section 2) for the geometry aware determination of singularities. Their placement is, in virtually all automatic methods, fundamentally influenced by the surface's Gaussian curvature distribution. We thus propose cost functions to be used in the decimation order determination that penalize changes in local Gaussian curvature:

$$E^i_{\mathrm{G}}(v) = |K^i(v)| + \sum_{v' \in N_1(v)} |K^{i+1}(v') - K^i(v')|$$

$$E^i_{\mathrm{G,mem}}(v) = |K^0(v)| + \sum_{v' \in N_1(v)} |K^{i+1}(v') - K^0(v')|$$

where $K^i(v)$ is the Gaussian curvature at vertex v in mesh M^i and $N_1(v) \subseteq V$ is the 1-ring neighborhood of v. K^{i+1} is the tentative Gaussian curvature if M^{i+1} is obtained from M^i by removal of the vertex v as the next step. Note that the tentative Gaussian curvature at the location of v itself is zero, hence the simpler first term. The difference between E_{G} and $E_{\mathrm{G,mem}}$ is that the latter *has a memory*: the Gaussian curvature of the original mesh $M^0 = M^{\text{fine}}$ is always taken as reference, rather than the current state M^i.

In Section 9, using the measures introduced in Section 8, we compare the performance of these Gaussian curvature based cost functions. We also compare to other common decimation cost functions: the quadric error metric E_{QEM} [Garland and Heckbert 1997] (adapted to our setting which requires the coarse mesh to be a subsampling of the fine mesh, i.e. rating and performing half edge collapses instead of edge-collapses) and a uniform edge length objective E_{uni} (cf. Section 5.3).

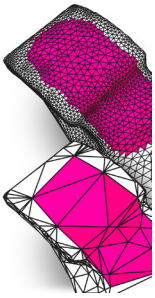
5.2 Constraint Awareness

In order to, during decimation, preserve sufficient degrees of freedom (i.e. vertices) to properly enforce the user-specified constraints we handle them specially during the decimation.

Index Constraints As indicated in Section 4, vertices that are constrained to a certain (non-zero) cross field index have to remain part of the decimated mesh. We thus simply exclude them from being selected for removal during decmation.

Feature Constraints Feature vertices (i.e. vertices incident to one or more than two edges marked as feature edges) are likewise preserved. Vertices within feature edges (i.e. vertices incident to two feature edges) are only allowed to collapse along one of their incident feature edges; but we completely disallow their removal if the feature edge meets any of the discontinuity curve criteria formulated in [Hoppe 1996]. Further, when restoring the local Delaunay criterion through edge flips we disallow flipping feature edges and thus establish a constrained local Delaunay triangulation.

Guiding Constraints Direction fields defined in certain mesh regions (e.g. specified by strokes) are used to guide the edge flow. In terms of decimation decisions, we treat edges separating these regions from one another and from the rest of the mesh like described above for feature edges.



In addition, in order to avoid guiding regions converging into a single point, we do not collapse a *corner* of a guiding region boundary (a vertex where the inner angle of the guiding region is less than $\frac{3}{4}\pi$), unless this collapses two corners closer than the IGM target edge length s (to not preserve small scale noise on the region boundaries).

The guiding direction field in the regions is easily transferred to the coarsened mesh during each decimation step via the planar 1-ring parametrizations (cf. Figure 4): a coarse triangle in the decimated 1-ring adopts the direction of the fine triangle containing its barycenter. More complex interpolation strategies with higher fidelity are imaginable, but this simple approach proved sufficient.

5.3 Acceleration of Re-Decimation

Intermediate Level

In an interactive workflow, users typically iteratively modify the set of guiding, index, layout and cycle constraints to explore the IGM design space and adjust the resulting mesh. Layout and cycle constraints can be applied directly at the minimally sampled decimation level and thus an updated IGM can be computed with a minimum delay.

For index and guiding constraints, however, the coarsest decimation level may not provide enough degrees of freedom. Consequently, a constraint aware re-decimation (cf. Section 5.2) is necessary.

On the other hand, index and guiding constraints do not need a granularity significantly finer than the quad mesh density (specified by the target edge length s). We exploit this observation by introducing the *redundancy free* intermediate level into the decimation hierarchy as outlined in Section 3. In order to distribute the vertices uniformly across the surface, we perform a uniform decimation from the fine to the intermediate level using the cost function

$$E_{\mathrm{uni}}(v) = \min_{v' \in N_1(v)} \|\mathbf{x}(v) - \mathbf{x}(v')\|$$

with the embedding of the mesh $\mathbf{x} : V \to \mathbb{R}^3$. We terminate the decimation as soon as $E_{\mathrm{uni}}(v^*) > s/2$ for the minimizer v^* of E_{uni}. This way we yield a sufficiently uniform tessellation with edge lengths just above $s/2$.

This intermediate level is then further decimated using E_G or $E_{\mathrm{G,mem}}$ to obtain the *minimally sampled* coarse mesh. Decimating to a constant complexity of 1000 vertices proved to be a suitable speed/quality trade-off for all of our experiments. Figure 5 displays an example of the decimation results.

Figure 3 shows starting from which level re-decimation needs to be performed if a certain type of constraint is modified or added. We thus have three tiers of constraints: layout and cycle constraints allow for the fastest feedback times, guiding and index constraints incur a small penalty for re-decimation starting from the intermediate level, and changes to feature constraints require a full re-decimation. Note that the set of sharp surface features the quad mesh edges should be strictly aligned to is often clear and known in advance, rather than to be discovered interactively, such that the impact of this circumstance is not even of relevance in practice according to our experience.

Cost Function Cache

Energy E_{G} is not trivial to compute: in order to compute the Gaussian curvature *after* a tentative decimation operation, the operation actually has to be performed. This involves computing the local planar parametrization and establishing the local Delaunay state as laid out in Section 5.

In interactive settings, where constraints are added iteratively, the second decimation stage is performed repeatedly. While everytime it is performed with different constraints it converges towards a different result (cf. Section 5.2) the vast majority of evaluations of E_{G} are performed on 1-ring neighborhoods that occurred in previous iterations already. We exploit this observation and gain a speed-up

Figure 5: *The three decimation levels of a mesh. The input mesh (left) has 346k faces, the redundancy free mesh (center) has 29k faces and the minimally sampled mesh (right) has 1000 vertices. Note how the guiding regions defined in the redundancy free mesh are preserved in the minimally sampled mesh.*

(up to a factor of 4 in our experiments) by caching values of E_{G} in a hash table. The hash key needs to be a unique description of the 1-ring (e.g. the index of the center vertex followed by a clockwise enumeration of its neighbors starting with the smallest index).

In the case that $E_{\mathrm{G,mem}}$ is used, caching is less attractive: only if center vertex v, 1-ring, and 2-ring vertices occur in the same configuration in two decimation sequences, $E_{\mathrm{G,mem}}(v)$ is the same in both cases. The likelihood of cache misses is thus significantly higher.

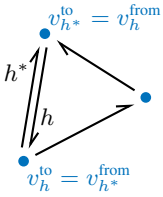
6 Coarse-to-Fine Mapping

Given the (fine) input mesh M_{fine} and the (coarse) mesh M_{coarse} generated by our decimation method, we need to faithfully map any global parametrization of M_{coarse} onto M_{fine} in a way that preserves the consistency of singularities, transition functions and charts.

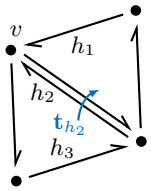
Before we explain how our coarse-to-fine mapping approach reaches this goal we need to introduce some notation.

