Zometool® Rationalization of Freeform Surfaces

Henrik Zimmer, Leif Kobbelt

Abstract—An ever broader availability of freeform designs together with an increasing demand for product customization has lead to a rising interest in efficient physical realization of such designs, the trend toward *personal fabrication*. Not only *large-scale* architectural applications are (becoming increasingly) popular but also different *consumer-level* rapid-prototyping applications, including toy and 3D puzzle creation. In this work we present a method for do-it-yourself reproduction of freeform designs without the typical limitation of state-of-the-art approaches requiring manufacturing custom parts using semi-professional laser cutters or 3d printers. Our idea is based on a popular mathematical modeling system (Zometool) commonly used for modeling higher dimensional polyhedra and symmetric structures such as molecules and crystal lattices. The proposed method extends the scope of Zometool modeling to freeform, disk-topology surfaces. While being an efficient construction system on the one hand (consisting only of a single node type and 9 different edge types), this inherent discreteness of the Zometool system, on the other hand gives rise to a hard approximation problem. We base our method on a marching front approach, where elements are not added in a greedy sense, but rather whole regions on the front are filled optimally, using a set of problem specific heuristics to keep complexity under control.

Index Terms—Rationalization, discrete optimization, Zometool, freeform surface approximation, advancing front, meshing

1 Introduction

Efficient fabricability and realization of modern freeform designs is a hot research topic with applications ranging from making foldable or 3D-printable toys to real-life realizations of architectural designs. There is an abundance of recent work in Geometry Processing dealing with different such rationalizations of geometric objects. One can classify these approaches by their respective “scales”, where the scale is often not only a measure of the actual real-life size of the objects involved, but also correlates well to the *public accessibility* of the results. One can generally differentiate between three classes of methods for realizing geometric designs:

- *large-scale rationalization* dealing with architectural designs and industry produced panels
- *intermediate-scale rationalization* requiring 3d printing or laser cutting of custom panels
- *small-scale rationalization* relying only on commodity ingredients such as paper

A major goal in large-scale processes is the minimization of costs related to the final realization of a design. Besides aesthetic criteria (e.g., smoothness and regularity), physical and constructional aspects are receiving increasing attention in Architectural Geometry, with a typical focus on either the optimization of panel properties, such as planarity (e.g. [1], [2], [3], [4]), or on the minimization of panel diversity (e.g. [5], [6], [7], [8]).

On the intermediate scale various novel algorithms exist for computing 3d puzzles (e.g., [9], [10], [11]) or other realizations of 3d models (e.g., [12], [13], [14], [15]) (even for dealing with moving mechanical parts [16], [17], [18]). Unfortunately, the public availability of the results is often limited by the underlying need for 3d printing or laser cutting of custom parts. Likewise, a widespread use of the physically based computation of custom-shaped balloons [19] is hindered by the general unavailability of liquid rubber.

• *The authors are with the Chair for Computer Sciences 8 – Computer Graphics & Multimedia, RWTH Aachen University, Aachen, Germany*
E-mail: {zimmer, kobbelt}@cs.rwth-aachen.de

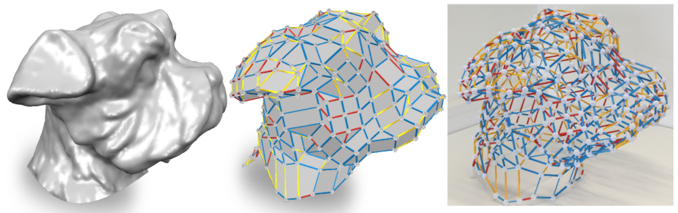


Fig. 1. Rationalized Dog. Our method computes a Zometool rationalization (middle) of a digital input model (left) which can then be manually assembled to yield a *real-life* rationalization (actual photo on the right).

Small-scale (toy-)paper crafting approaches (e.g. [20], [21], [22]) have the advantage of high accessibility of the used material (paper) but generally suffer from structural limitations as well as difficulties to accurately cut/glue proper seams. Furthermore, the class of realizable geometries is restricted to developable surfaces for folding-based approaches such as [23]. Another generally accessible technique is *beady* [24] which presents an interactive system to help design 3D beadwork from polygon meshes. The optimization of knitting patterns [25] is another interesting approach.

Besides paper-based methods the Zometool system (cf. Section 2.1) probably has the largest user base, being used in schools as a means to teach and visualize simple geometries or to model molecules and various higher dimensional polyhedra in different branches of science [26]. Furthermore, Zometool has the advantage of being a construction system consisting of tangible parts which can be fit together perfectly (requiring less manual skills than paper folding), while at the same time relying only on a small number of pre-defined, fixed-size elements (fabricational efficiency) and being over-the-counter purchasable (high public accessibility). However, until now, Zometool modeling of freeform surfaces has been largely overlooked. This paper presents a first method for realizing freeform surface patches as polygonal Zometool models with convex, planar facets. The rationalization is demonstrated in Figure 1.

Zometool is a powerful system which, unrestricted, allows for a huge variety of different panel types. For practical and fabricational reasons we limit our attention to a small subset consisting of triangles and convex (planar) quads. Due to the fact that there are roughly four times as many such quads than triangles, our surface approximation method naturally yields *planarized, quad-dominant* output meshes. With the set of panels fixed, the rationalization problem translates into finding the *highest quality* (best approximating) tessellation using only panels from this set. In this setting there is no continuous optimization of panels or relaxation of the input geometry, the problem is entirely discrete and thus inherently hard to solve. There is no analytical solution, no epsilons or tolerances, either two panels fit together or they don't. This puts substantial restrictions on meshing algorithms for this problem, since the common "vent" of adding new panels to let a solution relax locally (e.g. Steiner points in Delaunay meshing) is not admissible.

Due to the different target applications direct comparisons between methods on the different scales is often not meaningful. Still, typical rationalization-specific geometric problems tend to re-occur regardless of problem scale, often enabling comparisons on a technical level and even utilizing results between levels. For example, ignoring structural stability issues, the Zometool rationalization presented here could be directly transferred to an architectural level by manufacturing the corresponding elements at a different scale, although this is not a central claim of this paper.

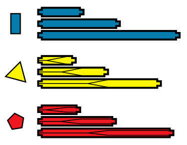
2 Zometool Surface Approximation

Before introducing the approximation problem considered in this paper along with our proposed approach and related work, this section reviews the basic properties of the Zometool system.

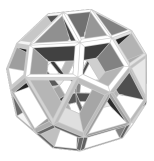
2.1 The Zometool System

The history of the Zometool system dates back to the 60's where it started out as a simple construction system inspired by Buckminster Fulleresque geodesic domes. In the early 70s the system was introduced as a toy, which over the years evolved into the current Zometool state (cf. Section 2.1), and has since been used for visualization and modeling in various branches of natural sciences, from mathematics to chemistry and research at space agencies [26]. Due to the symmetries inherent in the system it is particularly popular for modeling various kinds of symmetric and repetitive shapes, such as molecules or higher dimensional mathematical constructs.

2.1.1 *Struts and Nodes*



The standard Zometool system consists of three different types of colored edges or *struts* (*r*ed, *y*ellow, *b*lue) and a single node type. Each strut is available in three different lengths (0 = short, 1 = medium, 2 = long), e.g., b_0, b_1 and b_2 refer to the three different lengths of blue struts. The strut lengths of a certain type are golden-ratio scalings of each other, i.e., $b_1 = b_0 \cdot \gamma$ and $b_2 = b_0 \cdot \gamma^2$, where $\gamma = 0.5 \cdot (1 + \sqrt{5})$ is the golden-ratio. Furthermore, simple scalings relate the three strut types to each other as follows: $y_0 = 0.5 \cdot \sqrt{3} \cdot b_0$ and $r_0 = 0.5 \cdot \sqrt{2 + \gamma} \cdot b_0$. Note that the struts (lengths) described here are measured from/to the node midpoints, the real-life struts don't extend all the way to the node midpoints.



The Zometool node is a slightly modified rhombicosidodecahedron with 62 slots: 12 pentagonal slots for the red struts, 20 triangular slots for the yellow struts and 30 golden-ratio rectangular slots for the blue struts. Let $D = \{0, \ldots, 61\}$ denote the set of the 62 outgoing directions corresponding to the node slots.

Our surface approximation algorithm mainly relies on three important properties of the Zometool system (for more mathematical details on the struts we refer to [27]):

- *Node symmetry*: Due to the symmetry of the rhombicosidodecahedron there is for each slot an opposite slot of the same type, and incidentally, as $\gamma^2 = 1 + \gamma$, the longest struts (r_2, y_2, b_2) can be built by combining the two shorter ones of the same type, e.g., $b_2 = b_0 + b_1$.
- *(Fixed) Node orientation*: Each strut only causes a translation of a node in space, i.e., the node's orientation (the orientation of the slots) remains constant.
- *Node Coordinates*: The possible coordinates of Zometool nodes in space are of the form $0.5 \cdot (a_0\gamma + a_1, a_2\gamma + a_3, a_4\gamma + a_5)$ with $a_i \in \mathbb{Z}$ and can theoretically take on any position in three-space.

Furthermore, we briefly introduce the Zometool planes and panels used later on.

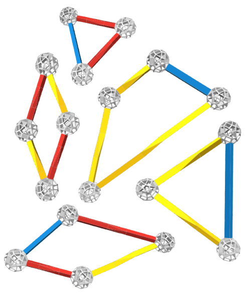
2.1.2 *Planes*

The directions of any two non-parallel slot directions in a Zome node describe a plane Γ with normal vector n_Γ. Let $D|_\Gamma$ denote the subset of all slot directions in D lying in the plane Γ, by the *node symmetry* above $D|_\Gamma$ always has at least four directions. By forming all combinations of pairs of slots and collecting planes with identical sets $D|_\Gamma$ (or equivalently with identical normal vectors, disregarding the sign) we extracted a total of

121 different planes. These planes can be divided into 6 different types, where the planes of the same type are defined by having sets $D|_\Gamma$ containing the same number and types (red, blue, yellow) of slots and only differ in orientation (cf. Figure 2). The first three types have as normal vector a blue, red or yellow strut of the system respectively, the last three types are not orthogonal to any direction in the system. The different plane types allow for a more or less rich variety of (planar) faces. Type 1 can be considered the richest, as it contains the most struts. For tessellation purposes, planes of types 4 and 6 are the most restrictive in the sense that they only allow for quad panels.