6.1 Notation

Meshes We work with a half-edge based triangle mesh representation $M = (V, H, E, F)$, a tuple containing the set of vertices V, directed half edges H, edges E and triangles F. We specify individual vertices, half edges and edges using lower case italics. Given a half edge $h \in H$, v_h^{to} and $v_h^{\mathrm{from}} \in V$ denote the vertex h is pointing to and originating from, respectively. $h^* \in H$ denotes the half edge opposite to h (i.e. $v_h^{\mathrm{to}} = v_{h^*}^{\mathrm{from}}$), and $e_h = e_{h^*} \in E$ denote the non-directed edge represented by h and h^*. As usual, in an oriented 2-manifold mesh, each half edge can be associated with one of its two incident faces (by convention: the one in which it is oriented in a counter-clockwise manner).



Parametrizations The integer grid maps we are going to map across meshes take the form of piecewise linear parametrizations of the triangle mesh with transitions across cuts. They can be defined through a map which assigns a point in the parameter domain (UV coordinates) to each triangle corner. As every triangle corner can be identified with the unique half edge pointing to it, this can be expressed as a map $\mathbf{f} : H \to \mathbb{R}^2$ of the half edges $h \in H$ into the parameter domain. Consequently, the same vertex $v = v_{h_1}^{\mathrm{to}} = v_{h_2}^{\mathrm{to}}$ can, in general, have different parametrizations in different faces, i.e. $\mathbf{f}(h_1) \neq \mathbf{f}(h_2)$. This allows the parametrization to consist of multiple charts and be non-continuous at the chart boundaries. For instance, a parametrization of the triangles depicted on the right is discontinuous across edge e_{h_2} if $\mathbf{f}(h_1) \neq \mathbf{f}(h_2)$ or $\mathbf{f}(h_3) \neq \mathbf{f}(h_2^*)$, i.e. if there are two images of e_{h_2} in the parameter domain. We associate an affine transition function $\mathbf{t}_h : \mathbb{R}^2 \to \mathbb{R}^2$ with every half edge h, $\mathbf{t}_h = \mathbf{t}_{h^*}^{-1}$, that maps between the two images of the edge e_h. Consequently, in the example above $\mathbf{t}_{h_2}(\mathbf{f}(h_2)) = \mathbf{f}(h_1)$ and $\mathbf{t}_{h_2}(\mathbf{f}(h_3)) = \mathbf{f}(h_2^*)$. For the sake of legibility we occasionally use the notation $\mathbf{t}_i$ for $\mathbf{t}_{h_i}$ and $\mathbf{t}_{i^*}$ for $\mathbf{t}_{h_i^*}$ when it does not introduce ambiguities. Whenever we make an assignment to a transition function $\mathbf{t}_h$ it is implied that we assign the inverse function to the opposite transition function $\mathbf{t}_{h^*} \coloneqq (\mathbf{t}_h)^{-1}$.



The conditions on the transitions that restrict the space of general chart-based parametrizations to that of integer grid maps are detailed by Bommes et al. [2013].

6.2 Setting

As indicated in Section 5 we obtain the coarse mesh M_{coarse} from the input mesh M_{fine} through a series of decimation steps. Each decimation operation removes one vertex and re-triangulates the resulting hole. As illustrated in Figure 4 this operation can be broken up into several atomic sub-operations: a half-edge collapse followed by zero or more edge flips.

Thus, in effect our incremental decimation approach creates a series of meshes

$$M_{\mathrm{fine}} = M^0 \xrightarrow{\omega_1} M^1 \xrightarrow{\omega_2} \dots \xrightarrow{\omega_n} M^n = M_{\mathrm{coarse}}$$

through fine-to-coarse operations ω_i that are either half-edge collapses or edge flips. We then compute a parametrization $\mathbf{f}_{\mathrm{coarse}}$ on M_{coarse} and, through coarse-to-fine operations $\bar{\omega}_i$, create a series of derived parametrizations $\mathbf{f}^i$ that represent the initial parametrization on increasingly finer meshes M^i:

$$\mathbf{f}_{\mathrm{coarse}} = \mathbf{f}^n \xrightarrow{\bar{\omega}_n} \mathbf{f}^{n-1} \xrightarrow{\bar{\omega}_{n-1}} \dots \xrightarrow{\bar{\omega}_1} \mathbf{f}^0 = \mathbf{f}_{\mathrm{fine}}.$$

The two decimation operations can naturally be defined to logically only discard existing but never introduce new half-edges (cf. Figure 6 and 7) so that $\mathrm{dom}(\mathbf{f}_{\mathrm{coarse}}) = H_{\mathrm{coarse}} \subseteq H_{\mathrm{fine}} = \mathrm{dom}(\mathbf{f}_{\mathrm{fine}})$ and, in general, $\mathrm{dom}(\mathbf{f}^i) \subseteq \mathrm{dom}(\mathbf{f}^{i-1})$.

Note that these fine-to-coarse and coarse-to-fine operators are defined in an asymmetric fashion: while the fine-to-coarse operators ω transform a fine *mesh* into a coarse one, the coarse-to-fine operators $\bar{\omega}$ transform a coarse *parametrization* into a fine one. The reason for this asymmetry is that in order to yield minimal round trip times (and thus maximum interactivity) it is favorable to transform the parametrization $\mathbf{f}_{\mathrm{coarse}}$ of M_{coarse} into $\mathbf{f}_{\mathrm{fine}}$ of M_{fine} without explicitly reconstructing intermediate decimation levels M^i of the mesh —only M_{coarse} and M_{fine} need to be available.

Transition Functions Note that, since the $\mathbf{f}^i$ imply transition functions $\mathbf{t}_h^i$, there is no need to explicitly map the transition functions from $\mathbf{f}_{\mathrm{coarse}}$ to $\mathbf{f}_{\mathrm{fine}}$. However, since explicit transition functions are a by-product of most IGM computation methods while extracting them from a parametrization suffering from numerical inaccuracies is not trivial (cf. [Ebke et al. 2013]) we explain how to implement an $\bar{\omega}$ operation that maps both, $\mathbf{f}$ and $\mathbf{t}$ in the following.

6.3 Coarse-to-Fine Operators

The effect of the flip operator (which transforms a "finer" mesh M^i into a "coarser" mesh M^{i+1}) is illustrated in Figure 6. The *un*-flip

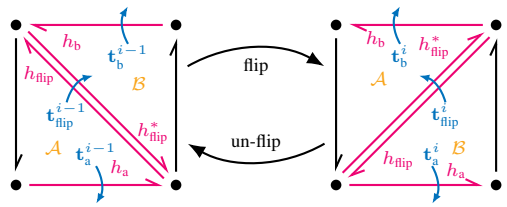


Figure 6: *The edge flip operator reconnects the diagonal half edges h_{flip} and h^*_{flip}, turning them counter-clockwise. Given a parametrization of the decimated mesh, the incident triangles are in (potentially different) charts $\mathcal{A}$ and $\mathcal{B}$ with transition function $\mathbf{t}_{\text{flip}}$ mapping between them. In order to yield a consistent parametrization, we define that the inverse operator makes the charts turn counter-clockwise and adjust the affected transition functions $\mathbf{t}_a, \mathbf{t}_b$, and $\mathbf{t}_{\text{flip}}$, as well as the parametrization of the affected half edges $h_{\text{a}}, h_{\text{b}}$, and h_{flip} accordingly.*

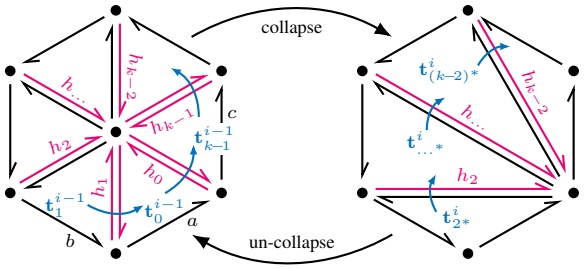


Figure 7: *The half edge collapse operator removes a vertex and three pairs of incident half edges while reconnecting the remaining incident half edges to one of its adjacent vertices. Consequently, the inverse collapse operator updates the parametrization of the half edges pointing to the removed vertex and of those half edges that were removed. In addition, consistent transition functions for the three removed edges are computed.*

operator (which transforms a parametrization $\mathbf{f}^i$ of a coarser mesh M^i into a parametrization $\mathbf{f}^{i-1}$ of a finer mesh M^{i-1}) permutes the parametrization (or UV coordinates) of the involved half edges and adjusts the transition functions accordingly.