2.1.3 Panels

While the Zometool system does not explicitly provide a set of polygonal faces, such a set can be defined in a natural way: each k-tuple of struts that can be connected to a simple closed loop can be thought of as a k-gon face. Let Z denote the set of all such polygonal panels. Besides for the case $k = 3$, the so defined faces may be neither planar nor convex. However, if the face-defining tuple of struts lies completely in one of the 121 planes defined above it is planar. While planarity is not essential for general surface approximation, it can be critical for architectural scenarios. For practical reasons, the rationalization algorithm presented in this work deals exclusively with triangle and quad panels. In detail, the used set of panels $P \subset Z$ consists of the 29 unique triangles of the Zometool system together with a set of 118 different convex, planar quads (extracted as described in Section 3.4). The inset figure visualizes five different panels from this set. For higher k the number of such faces steeply rises, making general k-gon-based Zometool tessellations unsuitable for efficient, low panel count rationalization.



2.2 Problem Statement

Our goal is to convert a freeform design $\mathcal{S}$ to a Zometool representation $\mathcal{Z}$, called *Zome mesh*. Naturally, the restricted set of directions of the system implies that not all curvature configurations can be reconstructed and, hence, the "best fitting" Zome mesh $\mathcal{Z}$ approximating $\mathcal{S}$ is sought. The complexity and inherent discreteness naturally exclude the possibility of a direct solution or even relaxation of the problem. Since a globally optimal solution is infeasible we are restricted to making (the best of) local decisions. A common class of meshing algorithms based on this metaphor is *advancing front* techniques, where, starting from a seed, the output mesh is grown over the input surface face-by-face. Advancing front techniques rely on the ability to insert arbitrary panels where needed. In general, two different parts of a front meeting somewhere on a curved surface can always be joined by inserting a custom panel, whereas in the Zometool system there may be no fitting (set of) panels available. Front self-collisions are related to the topological structure of a surface [28] and are inevitable on closed surfaces and surfaces with genus $\neq 0$. Unfortunately, front collisions can occur also on open flat surfaces if the growing is not handled properly. Furthermore, to be in accordance with the Zometool rule: *If it works, it works perfectly*, a solution to the approximation problem should not rely on manipulation or bending of nodes or struts.

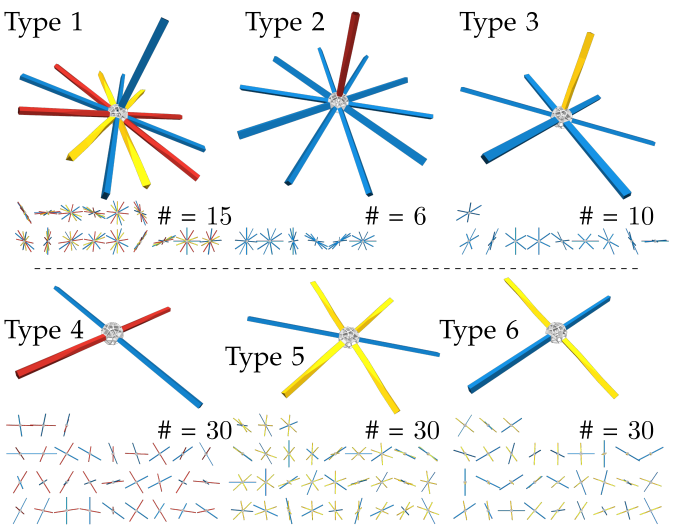


Fig. 2. The 121 planes of Zometool system can be divided into 6 types. A representative plane of each type is visualized as a node together with all struts lying in that particular plane. For the first three types additionally the orthogonal strut is shown. The small images below show the different variations (orientations) of planes of the respective type and # states the total number of variations within each type.

2.3 Our Approach

In this work we consider the class of freeform patches, i.e., disk-topology surfaces, and propose a novel advancing front technique to grow $\mathcal{Z}$ over such a surface $\mathcal{S}$, based on a pre-computed *harmonic* growing field designed to avoid front self-collisions. We generally do not grow on a face-by-face basis but rather compute locally optimal solutions to fill gaps on the front in a "best-fitting" manner. This process is made tractable by a controlled growing based on a pre-computed set of possible panels. While the method performs well on general freeform patches, on objects having reflectional symmetry asymmetric results can be distracting. For this we further present a simple, but effective symmetry plane constraint to enable reflective symmetric results.

Section 3 introduces the foundations of our advancing front approach. The symmetry constraint used for meshing reflective symmetric objects is the topic of Section 4.

Before presenting results in Section 6, Section 5 presents experiments, motivating various choices and heuristics used in our method. Limitations of the method are discussed in Section 7 and Section 8 concludes the paper.

2.4 Related Work

2.4.1 Zometool

For designing Zometool structures two software tools exist: vZome [29] and ZomeCAD [30]. Both implement powerful symmetry commands and also include different ready-defined polytopes to support the manual design interface where nodes and struts can be added one-by-one. Except for a recently submitted approach [31] dealing with Zometool approximation of closed surfaces, without guaranteeing the potentially important qualities of planar, convex panels, to the best of our knowledge no other attempts have been made at Zometool based freeform surface approximation.

2.4.2 Paneling Techniques

Rationalization approaches dealing with the reduction of panel diversity are in spirit similar to our approach as the goal is to cover an input design with a restricted set of panels. In [6], [7] an iterative process based on clustering and averaging of (quadrilateral resp. triangular) faces is used to reduce the set of different panels and to modify the input surface to accommodate to the change. Eigensatz *et al.* add another component to the optimization by explicitly also considering re-usage and optimization of different classes of panel molds [5]. The rationalization method of Zimmer *et al.* [8] leaves the input geometry unaltered while erecting a set of optimized pyramidal elements on the faces of the input mesh. These approaches typically consist of two main components: a discrete optimization (usually variants of the classical Set Cover problem) for clustering the different panels or molds and continuous optimizations to compute representative molds and adapt the input geometry to the change.

In a broader sense, remeshing algorithms ([32], [33], [34] provide an overview) also perform surface re-paneling. However, remeshing processes are often guided by continuous measures such as smoothness, inner-angles or alignment of the panels of the resulting mesh and seldomly consider discrete criteria such as panel diversity. To our knowledge there are no published remeshing techniques restricted to a predefined, fixed set of panels.

2.4.3 Advancing Front Paneling Techniques

A vast number of advancing front remeshing strategies exist, with applications ranging from Delaunay mesh generation [35], point cloud interpolation [36], height-field triangulation [37] to extracting iso-surfaces from implicit functions [38] to name a few, but all without the fixed-panel restrictions posed in our setting.

3 Zometool Front Growing

First, a few short notes on notation: The input freeform surface patch is denoted $\mathcal{S}$ and the approximating Zome mesh $\mathcal{Z}$. The vertices $v \in \mathcal{Z}$ are referred to as nodes. Depending on the context either s or a pair (d, l) of direction $d \in D$ and strut length $l \in \{0, 1, 2\}$ will be used to refer to a strut. Finally, $\pi : \mathbb{R}^3 \mapsto \mathcal{S}$ denotes a projection operator for mapping points (generally nodes of $\mathcal{Z}$) to their respective closest point on $\mathcal{S}$. For practical reasons $\mathcal{S}$ is represented as a (high resolution) triangle mesh.

The structure of our approximation method is illustrated in the block diagram in Figure 3. After placing an initial panel, $\mathcal{Z}$ is grown by incrementally adding panels $p \in P$ to the struts (edges) on the boundary $\partial\mathcal{Z}$ of the current mesh. To avoid front self-collisions, the order of growing is controlled by a harmonic field of arrival times (cf. Section 3.1), while the actual grow-operation performed depends on the local shape of the front (cf. Section 3.2). Panel approximation energies (cf. Section 3.3) are used to evaluate the quality of different grow-options. To enable efficient growing, the set of all available panels P is pre-computed and stored in a look-up table `Panels`(s) for quick access to the subset of panels compatible with the current strut s (cf. Section 3.4).

3.1 Harmonic Front-Growing Strategy

Assuming a first polygon $p \in P$ has been placed on $\mathcal{S}$, i.e., $\mathcal{Z} = p$, the goal is now to make sure that the front of polygons of $\mathcal{Z}$ grown from this position does not self-intersect. For this a field G of arrival times on $\mathcal{S}$ is pre-computed that is free of critical points. Field growing starts around $\pi(p)$ (the projection of the polygon onto $\mathcal{S}$) and ends at the boundary $\partial\mathcal{S}$. The intuitive wish of advancing the front at a constant rate from the starting point, i.e., according to a geodesic field, can lead to self-intersections on the front, as this field is generally not free of critical points. Figure 4 visualizes the difference between such a geodesic field and an *harmonic* field, which is indeed free of such critical points. Hence, $G : \mathcal{S} \mapsto \mathbb{R}$ is computed to be a harmonic field on $\mathcal{S}$, where the projection of p has arrival time 0 and the boundary of $\mathcal{S}$ has arrival time 1:

$$\Delta_{\mathcal{S}} G = 0 \quad \text{s.t.} \quad G(\pi(p)) = 0 \quad \text{and} \quad G(\partial\mathcal{S}) = 1,$$

where $\Delta_{\mathcal{S}}$ is the graph-Laplacian. Our front-growing strategy is now to advance the front $(\partial\mathcal{Z})$ according to G using a priority-queue of the nodes lying on the front. I.e., the part of the front around the node with the earliest arrival time shall be popped from the queue and expanded next. Note that the graph-Laplacian is guaranteed to always yield a field free of degeneracies, which can however be biased by irregular tessellations. The opposite holds when using geometrically motivated weights, e.g., the cotangent-Laplacian. In our setting, $\mathcal{S}$ is assumed to be a high-resolution, uniformly remeshed

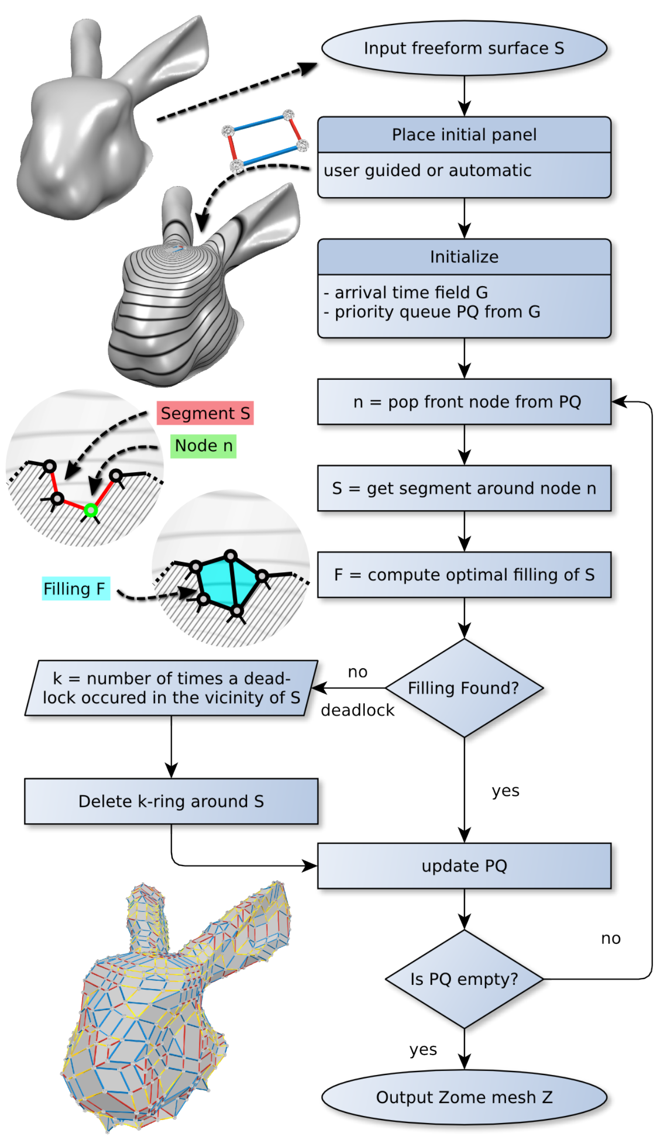


Fig. 3. Block diagram of the proposed Zome mesh growing algorithm.

triangle mesh, where both are acceptable choices and the graph-Laplacian can safely be chosen for simplicity.

3.1.1 Initial Panel

To accommodate for different usage scenarios we have implemented two different growing modes:

- Free Mode: free growing starting with the best fitting panel at the *harmonic center* of $\mathcal{S}$.
- Symmetry Mode: the growing starts close to the *harmonic center* of $\mathcal{S}$ with one edge constrained to lie on a user-defined symmetry plane with growing restricted to one side of the plane (cf. Section 4).

We define the *harmonic center* of $\mathcal{S}$ as the middle vertex of the harmonic disc parametrization of $\mathcal{S}$ with circular boundary conditions. Using this position as a starting point tends to produce harmonic fields G with more concentric iso-contours. However, in coarse panelings

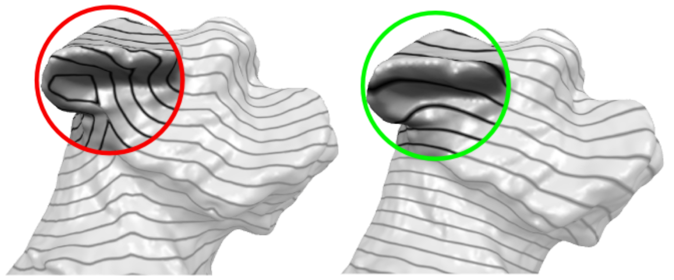


Fig. 4. Growing along a geodesic field (left) can lead to self-intersections of the front, whereas an harmonic field (right) is free of critical points.

important features on the input surface can be missed if they are not sampled (hit by a node). For this there is also the option of a user-selected starting position, e.g., to not miss HOMER's nose it was manually specified as the starting point in Figure 15.

3.2 Segment Filling Strategy

While the arrival time dictates *where* to grow next, the local *shape* of the front at that position dictates *how* it is grown. To avoid complicated intersection handling in each step we differentiate between *convex* and *concave* segments on the front (cf. Figure 5). Intuitively, in convex segments one can simply grow the front by adding panels without risking self-intersections, whereas in concave segments this is not the case. However, panels cannot be added independently from each other (especially not in concave segments) as a new panel might easily generate a concavity not fillable by any other (combination of) panels in P. We thus adapt a [39]-inspired filling strategy to *optimally* fill entire connected segments on the front in one step. The same strategy is applied to fill concave and convex segments, but, since the concave areas are the more problematic ones, our strategy is to handle them first. The strategy is explained in more detail below and Section 3.2.1 describes the optimal fill computation.

We measure convexity on the input surface by first projecting the front $\partial\mathcal{Z}$ onto $\mathcal{S}$. For a node $v \in \partial\mathcal{Z}$ let $A(v) \in [0, 360°]$ be the *front angle* or *convexity* of v, defined as the angle between the two edges spanned by v and its boundary neighbors w and u when projected onto

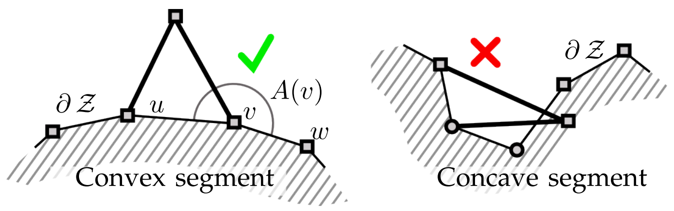


Fig. 5. Care needs to be taken when advancing the front in concave segments (right). Squares denote convex nodes and circles concave ones. The nodes u and v are *purely* convex. A concave segment is a connected set of struts with concave inner nodes.

the tangent plane at $\pi(v)$. We call v *convex* if $A(v) > 180°$. A *segment* on the boundary is defined as a connected set of struts $S := \{s_0, \cdots, s_n\}$ (the inner nodes between these struts shall be covered by the filling). In a *concave segment* all inner nodes are concave and a *convex segment* is defined to consist only of the two outgoing struts of a (purely) convex center node. A node is considered purely convex if also both its neighbors are convex.

Now depending on the convexity of a node v (popped from the priority-queue) and its surrounding segment, the front is grown differently around v:

1) If v is purely convex, then an optimal fill is computed for its convex segment $S := \{s_0, s_1\}$.
2) If v is convex but has concave neighbors, then the concave segment of the most concave (potentially most problematic) neighbor is filled first (see next step).
3) If v is concave, then the concave segment $S := \{s_0, \cdots s_n\}$ around v is optimally filled.

3.2.1 Computing the Optimal Segment Filling

First, a *valid filling* of a segment S is defined as a strip of panels $\mathcal{F} \subset P$ which cover the inner nodes of the segment without intersecting other parts of $\mathcal{Z}$ or changing the topology. A node is considered as covered if it has a complete 1-ring (i.e., no longer lies on the boundary). Figure 6 shows a valid filling (left) and an invalid filling (right) which neither covers the vertices nor leaves the topology intact. Assuming the existence

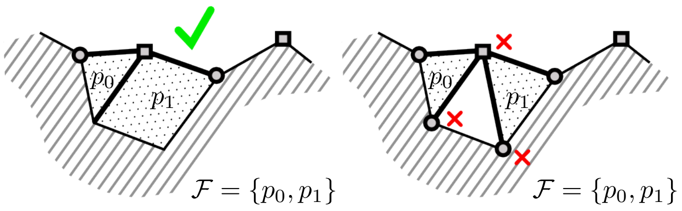


Fig. 6. Two fillings of the concave segment from Figure 5. Note that after the filling, some previously convex front nodes can turn concave.

of an energy functional $\texttt{Cost}(p, \mathcal{S}) \geq 0$ (to be defined later) measuring the approximation quality of the panel p w.r.t. the freeform surface $\mathcal{S}$, an *optimal filling* of a segment is defined as the valid filling $\mathcal{F}$ minimizing $\texttt{Cost}(\mathcal{F}) = \sum_{p \in \mathcal{F}} \texttt{Cost}(p, \mathcal{S})$.

An optimal filling can be computed in a recursive fashion by exhaustively trying out "all fillings" of the gap, but for that to be practicable the number of candidate fillings has to be bounded. An unfortunate difference to [39] is that the Zometool setting does not permit for an efficient dynamic programming-based solution, as, due to the limited diversity of panels, it is not possible to efficiently enumerate all possible solutions a priori. The approach described here therefore limits the full-search and makes it practicable by (1) only considering a thin 1-ring filling strip, i.e., all new nodes must only be one strut away from the segment, (2) restricting the number of new panels (the recursion depth $maxdepth$) and (3) using pruning to early discard invalid and energetically poor solutions. These three components are detailed in the following.