Figure 7 illustrates the effect of the half edge collapse operator ($M^{i-1} \xrightarrow{\omega_i} M^i$). The *un*-collapse operator ($\mathbf{f}^i \xrightarrow{\bar{\omega}_i} \mathbf{f}^{i-1}$) interpolates the parametrization for the removed center vertex using the stored barycentric coordinates, generates parametrizations for the two collapsed triangles and computes consistent transition functions for the three collapsed edges.

Details on the implementation of the un-flip and un-collapse operators for IGMs are given in Appendix A.

6.4 Decimation Log

The information on what sequence of un-flip and un-collapse operations need to be performed is contained in the decimation log recorded during decimation. It is a list of n records corresponding to the operations ω_i. There are two types of records corresponding to the two operations: a flip record and a collapse record. A flip record stores pointers to the corresponding half edges h_{flip}, h_{a}, h_{b}, the collapse record stores pointers a, b, c, h_0, $h_1, \ldots, h_{k-1}$, as well as the interpolation information: local vertex indices i_α, i_β, i_γ, and barycentric coordinates α, β, and (implicitly) $\gamma = 1 - \alpha - \beta$ (cf. Figure 14). Note that this representation is related to the split records of progressive meshes [Hoppe 1996] but differs in being half edge instead of vertex based. This is crucial in order to properly handle the mapping of chart-based parametrizations with transitions. The interpolation vertices and weights are determined during decimation by intersecting the removed vertex with the new triangulation of its 1-ring in its local parameter domain and computing its barycentric coordinates within the intersecting triangle (cf. Figure 4 (e)). A detailed description of an efficient storage format for the decimation log is given in Appendix B.

6.5 Atlas Simplicity

While the parameter domain generated by state-of-the-art IGM based quad meshing methods is relatively simple (usually one connected, relatively compact (self-overlapping) chart), during the coarse-to-fine mapping the chart atlas can become arbitrarily fragmented with multiple charts and many superfluous non-identity transitions. In the context of quad meshing the complexity of the atlas is entirely irrelevant, as the implied integer grid is oblivious. If, however, a downstream application benefits from a simple domain, (e.g. in the context of texture mapping) the charts can easily be rearranged and combined using the disk growing approach laid out in [Bommes et al. 2009].

7 Complexity Considerations

Using our decimation and coarse-to-fine mapping framework, we can achieve linear run time and memory complexity.

Memory Complexity The only data generated by our algorithm is the decimation log and (optionally) the hash table used to cache results of expensive energy computations. The size of both of these is linear in the number of decimated vertices. Since we decimate to a constant number of vertices, the memory complexity of any IGM computation method is constant.

Run Time Complexity The run time complexity of the IGM computation is constant for the same reason. Our incremental decimation scheme performs less than $|V|$ decimation operations each of which consists of one half edge collapse and an amortized constant number of edge flips. Both of these operators have (amortized) constant run time and so do their inverse counterparts. If the selection of the next vertex to remove is implemented using a priority queue, it has $O(\log |V|)$ complexity. We can, however, resort to a constant-time stochastic implementation (draw c random vertices and pick the one with the lowest energy) until $|V|$ goes below a fixed threshold (e.g. 10^6) without a significant loss in result quality, thus achieving an overall linear run time.

Note that the benefit of the linear run time complexity is merely of a theoretical nature. The constant factor in the necessary $|V|$ evaluations of the decimation energies E_1, E_2, albeit small on an absolute scale, is much larger than the one in the priority queue updates. As a result, on all meshes used in our experiments (none of which had more than 10M vertices) the relative savings of the stochastic implementation are minuscule (on small meshes even negative) and thus the $O(|V| \log |V|)$ approach with a priority queue was used.

8 Evaluating IGM Similarity

Our method is intended as an acceleration add-on to existing IGM computation methods. Consequently, more importantly than being generically of an overall high quality, an IGM produced using our approach should specifically be close to the IGM produced without our approach subject to identical design constraints. Unfortunately, there are no generally accepted, universally meaningful measures for the similarity of two IGMs. In particular, note that evaluating

some kind of parametrization difference (e.g. based on the eigenvalues or eigenvectors of the Jacobian) in a triangle by triangle manner is not adequate: for instance, introducing an additional pair of close-by singularities with opposite (arbitrarily extreme) indices may have an arbitrarily small effect on the parametrization (if they are close enough); however, such superfluous singularities, singularities with extreme indices, as well as singularities very close to each other are all very unfavorable and even problematic in the context of mesh generation.

We propose to consider the following measures instead, which capture properties of an IGM that are deemed important for artist guided quad meshing:

- number, type, and location of singularities (irregular vertices),
- distortion of the parametrization (the shape of the quads).

Irregular Vertices correspond to metric cones [Myles and Zorin 2012] in the IGM which in turn correspond to singularities [Ray et al. 2008] in the guiding cross field. The difference between two IGMs on a common surface in terms of irregular vertex configuration can thus be quantified as the distance between the Gaussian curvature distributions in the corresponding cone metrics, or equivalently (up to a factor 2π) as the distance between the index distributions in the underlying cross fields. Being distributions (with constant integral, determined by the genus), the 1st Wasserstein (*earth mover's*) distance (EMD) is an appropriate choice. Intuitively, it quantifies how much effort is needed to transform one configuration into another one by means of moving singularities over the surface (including splits and merges where necessary), according to the optimal transport plan. As the distributions under consideration are collections of scaled Dirac impulses, we are effectively facing a discrete setting in which the distance is easily computed [Rubner et al. 1998].

In our context of quad mesh generation, however, this measure has a shortcoming: only moving is associated with a cost, not the splitting or merging of singularities that might happen implicitly. A low EMD is thus a required but not a sufficient indicator for irregular vertex configurations being similar. For instance, as mentioned above, the appearance of an additional pair of nearby singularities with indices $+\frac{i}{4}$ and $-\frac{i}{4}$ has an arbitrarily small effect on the EMD (depending on their distance on the surface). Even clusters of hundreds of additional singularities (of arbitrarily extreme indices), certainly leading to unacceptable or even invalid quad meshes, potentially remain undetected by the EMD alone.

One could associate additional costs with splits and merges, but there is no obvious natural choice of cost relative to moving. We thus abstain from consolidating both aspects into a single value and instead consider the singularity index histogram (or the irregular vertex valence histogram) in addition to the distribution distance, allowing to judge both aspects separately.

Sizing, Anisotropy, and Shearing In order to judge the similarity of two IGMs with respect to sizing, anisotropy, and shearing, we measure these quantities in both IGMs per face, recalling that IGMs are piecewise linear. We then study the differences between the distributions based on histograms to judge the similarity.