To guarantee a local, thin filling-strip (1), the gap is filled in a structured, recursive manner from left to right. In each step (recursion level) a panel from $\texttt{Panels}(s)$ is added to the current so-called *active strut* s. Every added panel updates the segment S: struts in S which are part of the new panel are removed from S and when S is empty and the filling is valid, it is completed. Figure 7 demonstrates the filling process and how the segment and the active strut are updated. In detail, the filling starts from the left-most strut $s_0 \in S$, the initial *active strut*, it points to the first inner node to be covered. After a panel p has been added and the corresponding struts have been removed from S, unless the filling is completed or $maxdepth$ has been reached, the new strut that points to the next un-covered node is added to S and gets activated. Note that the next

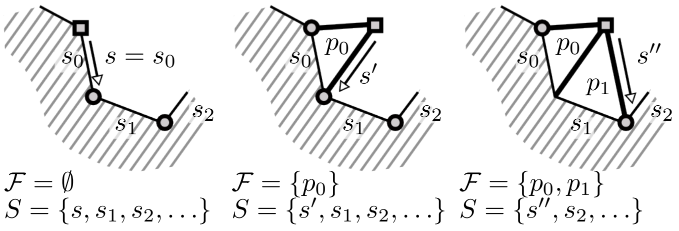


Fig. 7. A "gap" on the front to be filled is defined by a set of directed struts S, called segment. The optimal filling $\mathcal{F}$ is computed in a structured manner from left to right, in each iteration a panel is appended to the current *active strut* s (denoted by an arrow). The three subfigures demonstrate the filling process. Each added panel modifies the segment S (and the active strut), when S is empty the gap is filled.

un-covered node is the same as the previous one if the added panel did not complete the corresponding 1-ring. Pseudo-code for the `OptFill` function for computing an optimal filling of a gap is detailed in Algorithm 1. A useful heuristic in practice is the sorting of the panels in the look-up table $\texttt{Panels}(s)$ in descending order of approximation error (or cost). Based on the observation that panels with high costs (having a bad orientation and/or distance w.r.t. $\mathcal{S}$) are less likely to be part of an optimal filling of a smooth surface $\mathcal{S}$ than panels with low costs, this enables reaching the optimal filling faster in general. The function `UpdateSegment` referred to in the code is responsible for updating the segment S as mentioned above, i.e., removing struts covered by a panel and possibly adding the new active strut. The function `NoLocalIntersect` tests for intersections between the filling and the neighboring part of $\mathcal{Z}$.

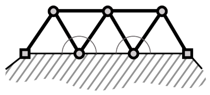
Continuing the recursive growing process only along the active strut localizes the filling, as it always remains directly connected to the segment to be filled.

Algorithm 1: Procedure to recursively compute the optimal filling $bestFill$ with cost $bestCost$ of a segment S.

```
Input: Segment to be filled $S$
Output: Optimal filling $bestFill$, cost $bestCost$
Initialization: $bestFill \leftarrow \emptyset$, $bestCost \leftarrow \infty$,
                $minGS \leftarrow \min_{v \in S}(G(\pi(v)))$

Function OptFill($S$, $\mathcal{F}$, $depth$);
Input: Current segment $S$, filling $\mathcal{F}$ and $depth$
if $S = \emptyset$ then                          /* filling done */
    if IsValidFill($\mathcal{F}$) then
        $bestFill \leftarrow \mathcal{F}$;
        $bestCost \leftarrow$Cost($\mathcal{F}$);
    end
else                                             /* continue filling */
    if $depth < maxdepth$ then
        $s \leftarrow$ get next active strut from $S$;
        $E \leftarrow$ sort Panels($s$) by descending cost;
        forall the panels $p$ of $E$ do
            if $\min_{v \in p}(G(\pi(v))) > minGS$ then
                /* remove struts covered by $p$ */
                $S' \leftarrow$UpdateSegment($S$,$p$) ;
                /* add panel to filling */
                $\mathcal{F}' \leftarrow \mathcal{F} \cup p$ ;
                if Cost($\mathcal{F}'$) $< bestCost$ then
                    if NoLocalIntersect($\mathcal{F}'$, $\mathcal{Z}$) then
                        /* recurse */
                        OptFill($S'$,$\mathcal{F}'$,$depth + 1$)
                    end
                end
            end
        end
    end
end
```

Still, to guarantee termination (2) the maximal recursion depth (i.e., number of panels) must somehow be restricted. To do this we first limit the size of segments by restricting the total convexity of their inner nodes to max. 360°, i.e., $\sum_{v \in S} A(v) < 360°$. I.e., the corresponding segment S around a node popped from the front of the priority queue is grown by adding struts in both directions until this bound is reached. As a heuristic to correspondingly limit $maxdepth$, we consider the flattest possible concave segment with convexity sum 360° and set $maxdepth$ equal to the expected number of equilateral triangles needed to fill it, i.e., $maxdepth = 5$. For depths > 5 computations rapidly become less practicable and experiments showed no quality improvement.

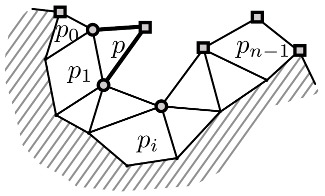


Pruning (3) is enabled by using a monotonically increasing cost function and the above structured growing strategy. Partial fillings $\mathcal{F}$ having a greater energy than the currently best valid filling Cost($\mathcal{F}$) $> bestCost$ can be pruned, as can fillings containing panels which intersect $\mathcal{Z}$. Furthermore, unnecessary energy evaluations and intersection tests can be avoided for "inward" pointing panels by pruning these away based on their nodes' arrival time values. The panels in Panels(s) for a strut s point in all different directions around the strut, i.e., not only in the current growing direction but also back over the already covered part of $\mathcal{S}$. We utilize the growing field G to quickly discard panels having nodes v with smaller arrival time than the current minimum of the segment S, i.e., $G(\pi(v)) \leq \min_{u \in S} G(\pi(u))$.

The intersection handling in explained next.

3.2.2 Handling Filling Intersections

The harmonic growing strategy was devised to avoid two distant parts of the front colliding with each other and the intersecting handling is correspondingly restricted to the *local* configurations which occur during the filling of a segment. When filling a segment, a newly inserted panel p must not intersect the rest of $\mathcal{Z}$. Luckily, p needs not be tested against the whole of $\mathcal{Z}$ but intersection tests can be restricted to a local neighborhood of panels $\{p_i\}$ around the segment to be filled. The thickness of the neighborhood can be derived from the maximal strut length in the Zometool system, as the new panel p can maximally extend so far away from its active strut. The inset figure shows an example of such a local neighborhood of panels (not necessarily a strip in general) for the concave segment in Figure 5.



Locally around the nodes of p the discrete set of 62 directions of the Zometool system can be utilized to perform very efficient and numerically stable intersection tests between p and the neighboring panels sharing one (or more) of its nodes. Note that this always includes the panels incident to the nodes on the *active strut* but can also include other nodes of p mapped to existing nodes on the front. Two neighboring panels sharing a node v are considered intersecting (cf. Figure 8) if they are co-planar and their interiors overlap (two 2D problems) or if they are not co-planar and the intersection axis of their supporting planes is contained in both panels (two 2D problems). This works since the panels in P are convex, i.e., only have inner angles $< 180°$. For a pair of panels p and p' sharing a common node v the set of all such node based intersections can be pre-computed and parametrized over 4 direction indices $d_0, d_1, d_2, d_3 \in D$, with d_0, d_1 corresponding to the struts of panel p at v and d_2, d_3 corresponding to the struts of panel p' at v.

Now, general polygon intersection tests only need to be performed between p and the non-neighboring panels in $\{p_i\}$. The efficient tri_tri_intersect test by Möller [40] is used for panels in general position, while for co-planar panels CGAL [41] is used for stability. In these tests quads are divided into two triangles. Note that, due to numerical issues, intersections between struts can be missed the new struts of p need to be tested against all other struts in the strip.

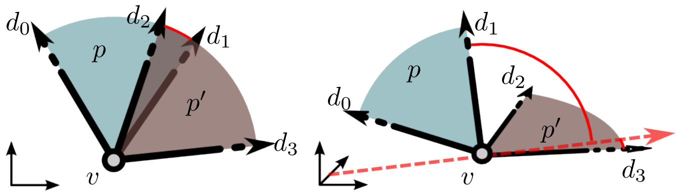


Fig. 8. The two *node*-intersection cases between two panels p and p' incident to a node v. For co-planar panels it is enough to check if a strut of one panel is contained in the angle span of the other. If they are not co-planar, the intersection axis (red, dashed) is checked for containment in both angle spans. Here, the left configuration intersects while the right doesn't.

3.3 Panel Cost Function

The cost of an panel $p \in P$ consists of a *closeness* and an *orientation* energy: $\texttt{Cost}(p, \mathcal{S}) = (1-\alpha)E_{\text{close}}(p, \mathcal{S}) + \alpha E_{\text{orient}}(p, \mathcal{S})$, both are evaluated at a regular set of samples $\{x_i\}$ on the panel p. The closeness energy measures the distance between the samples and their projections onto $\mathcal{S}$:

$$E_{\text{close}}(p, \mathcal{S}) = \frac{1}{N \cdot c^2} \sum_{i=1}^{N} (\pi(x_i) - x_i)^2,$$

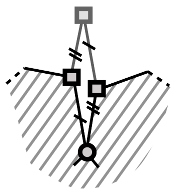
where the normalization factor c is used to relate distances to the fixed lengths of the struts. We choose $c = b_0$, which keeps the energy within the range $[0, 1]$ for distances less than b_0. The orientation energy measures the deviation between the normal n of the panel p and the normals at the projections of the samples:

$$E_{\text{orient}}(e, \mathcal{S}) = \frac{1}{4N} \sum_{i=1}^{N} (n_{\pi(x_i)} - n)^2.$$

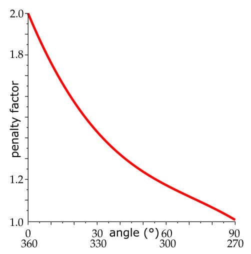
The normalization by 4 keeps the energy in the range $[0, 1]$, since the squared length of two maximally different, opposite normals n and $-n$ is 4. In all our experiments we use $\alpha = \frac{2}{3}$ for the linear combination of energies (cf. Section 5.1).

3.3.1 Penalizing Pointy Fillings

Even though the growing field G prescribes a continuous, interference-free front, the discrete panels do not perfectly adhere to this paradigm. The optimal filling of a segment can include pointy panels unnecessarily intruding into neighboring areas or create thin slits later only fillable by a corresponding pointy parallelogram. The inset illustrates a thin slit where no appropriate slots or struts exist to directly connect the two square nodes. The slit can only be closed by a parallelogram, guaranteed to exist by the *node symmetry* property.



We consider angles below 90° and above 270° increasingly problematic. To keep the front of a filling $\mathcal{F}$ compact, we penalize the cost depending on the minimal (A_{min}) and maximal (A_{max}) front angle values according to a penalty function f: $\texttt{Cost}(\mathcal{F}) \leftarrow \texttt{Cost}(\mathcal{F}) \cdot f(A_{\text{min}}) \cdot f(A_{\text{max}})$. We use the empirically determined penalty function illustrated in the inset. If all angles on the front of the new filling are greater than 90° and less than 270° the factors are 1. If, e.g., the smallest angle is 55° but the greatest is still less than 270° the cost is multiplied by ca. 1.2 (only) once. Simply forbidding certain front angles would increase the risk of not finding any solution at all in cases where more extremal panels are necessary. This is why we choose to just penalize them in favor of another solution where one is available.



3.4 Panel Look-up Table

A table $\texttt{Panels}(s) \subset P$ of all admissible triangular and (planar) quadrilateral panels is pre-computed for each strut. For a strut s (or equivalently a pair (d, l)) the table returns a list of all panels having an edge parallel to the direction of d and the corresponding strut length type l.

All triangles can be generated by simply enumerating all pairs of directions d_0 and $d_1 \neq d_0$ emanating from a node, in combination with all possible lengths and checking if a connection between the end points exist.

The quadrilateral panels can be generated in a similar fashion. However, here care has to be taken to (1) ensure planarity, and avoid (2) self-intersecting and (3) backwards growing/non-convex panels (not adhering to the harmonic growing paradigm). Quads not fulfilling these requirements are easily discarded by evaluating and comparing the cross-products (normals) at each of the four corners. To allow for efficient intersection computations no adjacent edges are allowed to be parallel (i.e., quad degenerating to triangle). On average the look-up table for a direction $d \in D$ has about 40 triangles and 370 quads. Note that, since P only consists of 29 different triangles and 118 different quads in total, some of these panels must be identical except for different orientations.

3.5 Implementation Details

$\mathcal{Z}$ is represented by a halfedge-based data structure. This has several advantages, e.g., an efficient traversal of $\partial \mathcal{Z}$ (front nodes) and neighboring faces of a node for intersection tests, a simple `is_boundary` check to see if a node has been covered by a filling operation etc.

$\pi(v)$ can be efficiently implemented using a BSP search structure. However, when evaluating fillings a, potentially huge, number of projections is carried out, many on the same or similar samples which can quickly become an unnecessary bottle-neck. By using a spatial-hashing inspired sparse crust of octree cells surrounding $\mathcal{S}$ to cache already computed projections, we were able to reduced projection times by as much as 2 orders of magnitude (depending on the model) compared to using BSP only.

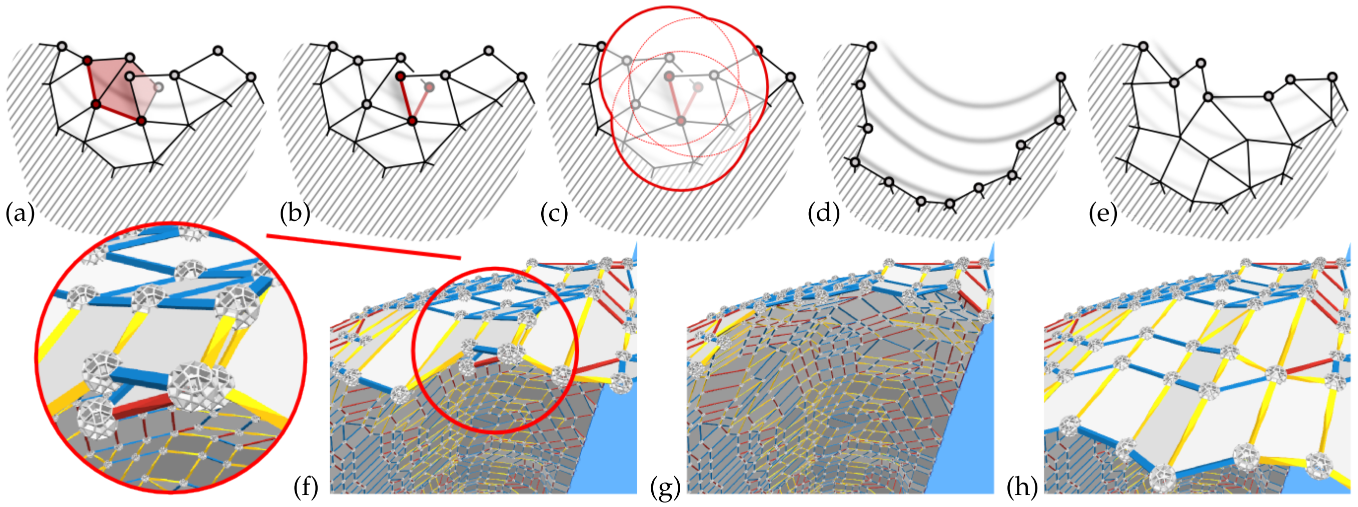


Fig. 9. (a) The filling of the segment (red) has grown underneath a neighboring part of $\mathcal{Z}$. The neighboring segment in (b,f) cannot be filled. This deadlock is resolved by deleting a neighborhood of panels (c,d,g) and letting the standard growing re-fill the gap (e,h).

3.6 Repairing Deadlocks

Finally, in the vicinity of concavities or areas constrained by symmetry (cf. next section) bad local segments can occur on the front, which have very narrow or wide angles. This can lead to fillings growing "underneath" neighboring parts of $\mathcal{Z}$ (without actually intersecting). Depending on the configuration, there might be no valid, intersection-free filling for a neighboring segment, which constitutes a deadlock, cf. Figure 9. We deal with deadlocks by (1) deleting the panels involved in the bad configuration together with a neighborhood of surrounding panels, (2) update the priority queue according to the new boundary and (3) let the standard growing pipeline re-fill the gap. It is not clear how to choose an appropriate radius for deleting the right amount of panels. Hence, we first start by conservatively removing only panels within a small radius (e.g., 1x longest strut) around the nodes of the non-fillable segment, and if a problem continues to arise at a similar location, then a larger neighborhood (2x longest strut) is cleared and so on. While this in theory does not guarantee a solution, changing the local neighborhood structure also changes the order of growing and in all our tests sufficed to resolve deadlocks. As indicated by Table 1 such fixes are rarely needed. To keep track of the number of times problems have arisen, the corresponding positions in space can be tagged using the spatial hashing mentioned above.

In case growing can continue without causing a deadlock, still unwanted "fold-over" configurations result. While these could be dealt with in a similar fashion, a better approach would be a post-processing operator to locally nicify such areas as discussed in Section 7.

4 Respecting Reflectional Symmetry

On objects having reflectional symmetries, as can be found in characters or various man-made designs, the randomness of a freely grown Zome mesh may look disturbing to the eye where a symmetry is expected. To this end we implemented a constraint to restrict the growing to one side of a user-definable symmetry plane $\Sigma = (x, n)$, where x is a point on the plane and n its normal. Afterwards a simple mirroring operation can be used to obtain a $\mathcal{Z}$ covering the whole input surface. However, to avoid holes when mirroring, there must exist a common, simple interface chain on Σ connecting the two sides. This calls for a slightly modified initialization and growing procedure in the vicinity of Σ, while $\mathcal{Z}$ can be grown as usual away from the plane.

4.1 Node Orientation

To yield a symmetric output, it is important that not only Σ is symmetrically placed on the input shape $\mathcal{S}$ but that a symmetric Zome plane (cf. Figure 2) is also properly aligned with Σ. Planes of the first three types are guaranteed symmetric, as each of them is orthogonal to a system direction $d \in D$, for which there always exists the opposite direction (or slot) $-d \in D$. To guarantee a common interface of symmetry plane nodes and struts, we first rotate the (global) node orientation to align one of the planes inherent in the Zometool system with Σ, cf. Figure 10. Let $D|_\Sigma$ denote the set of slots (directions) lying in Σ. Motivated by the symmetry experiment in Section 5, to account for diverse curvature profiles and enable highest quality approximation of the intersection curve $\mathcal{S} \cap \Sigma$ we always use a type 1 plane. Optionally, if the curvature profile of the $\mathcal{S} \cap \Sigma$ is simple and has only one or a few prominent directions (e.g., a straight line), we also rotate the node in the plane (around n) to align its directions $D|_\Sigma$ (shown in green in the figure) as well as possible with these directions.

4.2 Initialization

As above, the initial panel $\mathcal{Z} = \{p\}$ should be placed close to the *harmonic center* but must now also have

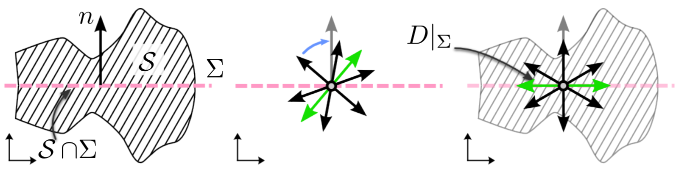


Fig. 10. For an exemplary 2d configuration (middle) the default global node orientation (denoted by a node with outgoing struts) is not reflective symmetric w.r.t. Σ (pink dashed), hence symmetric meshing is not possible. Rotating the global node orientation as indicated by the blue arrow aligns one of its symmetry planes with Σ and enables symmetric results by mirroring.

a strut s (two connected nodes) lying on Σ. We create p in two steps: first, the strut direction $d \in D|_\Sigma$ and length l with the most similar tangent to $\mathcal{S}$ around the harmonic center is selected and subsequently the strut position is optimized to minimize its distance to $\mathcal{S} \cap \Sigma$, this defines s and its end nodes. Then, the best fitting panel $p \in \texttt{Panels}(s)$ is added as the first panel of $\mathcal{Z}$.