Given a linear map $\mathbf{g}$ mapping a face from the parameter domain onto the surface, we measure

- *sizing* as $\det(\nabla\mathbf{g})/s^2$, the ratio of the determinant of the Jacobian of $\mathbf{g}$ and the squared target edge length s,
- *anisotropy* as σ_1/σ_2, the larger singular value of $\nabla\mathbf{g}$ divided by the smaller one, and

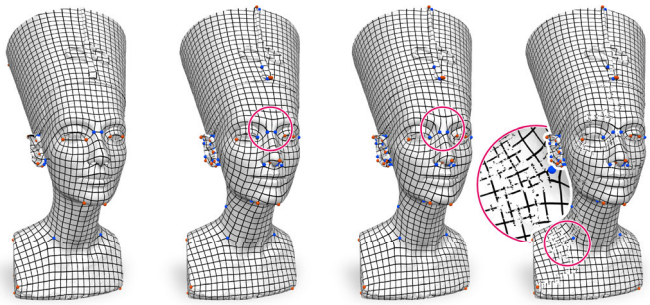


Figure 8: *Our pipeline used with (from left to right) [Ebke et al. 2014], [Bommes et al. 2009], [Kälberer et al. 2007] and a seamless parametrization (without integer rounding of translations and singularity parameters), each computed with the same set of alignment constraints. The magnification shows the fragmentation of the chart atlas (which can be defragmented if desired, cf. Section 6.5).*

- *shear* as the arc-sine of the scaled Jacobian of $\mathbf{g}$ (i.e. the (smaller) angle at which isolines cross locally).

In the following we will use these measures to evaluate the performance of different decimation strategies for our purpose.

9 Results

Setup To test our framework we implemented it in a single-threaded fashion and combined it with the parametrization method described in [Ebke et al. 2014] and a rudimentary UI to interactively specify constraints. This setup was used for our experiments, but as we show in Figure 8, our framework can likewise be combined with other IGM and seamless parametrization computation methods. To judge the performance we determined two timings: the one-off initialization delay and the iterative delay incurred when updating the parametrization after constraints were added. The initial delay includes (1) the computation of the full decimation hierarchy, (2) the computation of an initial parametrization and (3) the coarse-to-fine mapping procedure of the parametrization onto the original mesh. The iterative delay contains (1) a repeated decimation from the redundancy free (intermediate) decimation level incorporating a new constraint set, (2) the computation of a new IGM and (3) the coarse-to-fine mapping procedure. The timings of steps (2) and (3) are almost identical between the steps. The delay for step (1) is much lower for the iterative updates because the first decimation stage is skipped and because the decimation priority cache is already filled.

Workflow The title picture (Figure 1, NEFERTITI model) is quite instructive as it shows several stages of a typical workflow where the user explores the space of possible quadrangulations by iteratively adding constraints and evaluating the instant visual feedback.

Timings In Table 1 we present a list of timings acquired in our experiments on an Intel Core i7-4770 processor. Note that the timings for the iterative update vary slightly, typically by less than 10%, due to the varying number of cache misses depending on how the constraint set is updated in each iteration. The timings provided in Table 1 are average timings. The reference timings obtained with [Ebke et al. 2014] *without* our framework are given as well.

In Figure 9 we present timings for the same input mesh when using different target edge lengths. While the target complexity of the minimally sampled (coarse) decimation level is constant, the complexity of the redundancy free (intermediate) decimation level depends on the chosen target edge length: if the target edge length

Model	$\lvert F\rvert$	t_{base}	Initialization				$E_{\text{G,mem}}$		E_{G}	
			t_{d}	t_{p}	t_{m}	Σ	t_{d}	Σ	t_{d}	Σ
ARMADILLO	346k	58	2.2	0.8	0.1	3.1	0.6	**1.5**	0.2	**1.1**
CHARLEMAGNE 5	1.2M	182	5.8	0.2	0.4	6.4	0.4	**1.0**	0.1	**0.7**
CHARLEMAGNE 2.5	1.2M	269	6.6	0.3	0.4	7.3	1.0	**1.7**	0.3	**1.0**
NEFERTITI	2M	318	10.3	0.3	0.7	11.3	1.1	**2.1**	0.3	**1.3**
DRAGON1	846k	161	4.4	1.0	0.3	5.7	0.9	**2.2**	0.2	**1.5**
DRAGON2	7.2M	>1h	40.0	6.0	2.5	48.5	3.0	**11.5**	0.8	**9.3**

Table 1: *Timings (in seconds) taken on meshes of different complexity. t_{base} are baseline timings obtained with [Ebke et al. 2014] running without our framework. The other timings (decimation t_{d}, parametrization t_{p} and coarse-to-fine mapping t_{m}) were obtained using our framework on top of the same parametrization method, once with $E_{G,mem}$ and once with E_{G} exploiting a cache. The one-off initialization costs are unaffected by the cache and thus identical for both energies. The columns printed in bold show the most important times: the total time for an update when the user changes guiding or index constraints (with same t_{p} and t_{m} (thus not repeated in the table), but shorter t_d due to the intermediate level of the hierarchy, cf. Section 5.3).*

gets shorter, the redundancy free level is more complex and thus more vertices are decimated in the second, more expensive decimation phase. As a consequence, the run time of our method is (to some extent) target edge length sensitive.

Decimation Strategies In Figures 10 and 13 we show how our Gaussian curvature based decimation order E_{G} performs in comparison to E_{QEM} (cf. Section 5.1). Figure 10 uses the ARMADILLO mesh to demonstrate that using E_{QEM}, the local Gaussian curvature gets concentrated on single vertices. This causes the guiding field algorithm to place singularities of very high indices (translating into quad mesh vertices of very high or very low valence) onto the mesh. Even indices corresponding to vertex valences of zero or below zero can be caused, which render the generation of a valid IGM or quad mesh impossible. With $E_{\text{G,(mem)}}$ by contrast, we get the common valence 3 and 5 vertices, just like on the fine mesh. Their number is sometimes increased, but no extreme valences are caused. It can be noticed that, on average (but not even consistently), $E_{\text{G,mem}}$ behaves marginally better than E_{G}, whereas the use of E_{G} (due to its efficient cacheability) decreases update delays by around 20%-40% in our experiements.

In Figures 9 and 12 we made use of different types of constraints to demonstrate the creation of IGMs with a specific desired layout just as an artist would. By contrast, in Figures 10 and 13 we refrained from using any index or layout constraints in order to judge the quality of the naturally arising singularity configurations of the different decimation methods.

Comparison The recent work by Jakob et al. [2015] (cf. Section 2) is related in that it allows for interactive generation of quad-dominant meshes. By performing one step of subdivision, pure quad meshes can be obtained from them. In Figure 11 we demonstrate the characteristic differences of the quad meshes obtained in this way.

10 Limitations and Future Work

While our strategy of decimating the minimally sampled mesh to a fixed number of vertices provides us with a linear run time in theory and, in practice, made us very successful in reaching round trip times that were fast enough for interactive quad meshing even on millions of triangles, we realize that a constant complexity threshold might not be ideal in all application scenarios. The investigation

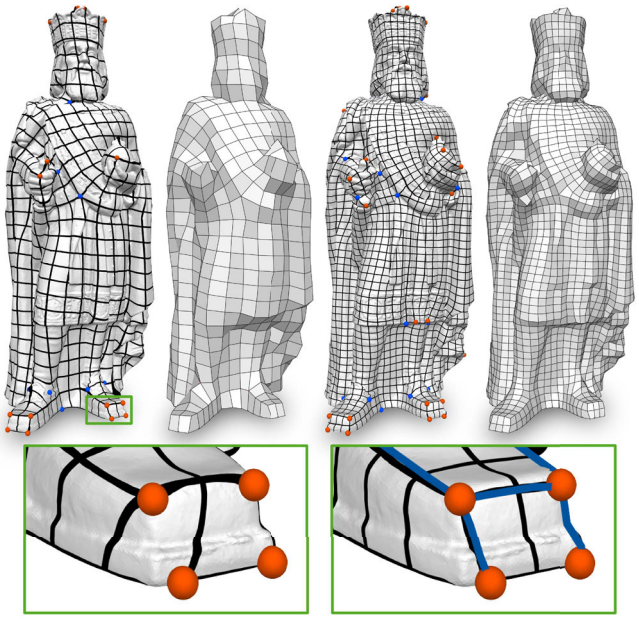


Figure 9: CHARLEMAGNE *(1.2M faces) remeshed with a target edge length of 5 (left) and 2.5 (right). Initialization took 6.4s and iterative reparametrization 0.7s with the larger target edge length. As expected, halving the target edge length affects both timings slightly (7.3s and 1.0s, respectively). The effect of layout constraints is demonstrated on the foot (see magnification).*

of decimation thresholds that directly take into consideration the quality of the guiding field on the decimated mesh is thus certainly of interest.

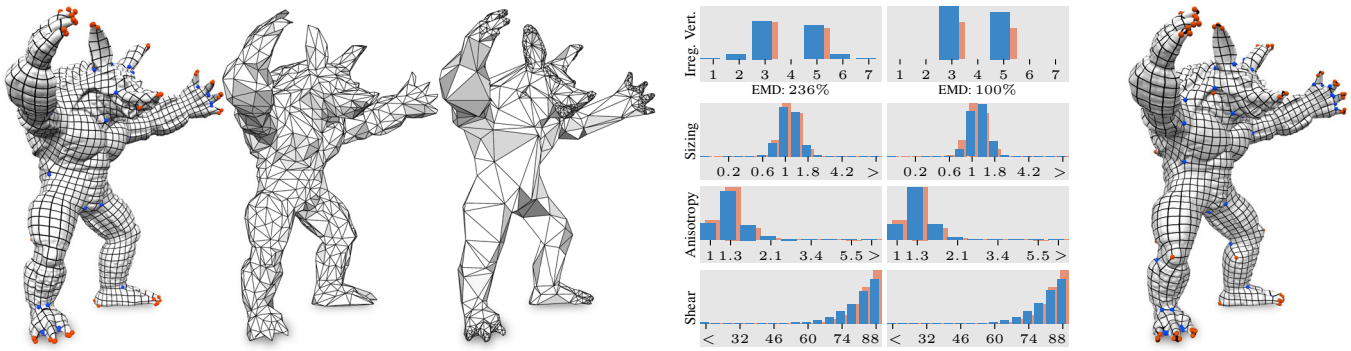
As demonstrated in the various figures above, especially Figure 12, the parametrizations resulting from our framework are only slightly more distorted than the ones computed without it. Yet, there is some additional distortion and it is mainly incurred due to the geometric distortion caused by the decimation. We think the possibility of (approximately) quantifying and accumulating this, and adjusting the underlying IGM computation method to compensate for it, is an interesting aspect for future work.

Furthermore, one can picture a hybrid application scenario when very high quality parametrizations are required: first, the design space is interactively explored using our framework. Once a good set of constraints is found, an IGM can be computed in a non-interactive post-process directly on the original mesh, using the same set of constraints.

While we presented a framework that enables an interactive work flow, we are not aware of any studies that systematically explore and evaluate UI metaphors specifically tailored to specifying the types of constraints commonly found in quad meshing methods. As these methods now become faster and faster and the interest in incorporating them into interactive applications grows, we think that it would be worthwhile exploring this avenue of research.

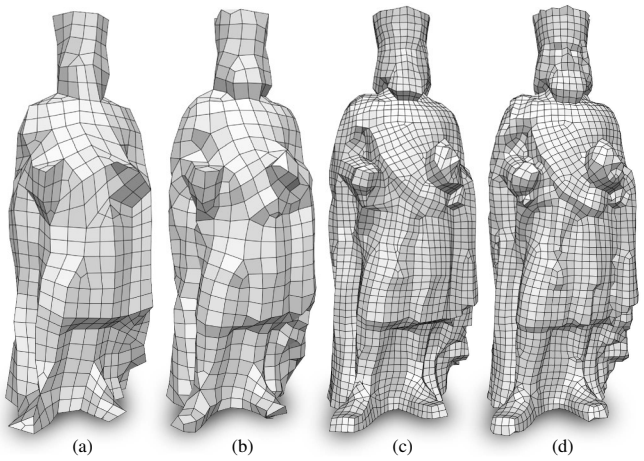
11 Conclusion

We presented a framework that drastically speeds up parametrization based quad meshing approaches. It is targeted at an interactive application scenario where the user relies on instant feedback when adding high level constraints to guide the edge flow and the layout of the quad mesh. The presented method operates by representing the complex input geometry in a hierarchical manner and propagat-



IGM computed on non-decimated mesh | decimation w/ [Garland and Heckbert 1997] | decimation with $E_{G,mem}$ | IGM properties using E_{QEM} | IGM properties using $E_{G,mem}$ | IGM computed on mesh decimated with $E_{G,mem}$

Figure 10: *The same mesh reduced to 1000 vertices with quadric error metric decimation and using our Gaussian curvature aware method. Note how features like the fingers and toes are approximated with single spikes with error quadric decimation. Since the entire Gaussian curvature of the finger tips gets concentrated on the spike, singularites of very low valence are created here. In particular, the irregular vertex valence histogram shows that singularities of valences 1, 2, 6, and 7 appear, which are not present in the reference IGM. With our $E_{G,mem}$ only irregular valences 3 and 5 are present. The similarity suggested by the histograms (the red shadows represent the reference solution created without decimation) can also be witnessed when comparing the reference IGM (far left) to the one created with our method (far right). The EMD is given in normalized form, relative to that of the $E_{G,mem}$ approach (100%), because the absolute value is of no relevance.*



(a) (b) (c) (d)

Figure 11: *For comparison to [Jakob et al. 2015]: results of the multi-threaded implementation provided by the authors, following a comparable set of constraints and to the same target complexities shown in Figure 9. The direct output (a, c) as well as a faired version (b, d; Laplacian smoothing with reprojection) is shown. As suggested by the authors, one step of subdivision was used to obtain a pure quad mesh instead of a quad-dominant mesh. The interaction delay when adding guiding constraints was 2.7s on this model (0.7s or 1.0s with our method). The quad meshes have 111 and 274 irregular vertices (38 and 60 with our method, cf. Figure 9), including some of uncommon valences 2, 6, and 7.*

ing parametrizations from the coarse to the fine level. The resulting parametrization is defined on the original input mesh, enabling downstream applications to operate on the full resolution surface.

We showed how the remaining degrees of freedom in the coarse representation can be chosen so that the underlying parametrization method can achieve low distortion parametrizations that satisfy the provided constraints and how to tackle the challenge of maintaining consistency of the parametrization across hierarchy levels in the presence of charts, transition functions, and singularities.

The fact that our method is oblivious to the underlying parametrization based quad meshing method as well as to the user interface used to generate the user provided constraints makes this framework useful for a broad range of quad meshing scenarios.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement n° [340884], and the German Research Foundation (DFG, Gottfried-Wilhelm-Leibniz Programm). The software prototype used for evaluation was implemented on top of the OpenFlipper geometry processing framework [Möbius and Kobbelt 2012]. The authors would like to thank Hannes Hergeth for implementing early experiments that shaped this research, as well as the reviewers for their insightful comments.