4.3 Growing

We call nodes lying on Σ *symmetry nodes*. At any time during growing, only two of the symmetry nodes are part of the priority queue, namely the two "outer" nodes having only one incident strut in the plane. The filling procedure detailed in Section 3.2.1 relies on the halfedge data structure of $\mathcal{Z}$ and is based on covering boundary vertices to no longer lie on $\partial\mathcal{Z}$. The symmetry nodes, however, by definition always lie on $\partial\mathcal{Z}$, calling for a slightly modified boundary test to only consider the halfspace defined by Σ. The first step when growing around a popped symmetry node is to add and connect the best fitting strut, lying in the symmetry plane, to the node. To fill the gap arising between this strut and the rest of $\mathcal{Z}$ growing is performed, similarly to above, by adding panels to complete the (half-)ring around the node, i.e., so that it no longer lies on the boundary (in that halfspace).

5 Experiments

5.1 Relative Weighting of Energies

We tested different coefficients $\alpha \in [0..1]$ for the linear combinations of energies in Section 3.3. Figure 11 demonstrates the results on a hemisphere test object. Naturally, the orientation error decreases with increasing α, while the closeness error increases. Also, giving more weight to the closeness error favors shorter edges (a finer tessellation) than when only penalizing orientation error. Note that low values for α increase the risk for bad/flipped configurations causing deadlocks (requiring fixing cf. Section 3.6). Our choice of $\alpha = \frac{2}{3}$ is a good trade-off, motivated by the observation that for higher values of α the orientation error decreases only slowly while the closeness error increases more rapidly.

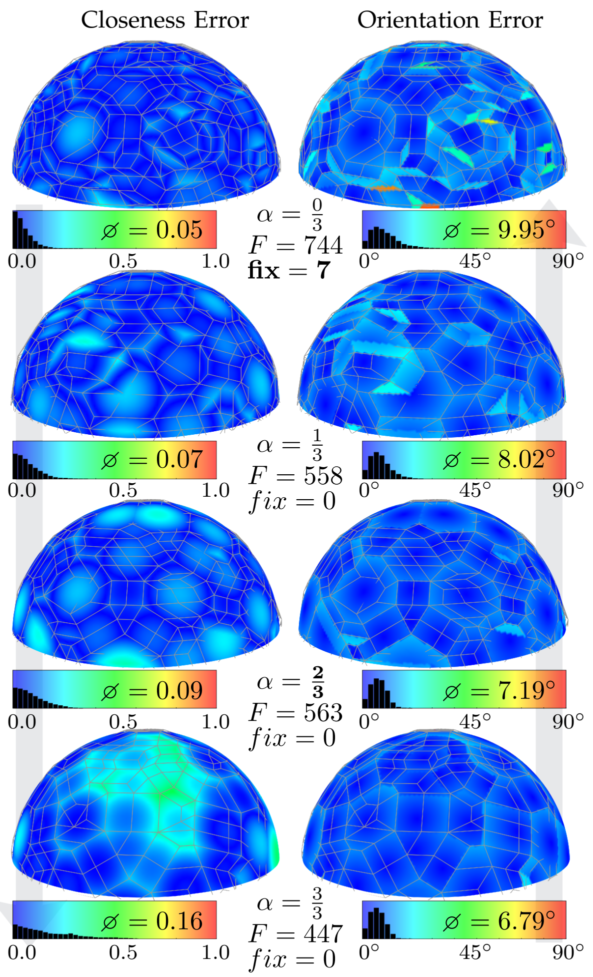


Fig. 11. Experiment: linear combination of energies on a hemisphere mesh. The errors are color-coded on the input freeform surface on a scale from dark blue (lowest) over green to red (highest). Here, the orientation error is the angle deviation. The Zome mesh wireframe is shown in gray. F is the number of faces in $\mathcal{Z}$. fix denotes the number of deadlock resolves (fixes) were needed.

5.2 Choice of Starting Position

To determine an appropriate starting position for the growing procedure we experimented with starting positions distributed at different distances from the boundary. Figure 12 shows the Child object partitioned in colored strips at different distances from the boundary. In each of these strips growing was initialized at 5 different random positions and the *time* and number of *fixes* was measured until growing was completed. The graph on the right in Figure 12 shows the average time and number of fixes per strip. The probability of good values generally improves for positions closer to the center of the object. Based on this observation, we start growing from the *harmonic center*, a natural "middle-position", which enables minimal stretch of the growing field and a more even growing speed along the front.

5.3 Choice of Node Orientation

The global node orientation potentially has an influence on the resulting Zome meshes. However, the relative

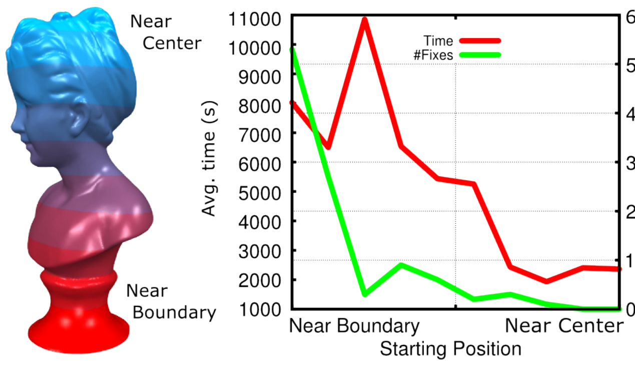


Fig. 12. Experiment: Starting position of the growing procedure. The lowest and most stable distribution of computation times and required number of fixes is achieved close to the "middle" of an object.

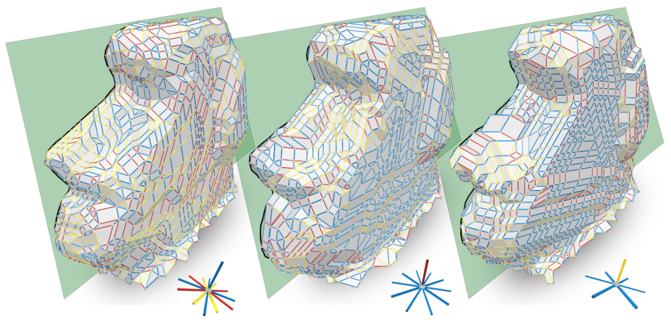


Fig. 13. Experiment: Moai mask meshed using symmetry planes of type 1, 2 and 3. A low directional resolution (node slots) in the symmetry plane leads to bad approximation of the intersection curve $\mathcal{S} \cap \Sigma$ (black) and propagation of errors into the interior of $\mathcal{Z}$. High energy values also cause longer computation times due to pruning being less effective (cf. Table 1).

frequencies $\%\{r, b, y\}$ of red, blue and yellow struts in the final Zome meshes shown in Table 1 are generally (except for the symmetry plane experiments on Moai) closely distributed around the actual availability of the respective slot directions in a system node: $(\%r, \%b, \%y) = (20\%, 48\%, 32\%)$. This suggests an already a well-balanced utilization of the available directions in D and corresponds well to the general curvature profiles of the input surfaces. Hence, the global node orientation is only optimized when using symmetry plane constraints and left to its (some) default orientation otherwise, i.e., for free growing.

5.4 Choice of Symmetry Plane

When meshing with symmetry constraints we used a Zometool plane of type 1 in all examples as this allows for the highest directional resolution along the intersection curve between the input surface and the symmetry plane. Experiments showed that a higher resolution in the plane generally also leads to better results for the rest of $\mathcal{Z}$ as it is less likely for bad decisions to arise along the plane and be propagated further. It also leads to lower computation times as pruning is more effective when low-energy solutions can be found. Figure 13 shows meshing results using planes of type 1, 2 and 3 on the Moai mask (cf. Table 1 for run-times).

6 Results

We computed Zome meshes, from a number of different freeform surfaces, on a standard i7 PC using OpenMP parallellization. The results are summarized in Table 1 and the therein bold-marked objects are shown in Figure 14. A real-life assembly of Dog's head is shown in Figure 1. The manual assembly took $5.5h$. See the project website http://www.rwth-graphics.de/zometool for more details and models.