References

AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. 2005. Multilevel solvers for unstructured surface meshes. *SIAM Journal on Scientific Computing 26*, 4, 1146–1165.

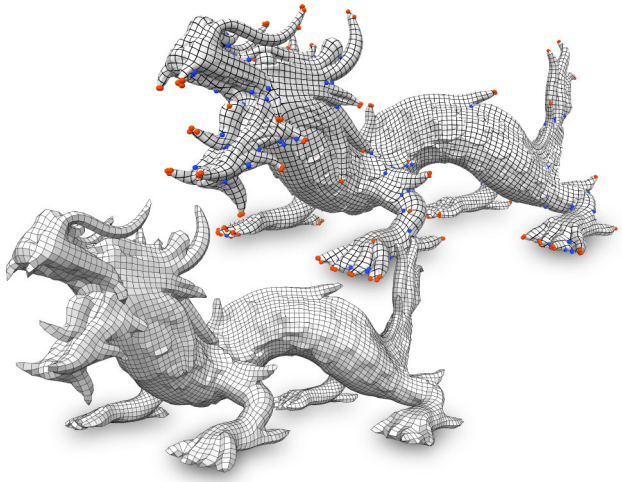
BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Transactions on Graphics 28*, 3, 77:1–77:10.

BOMMES, D., LEMPFER, T., AND KOBBELT, L. 2011. Global structure optimization of quadrilateral meshes. *Computer Graphics Forum 30*, 2, 375–384.

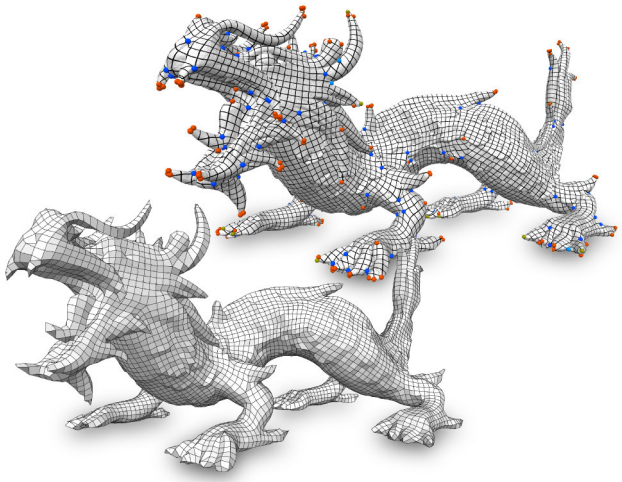
BOMMES, D., CAMPEN, M., EBKE, H.-C., ALLIEZ, P., AND KOBBELT, L. 2013. Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics 32*, 4, 98:1–98:12.

CAMPEN, M., AND KOBBELT, L. 2014. Quad layout embedding via aligned parameterization. *Computer Graphics Forum 33*, 8, 69–81.

CAMPEN, M., BOMMES, D., AND KOBBELT, L. 2015. Quantized global parametrization. *ACM Transactions on Graphics 34*, 6, 192:1–192:12.



[Ebke et al. 2014] used standalone (>60 minutes)



[Ebke et al. 2014] with our framework (iterative remeshing: 9s)

Figure 12: DRAGON2 *(7.2 million faces) remeshed without our framework (top) and with our framework (bottom). Observe how the singularities are similar in number and position on both meshes. Also observe how the larger number of degrees of freedom available to the parametrization algorithm when used without our framework allows for a slightly more uniform parametrization. However, the degradation in performance is severe: using our method the initialization cost is 49s, iterative remeshing takes 9s. Without our framework, every remeshing iteration takes more than 60 minutes.*

CAMPEN, M., IBING, M., EBKE, H.-C., ZORIN, D., AND KOBBELT, L. 2016. Scale-invariant directional alignment of surface parametrizations. *Computer Graphics Forum 35*, 5.

COHEN, J., OLANO, M., AND MANOCHA, D. 1998. Appearance-preserving simplification. In *Proc.*, SIGGRAPH '98, 115–122.

DANIELS II, J., LIZIER, M., SIQUEIRA, M., SILVA, C., AND NONATO, L. 2011. Template-based quadrilateral meshing. *Computers & Graphics 35*, 3, 471 – 482.

DEY, T. K., EDELSBRUNNER, H., GUHA, S., AND NEKHAYEV, D. V. 1998. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) 66*, 23–45.

DIAMANTI, O., VAXMAN, A., PANOZZO, D., AND SORKINE-HORNUNG, O. 2014. Designing N-PolyVector fields with complex polynomials. *Computer Graphics Forum 33*, 5, 1–11.

EBKE, H.-C., BOMMES, D., CAMPEN, M., AND KOBBELT, L. 2013. QEx: Robust quad mesh extraction. *ACM Transactions on Graphics 32*, 6, 168:1–168:10.

EBKE, H.-C., CAMPEN, M., BOMMES, D., AND KOBBELT, L. 2014. Level-of-detail quad meshing. *ACM Transactions on Graphics 33*, 6, 184:1–184:11.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH '97*, 209–216.

GELFAND, N., AND GUIBAS, L. J. 2004. Shape segmentation using local slippage analysis. In *Proc. Symp. Geometry Processing*, SGP '04, 214–223.

HOPPE, H. 1996. Progressive meshes. In *Proc. SIGGRAPH '96*, 99–108.

HOPPE, H. 1999. New quadric metric for simplifying meshes with appearance attributes. In *Proc. VIS '99*, 59–66.

HORMANN, K., GREINER, G., AND CAMPAGNA, S. 1999. Hierarchical parametrization of triangulated surfaces. In *Proc. Vision, Modeling, and Visualization 1999*, 219–226.

HUBELI, A., AND GROSS, M. 2001. Multiresolution feature extraction for unstructured meshes. In *Proc. VIS '01*, 287–294.

JAKOB, W., TARINI, M., PANOZZO, D., AND SORKINE-HORNUNG, O. 2015. Instant field-aligned meshes. *ACM Transactions on Graphics 34*, 6, 189:1–189:15.

JIANG, T., FANG, X., HUANG, J., BAO, H., TONG, Y., AND DESBRUN, M. 2015. Frame field generation through metric customization. *ACM Transactions on Graphics 34*, 4, 40:1–40:11.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. QuadCover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3, 375–384.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics 22*, 3, 350–357.

KNÖPPEL, F., CRANE, K., PINKALL, U., AND SCHRÖDER, P. 2013. Globally optimal direction fields. *ACM Transactions on Graphics 32*, 4, 59:1–59:10.

KNUPP, P. 1995. Mesh generation using vector fields. *J. Comput. Phys. 119*, 1, 142–148.

KOBBELT, L., CAMPAGNA, S., AND PETER SEIDEL, H. 1998. A general framework for mesh decimation. In *Proc. Graphics Interface*, 43–50.

LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. 1998. Maps: Multiresolution adaptive parameterization of surfaces. In *Proc.*, SIGGRAPH '98, 95–104.

LI, Y., LIU, Y., AND WANG, W. 2015. Planar hexagonal meshing for architecture. *IEEE Transactions on Visualization and Computer Graphics 21*, 1.

LIU, B., TONG, Y., GOES, F. D., AND DESBRUN, M. 2016. Discrete connection and covariant derivative for vector field analysis and design. *ACM Transactions on Graphics 35*, 3, 23:1–23:17.