The input to our method are disk topology surfaces, hence, if closed, the mesh needs to be cut open correspondingly. On standing objects (e.g. Child) it is natural to cut open the bottom, while for masks made for hanging against a wall (e.g. Suzanne) the back could be opened. Note that, while an infinitesimal hole suffices in theory, a heavily distorted growing field is more prone to cause problems on the front. We assume "sufficiently smooth" input surfaces and a target panel-size corre-

Model	$\lvert\mathcal{Z}\rvert$	$\square/\triangle$	$\%r$	$\%b$	$\%y$	time	mode	fix
Homer	114	0.90	25	46	29	15m	U/S1	0
	472	1.34	28	45	27	18m	U/S1	0
	1548	2.13	26	45	29	40m	U/S1	0
	5820	2.61	23	46	31	130m	U/S1	1
Duck	880	1.97	24	41	35	76m	A/S1	0
	797	**2.01**	**20**	**43**	**37**	**208m**	**A/F**	**0**
	1692	2.11	21	45	34	90m	A/S1	1
	1551	2.35	21	43	36	81m	A/F	0
Trainst.	**442**	**1.10**	**26**	**42**	**32**	**4m**	**U/S1**	**0**
Suzanne	**1354**	**1.84**	**22**	**48**	**30**	**80m**	**A/S1**	**0**
	1300	1.67	20	47	33	195m	A/F	3
Moai	2210	2.30	22	41	37	78m	A/S1	0
	2162	2.30	16	55	29	95m	A/S2	0
	2408	2.73	18	59	23	110m	A/S3	0
TradeFair	456	1.3	24	38	38	5m	A/F	0
	784	2.1	19	42	39	22m	A/F	0
Max Planck	1480	2.36	22	47	31	67m	A/S	0
Dog	**927**	**1.47**	**17**	**49**	**34**	**240m**	**A/F**	**0**
	1516	1.69	21	46	33	241m	A/F	0
Child	**1849**	**2.08**	**22**	**45**	**33**	**139m**	**A/F**	**0**
Bunny Head	1368	1.87	21	40	39	175m	A/F	0

TABLE 1
Results of our approach. $\lvert\mathcal{Z}\rvert$ denotes the number of faces of the resulting mesh and the second column the ratio of quads to triangles. The three $\%$ columns give the relativity of $r$ed, $b$lue and $y$ellow struts in the mesh. "mode" is a tuple of initialization type (User-defined or Automatic) and growing mode (Free or Symmetric). The x in Sx denotes the used plane type. "fix" is the number of deadlock repairs that were necessary.

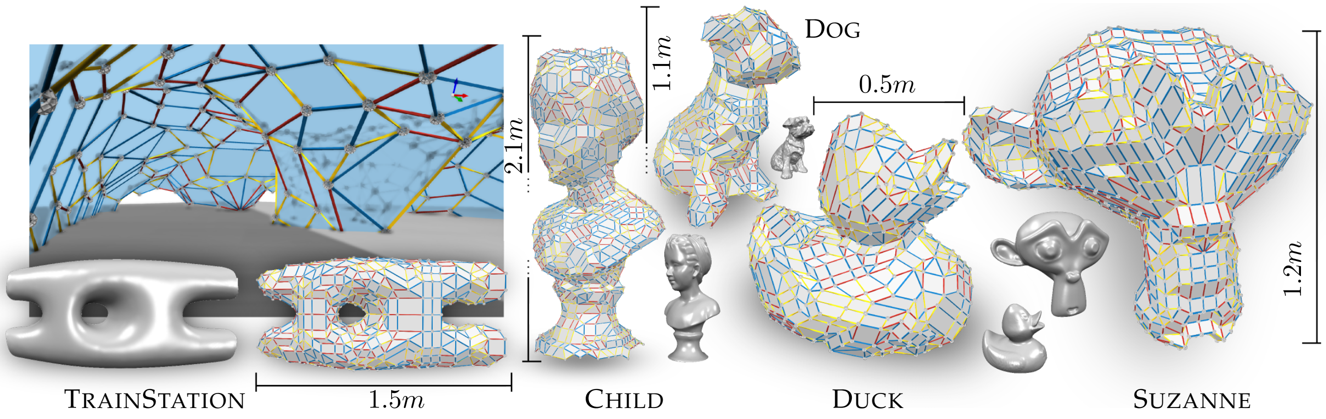


Fig. 14. Resulting Zome meshes of different freeform surfaces. The TRAINSTATION object (non disk-topology) was computed by temporarily filling the hole in the middle.

sponding reasonably well to the feature granularity of the input, still, HOMER and DOG show that also very coarse solutions and feature-rich inputs are possible.

While our method is completely deterministic, there is a certain randomness or variance involved in the results with regards to both the generated output geometry as well as the run-time. This is because slightly varying initializations (e.g., starting position or target edge length) can cause quite different local configurations at another point during growing. While this is less critical for the actual approximation quality of the output, since segments are filled in an optimal manner regardless of the actual segment configuration, the run-times can vary strongly depending on the local configuration and surface smoothness. The reason lies in poor pruning performance in certain configurations together with the used, structured filling strategy. When filling a segment from “left” to “right” along the active strut, expecting a low-energy solution, in each iteration the possible panels are first sorted by descending energy values and recursed accordingly. This makes the pruning less effective if for example the optimal solution requires high energy panels to the left.

Zometool enables a huge re-usability potential as only 9 different struts are involved. This paper presented a method to enable efficient paneling of surfaces by defining a restricted set of panels P. As this set is not specialized for a single shape but applicable to any input, it can be mass produced. However, the fixed scaling of the struts and panels calls for sets of differently sized panels depending on the scale of the application at hand (e.g., building-sized architecture vs. toy-sized bunnies). The measures in Figure 14 specify the sizes of the objects using the standard Zometool elements. It is important to note that the relative approximation error can always be trivially reduced by simply scaling up the input object $\mathcal{S}$, which results in a finer Zome mesh $\mathcal{Z}$ (cf. Figure 15). The absolute error depends on the lengths of the struts.

6.1 Quad-to-Triangle Ratio

The output meshes are in general quad-dominant with a □/△ ratio of around 2. Due to the flexibility of triangles to better handle different curvature configurations and varying resolutions, this ratio decreases for feature-rich inputs or when computing coarse outputs.

6.2 Features

Given the finite set of angles present in the Zometool system, the sharp features common in technical objects can in general not be represented. However, given an appropriate panel resolution, details on smooth freeform surfaces can be represented as shown in Figure 15 on a series of HOMER models at different scales. Naturally, the preservation of small features cannot be guaranteed in practice (although theoretically representable), as capturing them would require the growing process to place nodes precisely at the respective feature, which is hard to achieve in general due to the arbitrary distribution of features and nature of the growing and starting position.

7 LIMITATIONS AND OUTLOOK

The proposed method still leaves room for improvement and we have identified a number of open questions and limitations, which we believe should be addressed in future work.

7.1 Topology Optimization

A high distortion of the growing field front panels intruding too much into neighboring regions can cause ugly fold-over configurations. Such distortions are more prone to arise in concave areas or around surface features, e.g., back of SUZANNE’s chin. Instead of utilizing the, rather brute-force deadlock fixing procedure, it could be possible to develop a local remeshing operator by investigating the topological region of influence of nodes located in the interior of a Zome mesh. If such an operator can be found and pre-computed for different neighborhood sizes even a kind of direct, Laplacian modeling inspired, freeform Zometool modeling is imaginable.

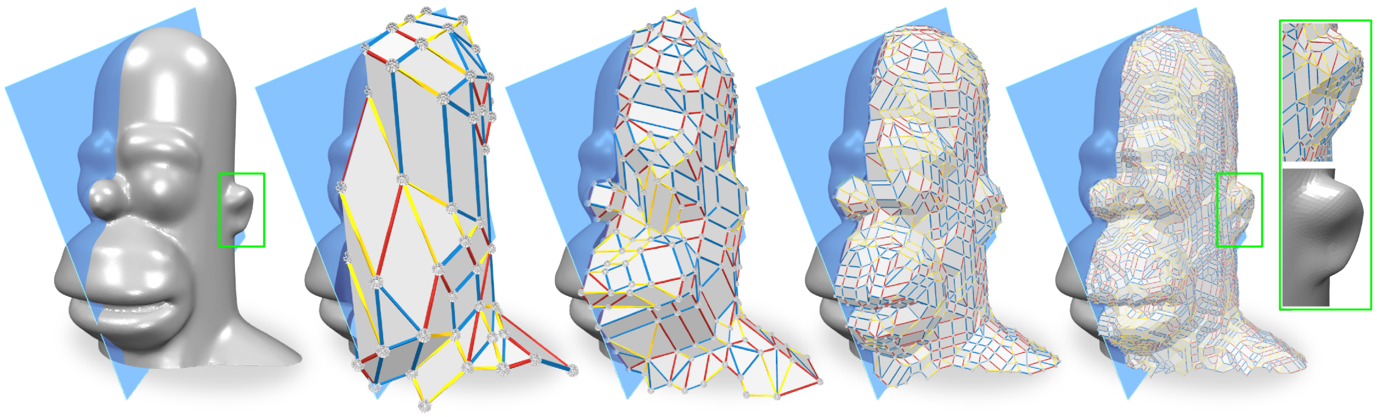


Fig. 15. Different scale quad-dominant Zometool meshings of a HOMER mask, having $0.1k$, $0.5k$, $1.5k$ and $6k$ faces. The real-life heights of the models are ca. $0.7m$, $1.4m$, $2.6m$ and $5.5m$.

7.2 Closed Surfaces of Higher Genus

We believe a generalization of the presented growing approach to non-disk topologies to be very hard. Although it is imaginable to timely identify colliding parts of the front and applying the same exhaustive filling (used for filling local segments above), to the enclosed areas, this could ultimately mean computing an optimal tessellation for the whole input, which is not tractable. The presented approach starts the growing from the center of a surface towards the boundary, which can naturally lead to a frayed boundary of the Zome mesh. However, if alternatively starting from the boundary towards the middle one could have more control over quality of the boundary, however, a closure problem similar to that of closed surfaces would result. While [31] presents an approach for closed surfaces based on locally, incrementally updating an initial solution, it, does not handle the case where one must be able to guarantee that no intersections occur and that panels are planar.

7.3 Admissible Panels

The triangles and convex, planar quads used in this work corresponded well to the harmonic growing strategy and enabled efficient intersection tests. However, with a limited set of directions and panels in the system comes a limited fairness of the results. More detailed evaluations of the obtainable fairness would be required for architectural applications. Extending the set of panels by introducing planar, convex n-gons should be straight forward, but at the cost of further increasing the exponentially growing branching possibilities of Algorithm 1. A generalization to concave panels could be possible but would require extra care in the prioritization. It is unclear how to handle the intersection tests for non-planar panels.