MARCIAS, G., PIETRONI, N., PANOZZO, D., PUPPO, E., AND SORKINE-HORNUNG, O. 2013. Animation-aware quadrangulation. *ACM Transactions on Graphics 32*, 5.

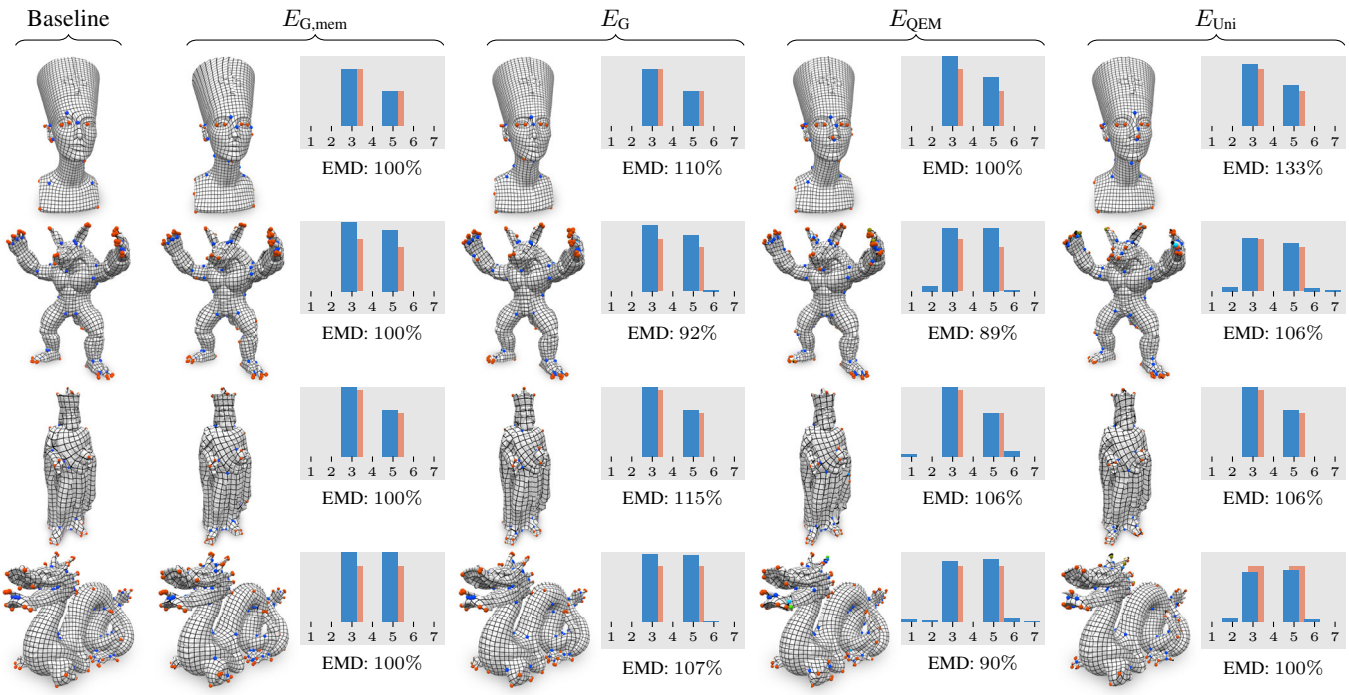


Figure 13: *Evaluation of IGMs created with our method using different decimation cost functions on (from top to bottom):* NEFERTITI, ARMADILLO, CHARLEMAGNE, *and* DRAGON1. *The baseline solution (IGMs created without decimation at several orders of magnitude slower run times) is represented as red "shadows" in the histograms in order to judge how faithful its characteristics are captured using our method. The singularity histogram is most important here, since it reveals whether irregular vertices of extreme valences are created. The EMD, i.e. the cost to transform the reference singularity configuration into the ones generated using our framework show the performance of the decimation orders relative to $E_{G,mem}$. The corresponding distortion histograms can be found in the supplemental material.*

MYLES, A., AND ZORIN, D. 2012. Global parametrization by incremental flattening. *ACM Transactions on Graphics 31*, 4, 109:1–109:11.

MYLES, A., AND ZORIN, D. 2013. Controlled-distortion constrained global parametrization. *ACM Transactions on Graphics 32*, 4, 105:1–105:14.

MYLES, A., PIETRONI, N., KOVACS, D., AND ZORIN, D. 2010. Feature-aligned T-meshes. *ACM Transactions on Graphics 29*, 4, 117:1–117:11.

MÖBIUS, J., AND KOBBELT, L. 2012. Openflipper: An open source geometry processing and rendering framework. In *Curves and Surfaces*, vol. 6920 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 488–500.

NIESER, M., PALACIOS, J., POLTHIER, K., AND ZHANG, E. 2012. Hexagonal global parameterization of arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics 18*, 6.

PALACIOS, J., AND ZHANG, E. 2007. Rotational symmetry field design on surfaces. *ACM Transactions on Graphics 26*, 3.

PANOZZO, D., PUPPO, E., TARINI, M., AND SORKINE-HORNUNG, O. 2014. Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Transactions on Graphics 33*, 4, 134:1–134:11.

PIETRONI, N., TARINI, M., AND CIGNONI, P. 2010. Almost isometric mesh parameterization through abstract domains. *IEEE Transactions on Visualization and Computer Graphics 16*, 4, 621–635.

PIETRONI, N., TARINI, M., SORKINE, O., AND ZORIN, D. 2011. Global parametrization of range image sets. *ACM Transactions on Graphics 30*, 6.

RAY, N., AND LÉVY, B. 2003. Hierarchical least squares conformal map. In *Proc. Pacific Conference on Computer Graphics and Applications, 2003*, 263–270.

RAY, N., LI, W. C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic global parameterization. *ACM Transactions on Graphics 25*, 4, 1460–1485.

RAY, N., VALLET, B., LI, W. C., AND LÉVY, B. 2008. N-symmetry direction field design. *ACM Transactions on Graphics 27*, 2, 10:1–10:13.

RAY, N., NIVOLIERS, V., LEFEBVRE, S., AND LÉVY, B. 2010. Invisible seams. In *Proc. EGSR'10*, 1489–1496.

RUBNER, Y., TOMASI, C., AND GUIBAS, L. J. 1998. A metric for distributions with applications to image databases. In *Proc. ICCV '98*, 59–66.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proc. SIGGRAPH '01*, 409–416.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-Specialized Parametrization. In *Proc. EGRW '02*, 87–98.

VAXMAN, A., CAMPEN, M., DIAMANTI, O., PANOZZO, D., BOMMES, D., HILDEBRANDT, K., AND BEN-CHEN, M. 2016.

Directional field synthesis, design, and processing. *Computer Graphics Forum 35*, 2.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proc.*, SIGGRAPH '94, 247–256.

WESSELING, P. 2004. *An Introduction to Multigrid Methods*. R.T. Edwards.

A Coarse-to-Fine Operator Details

Below we give some implementation details about and insights into the un-flip and un-collapse operator briefly mentioned in Section 6.3. Refer to Figures 6 and 7 for illustrations of the symbols used in the following.

Un-Flip Operator

Given a flip operator ω_i, the corresponding un-flip operator $\bar{\omega}_i$ computes $\mathbf{f}^{i-1}$ and $\mathbf{t}^{i-1}$ as

$$\begin{aligned}
\mathbf{f}^{i-1}(h_{\text{flip}}) &\coloneqq \mathbf{f}^i(h_{\text{b}}) & \mathbf{f}^{i-1}(h^*_{\text{flip}}) &\coloneqq \mathbf{f}^i(h_{\text{a}}) \\
\mathbf{f}^{i-1}(h_{\text{a}}) &\coloneqq \mathbf{t}^i_{\text{flip}}(\mathbf{f}^i(h_{\text{a}})) & \mathbf{f}^{i-1}(h_{\text{b}}) &\coloneqq (\mathbf{t}^i_{\text{flip}})^{-1}(\mathbf{f}^i(h_{\text{b}})) \\
\mathbf{t}^{i-1}_{\text{flip}} &\coloneqq (\mathbf{t}^i_{\text{flip}})^{-1} & \mathbf{t}^{i-1}_{\text{a}} &\coloneqq \mathbf{t}^i_{\text{a}} \circ (\mathbf{t}^i_{\text{flip}})^{-1} \\
\mathbf{t}^{i-1}_{\text{b}} &\coloneqq \mathbf{t}^i_{\text{b}} \circ \mathbf{t}^i_{\text{flip}}
\end{aligned}$$

and as well as

$$\mathbf{f}^{i-1}(h) \coloneqq \mathbf{f}^i(h) \qquad \text{and} \qquad \mathbf{t}^{i-1}_h \coloneqq \mathbf{t}^i_h$$

for all remaining half edges h not specified above. Figure 6 illustrates the different half edges h and transition transition functions $\mathbf{t}$ referred to above.

Observe how $\text{dom}(\mathbf{f}^{i-1}) = \text{dom}(\mathbf{f}^i)$ for the un-flip operator.

Un-Collapse Operator

Given a half edge collapse operator ω_i, the corresponding un-collapse operator $\bar{\omega}_i$ computes the transition functions $\mathbf{t}^{i-1}$ as

$$\begin{aligned}
\mathbf{t}^{i-1}_0 &\coloneqq \mathbf{t}^i_{2\ldots k-2} \\
\mathbf{t}^{i-1}_1 &\coloneqq \mathbf{t}^{i-1}_{k-1} \coloneqq \text{Id} \\
\mathbf{t}^{i-1}_j &\coloneqq \mathbf{t}^i_j \quad \text{for all } 2 \leq j < k-1
\end{aligned}$$

with the accumulated transition functions

$$\mathbf{t}^i_{j\ldots l} \coloneqq \begin{cases} \mathbf{t}^i_{l^*} \circ \mathbf{t}^i_{(l-1)^*} \circ \ldots \circ \mathbf{t}^i_{j^*} & j \leq l \\ \text{Id} & \text{otherwise} \end{cases}$$

and the parametrization $\mathbf{f}^{i-1}$ is computed as

$$\begin{aligned}
\mathbf{f}^{i-1}(h^*_0) &\coloneqq \mathbf{t}^{i-1}_0(\mathbf{f}^i(a)) \\
\mathbf{f}^{i-1}(h^*_1) &\coloneqq \mathbf{f}^i(b) \\
\mathbf{f}^{i-1}(h^*_{k-1}) &\coloneqq \mathbf{f}^i(c) \\
\mathbf{f}^{i-1}(h_{k-1}) &\coloneqq \alpha(\mathbf{t}^{i-1}_{0\ldots i_\alpha-1})^{-1}(\mathbf{f}^{i-1}(h^*_{i_\alpha})) + \\
&\quad\ \beta(\mathbf{t}^{i-1}_{0\ldots i_\beta-1})^{-1}(\mathbf{f}^{i-1}(h^*_{i_\beta})) + \\
&\quad\ \gamma(\mathbf{t}^{i-1}_{0\ldots i_\gamma-1})^{-1}(\mathbf{f}^{i-1}(h^*_{i_\gamma})) \\
\mathbf{f}^{i-1}(h_j) &\coloneqq \mathbf{t}^{i-1}_{0\ldots j}(\mathbf{f}^{i-1}(h_{k-1})) \quad \text{for all } 0 \leq j < k-1.
\end{aligned}$$

Figure 7 provides an overview of the half edges h, a, b, c and transition functions $\mathbf{t}$ referred to above. The factors α, β and γ are the barycentric coordinates used to interpolate the UV coordinates of the vertex removed during the half edge collapse (c.f. Figure 4 (e)).

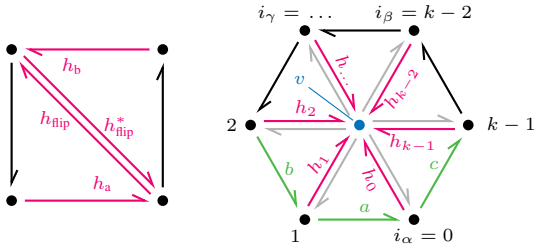


Figure 14: *An illustration of all the pointers and quantities stored in an example of a flip record (left) and a collapse records (right) in the decimation log. Note that both illustrations represent the state* prior *to the decimation operation. Figures 6 and 7 provide a before and after view.*

Note how the inverse collapse operator extends the domain of the parametrization:

$$\text{dom}(\mathbf{f}^{i-1}) = \text{dom}(\mathbf{f}^i) \cup \{h_0, h_1, h_{k-1}, h^*_0, h^*_1, h^*_{k-1}\}.$$

B Decimation Log Storage Format

The following piece of EBNF notation describes the decimation log. Refer to Figure 14 for an illustration of all the half edges and quantities referred to.

$$\begin{aligned}
\langle \text{decimation log} \rangle &= \{\ \langle \text{record} \rangle\ \} \\
\langle \text{record} \rangle &= \langle \text{flip record} \rangle \mid \langle \text{collapse record} \rangle \\
\langle \text{flip record} \rangle &= 0, h_{\text{flip}}, h_{\text{a}}, h_{\text{b}} \\
\langle \text{collapse record} \rangle &= k, a, b, c, i_\alpha, i_\beta, i_\gamma, \alpha, \beta, \\
&\quad\ h_0, h_1, \ldots, h_{k-1}
\end{aligned}$$

The flip record starts with 0 to distinguish it from a collapse record that always starts with a positive integer. h_{flip} represents a pointer to one of the half edges of the flipped edge. h_{a} and h_{b} point to the preceding half edges of h_{flip} and h^*_{flip} *prior* to the flip operation, respectively (cf. Figure 14).

The collapse record starts with $k = |v|$, the valence of the vertex that is collapsed along h^*_0. Consequently, $h_0, \ldots, h_{k-1}$ represent the half edges pointing to v in clockwise order. a, b and c represent the preceding half edges of h_0, h_1 and h_{k-1}, respectively. Three of the half edges pointing away from v identified by their local 1-ring indices $i_\alpha, i_\beta, i_\gamma \in \{0, \ldots, k-1\}$ (i.e. $h^*_{i_\alpha}, h^*_{i_\beta}, h^*_{i_\gamma}$) serve a special role: They are used to interpolate v in the parameter domain using the weights α, β and $\gamma = 1-\alpha-\beta$ when the un-collapse operation is performed as described in Section 6.3. The interpolation vertices and weights are determined by intersecting the removed vertex with the new triangulation of its 1-ring in its local paramter domain and computing its barycentric coordinates within the intersecting triangle (cf. Figure 4 (e)).

Assuming that we store all global half edge indices as 32 bit integers, the local 1-ring indices i_α, i_β, and i_γ, as well as k as 8 bit integers, and the barycentric coordinates α and β as 64 bit double precision floating point numbers, this format incurs 13 bytes of storage cost for a flip record and $32 + 4k$ bytes of storage cost for a collapse record (i.e. 56 bytes on average). This very compact representation is crucial when implementing the decimation log so that performance is not limited by the memory bandwidth.