8 Conclusion

To the best of our knowledge, the approach presented in this paper is the first attempt at rationalizing freeform surface patches by a polygonal Zome mesh consisting only of planar, convex panels. The devised front growing strategy is guided by a harmonic field of arrival times, designed to minimize the number of problematic cases on the front, and the front itself is advanced by adding locally optimal strips of Zometool panels. By a simple, but effective, extension the method is also capable of reproducing reflectional symmetries often found in man-made characters and designs. To demonstrate the performance of our method we computed Zometool meshes for a wide range of input surfaces with different smoothness and complexities. Although finding a solution is not theoretically guaranteed, the rarely encountered problems were all solvable by our repair strategy.

The Zometool system has a huge potential for reusability and efficient realization in modern freeform architecture. We hope our work can serve as a first step towards extending the application area of Zometool to a new class of recreational and architectural applications.

Acknowledgments

The authors would like to thank the reviewers for their helpful and extensive feedback. Furthermore, we are grateful to David Bommes for introducing us to Zometool, Marcel Campen for the many technical discussions, Jan Möbius for OpenFlipper [42] and Dominik Sibbing for the spatial hashing code. This work was funded by the DFG Cluster of Excellence UMIC (DFG EXC 89).

References

[1] H. Pottmann, S. Brell-Cokcan, and J. Wallner, "Discrete surfaces for architectural design," in *Curves and Surface Design: Avignon 2006*, P. Chenin, T. Lyche, and L. L. Schumaker, Eds. Nashboro Press, 2007, pp. 213–234.
[2] H. Pottmann, Y. Liu, J. Wallner, A. Bobenko, and W. Wang, "Geometry of multi-layer freeform structures for architecture," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 65:1–65:11, 2007.
[3] Y. Liu, H. Pottmann, J. Wallner, Y.-L. Yang, and W. Wang, "Geometric modeling with conical meshes and developable surfaces," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 681–689, '06.
[4] H. Zimmer, M. Campen, R. Herkrath, and L. Kobbelt, "Variational tangent plane intersection for planar polygonal meshing," in *Advances in Architectural Geometry 2012*, L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, Eds. Springer Vienna, 2013, pp. 319–332.

[5] M. Eigensatz, M. Kilian, A. Schiftner, N. J. Mitra, H. Pottmann, and M. Pauly, "Paneling architectural freeform surfaces," *ACM Trans. Graph.*, vol. 29, pp. 45:1–45:10, July 2010.

[6] C.-W. Fu, C.-F. Lai, Y. He, and D. Cohen-Or, "K-set tilable surfaces," *ACM Trans. Graph.*, vol. 29, pp. 44:1–44:6, '10.

[7] M. Singh and S. Schaefer, "Triangle surfaces with discrete equivalence classes," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 46:1–46:7, 2010.

[8] H. Zimmer, M. Campen, D. Bommes, and L. Kobbelt, "Rationalization of Triangle-Based Point-Folding Structures," *Comp. Graph. Forum*, vol. 31, no. 2, pp. 611–620, 2012.

[9] K.-Y. Lo, C.-W. Fu, and H. Li, "3d polyomino puzzle," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 157:1–157:8, Dec. 2009.

[10] S. Xin, C.-F. Lai, C.-W. Fu, T.-T. Wong, Y. He, and D. Cohen-Or, "Making burr puzzles from 3d models," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 97:1–97:8, Jul. 2011.

[11] P. Song, C.-W. Fu, and D. Cohen-Or, "Recursive interlocking puzzles," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 128:1–128:10, Nov. 2012.

[12] K. Hildebrand, B. Bickel, and M. Alexa, "crdbrd: Shape fabrication by sliding planar slices," *Comp. Graph. Forum*, vol. 31, no. 2pt3, pp. 583–592, May 2012.

[13] Y. Schwartzburg and M. Pauly, "Fabrication-aware design with intersecting planar pieces," *Comp. Graph. Forum*, vol. 32, no. 2pt3, pp. 317–326, 2013.

[14] L. Luo, I. Baran, S. Rusinkiewicz, and W. Matusik, "Chopper: partitioning models into 3d-printable parts," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 129:1–129:9, Nov. 2012.

[15] R. Prévost, E. Whiting, S. Lefebvre, and O. Sorkine-Hornung, "Make It Stand: Balancing shapes for 3D fabrication," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 81:1–81:10, 2013.

[16] S. Coros, B. Thomaszewski, G. Noris, S. Sueda, M. Forberg, R. W. Sumner, W. Matusik, and B. Bickel, "Computational design of mechanical characters," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 83:1–83:12, Jul. 2013.

[17] D. Ceylan, W. Li, N. J. Mitra, M. Agrawala, and M. Pauly, "Designing and fabricating mechanical automata from mocap sequences," *ACM Trans. Graph.*, vol. 32, no. 6, 2013.

[18] L. Zhu, W. Xu, J. Snyder, Y. Liu, G. Wang, and B. Guo, "Motion-guided mechanical toy modeling," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 127:1–127:10, Nov. 2012.

[19] M. Skouras, B. Thomaszewski, B. Bickel, and M. Gross, "Computational design of rubber balloons," *Comp. Graph. Forum*, vol. 31, no. 2pt4, pp. 835–844, May 2012.

[20] J. Mitani and H. Suzuki, "Making papercraft toys from meshes using strip-based approximate unfolding," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 259–263, Aug. 2004.

[21] S. Takahashi, H.-Y. Wu, S. H. Saw, C.-C. Lin, and H.-C. Yen, "Optimized topological surgery for unfolding 3d meshes," *Comp. Graph. Forum*, vol. 30, no. 7, pp. 2077–2086, 2011.

[22] D. Chen, P. Sitthi-amorn, J. T. Lan, and W. Matusik, "Computing and fabricating multiplanar models," *Comp. Graph. Forum*, vol. 32, no. 2pt3, pp. 305–315, 2013.

[23] M. Kilian, S. Flöry, Z. Chen, N. J. Mitra, A. Sheffer, and H. Pottmann, "Curved folding," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 75:1–75:9, Aug. 2008.

[24] Y. Igarashi, T. Igarashi, and J. Mitani, "Beady: interactive beadwork design and construction," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 49:1–49:9, Jul. 2012.

[25] C. Yuksel, J. M. Kaldor, D. L. James, and S. Marschner, "Stitch meshes for modeling knitted clothing with yarn-level detail," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 37:1–37:12, Jul. 2012.

[26] Zometool Inc. [Online]. Available: http://zometool.com

[27] T. Davis. (2007) The mathematics of zome. [Online]. Available: http://geometer.org/mathcircles/zome.pdf

[28] X. Ni, M. Garland, and J. C. Hart, "Fair morse functions for extracting the topological structure of a surface mesh," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 613–622, Aug. 2004.

[29] S. Vorthmann. vZome. [Online]. Available: http://vzome.com

[30] E. Schlapp. ZomeCAD. [Online]. Available: http://www.softpedia.com/get/Science-CAD/ZomeCAD.shtml

[31] H. Zimmer, F. Lafarge, P. Alliez, and L. Kobbelt, "Efficient exploration of the zometool model space," *Preprint*, 2014.

[32] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene, "Recent advances in remeshing of surfaces," in *Shape Analysis and Structuring, Mathematics and Visualization*. Springer, 2008.

[33] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon Mesh Processing*. AK Peters, 2010.

[34] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin, "Quad-mesh generation and processing: A survey," *Comp. Graph. Forum*, 2013.

[35] P. J. Frey, H. Borouchaki, and P.-L. George, "Delaunay tetrahedralization using an advancing-front approach," in *5th Int. Meshing Roundtable, Sandia Nat. Lab.*, 1996, pp. 31–46.

[36] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, and S. Member, "The ball-pivoting algorithm for surface reconstruction," *IEEE TVCG*, vol. 5, pp. 349–359, 1999.

[37] C. T. Silva and J. S. Mitchell, "Greedy cuts: An advancing front terrain triangulation algorithm," 1998.

[38] J. Schreiner, C. Scheiclegger, and C. Silva, "High-quality extraction of isosurfaces from regular and irregular grids," *IEEE TVCG*, vol. 12, no. 5, pp. 1205–1212, 2006.

[39] P. Liepa, "Filling holes in meshes," in *Proc. SGP*, 2003, pp. 200–205.

[40] T. Möller, "A fast triangle-triangle intersection test," *Journal of Graphics Tools*, vol. 2, pp. 25–30, 1997.

[41] Computational Geometry Algorithms Library. [Online]. Available: http://www.cgal.org

[42] J. Möbius and L. Kobbelt, "Openflipper: An open source geometry processing and rendering framework," in *Curves and Surfaces*, ser. Lecture Notes in Computer Science, J.-D. Boissonnat, P. Chenin, A. Cohen, C. Gout, T. Lyche, M.-L. Mazure, and L. Schumaker, Eds. Springer Berlin / Heidelberg, 2012, vol. 6920, pp. 488–500.

Henrik Zimmer is a Ph.D. student in Computer Science at the RWTH Aachen University, Germany, where he completed his Diploma in 2008 with specialization on Computer Graphics and Geometry Processing. His research interests include Geometry Processing in general and Quad Remeshing and Architectural Geometry in particular.

Leif Kobbelt is a professor of Computer Graphics & Multimedia at RWTH Aachen University in Germany. He studied Computer Science and completed his PhD at the University of Karlsruhe in 1994. After a postdoc stay at the University of Wisconsin in Madison he joined the Computer Graphics Group at the University of Erlangen in 1996 and completed his Habilitation there in 1999. Shortly after being appointed an associate professor at MPI Informatik in Saarbrücken (1999), he received an offer for a full professorship from RWTH Aachen University and moved to Aachen in 2001 where he is now the head of the Computer Graphics Group. His research interests cover many areas of Computer Graphics and Computer Vision with a focus on Geometry Processing, 3D Reconstruction, Multiresolution- and Freeform-Modeling, 3D Model Optimization, and the efficient handling of polygonal meshes in interactive applications.