Optimization of 3D Models for Fabrication

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Per Henrik Joakim Zimmer
aus Väsby, Schweden

Berichter: Prof. Dr. Leif Kobbelt
Prof. Dr. Pierre Alliez

Tag der mündlichen Prüfung: 14.05.2014

Selected Topics in Computer Graphics

herausgegeben von
Prof. Dr. Leif Kobbelt
Lehrstuhl für Informatik 8
Computergraphik & Multimedia
RWTH Aachen University

Band 12

Henrik Zimmer

Optimization of 3D Models for Fabrication

Shaker Verlag
Aachen 2014

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: D 82 (Diss. RWTH Aachen University, 2014)

Printed in Germany.

ISBN 978-3-8440-2901-7
ISSN 1861-2660

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

This thesis presents a set of novel optimization approaches for dealing with three different architecturally motivated rationalization tasks: First, a planarization technique for enabling efficient (e.g., glass) panelings of tessellated freeform geometries is presented. The formulation is based on plane intersections and yields planar panels by construction. Furthermore, the used constraints are straightforward algebraic expressions of lower polynomial degree than used in various comparable methods. The generality of the method is demonstrated by application to a variety of architecturally inspired optimization problems. Then, for a new type of support structures called point-folded structures an anti-diversification technique is developed for reducing the number of geometrically different panels. By a problem-adapted parametrization and carefully designed search strategy, the shape redundancy can be reduced by over 90% for various freeform designs, enabling significant reductions of fabrication costs in practice. Finally, for the still largely unexplored high potential area of constrained tessellation techniques, i.e., tessellation algorithms restricted to using only structural elements from a predefined set, two novel approaches based on a commercially available construction system (Zometool) are presented. The first method concerns approximation of closed surfaces of arbitrary genus, and implements an effective model-exploration strategy to efficiently find a solution. Furthermore, for guaranteeing planarity of panels when tessellating the architecturally important class of freeform surface patches, a second method based on an advancing front method guided by a novel growing strategy is developed.

Acknowledgments

This work was made possible by the contribution and support of numerous people during the last years – I take this opportunity to express my deep regards and gratitude to them.

Ever since arriving in Germany as a young student last decade, Leif Kobbelt has been a source of inspiration and motivation for me, I am particularly grateful to him for giving me the opportunity to pursuit my doctorate under his expert guidance.

I thank Pierre Alliez for being the co-examiner of this work and for his willingness to always lend an ear for questions and sharing his experience in productive discussions.

A heartfelt thank you goes to my colleagues and friends in the Computer Graphics group at RWTH Aachen for making life in Aachen so enjoyable, both at and off work. In particular David Bommes, Marcel Campen, Jan Möbius, Torsten Sattler and Dominik Sibbing deserve recognition for taking the time to co-operate, co-author, proof-read and encourage.

I thank the Fold-In team at RWTH Aachen for the many fruitful discussions, in particular Karl-Heinz Brakhage, Ralf Herkrath, Susanne Hoffmann and Martin Trautz.

For initially introducing me to Computer Graphics I am obliged to Carl Loodberg.

My moving to Germany to study and ultimately finishing this work would not have been possible without the help of my late grandfather Arvid and the continuous support of my family and loving wife Anca. I especially thank my beloved parents Gun and Klas for early opening my eyes to natural sciences.

Finally, a big shout out to Amon Amarth for keeping my spirits up and my pulse beating at a productive rate during the intense last months of finishing this work.

Contents

1. Introduction

In recent years the field of *Architectural Geometry* has arisen as a symbiotic link between Architecture and Geometry Processing. On the one hand, bringing architecture forward by leveraging on decades of experience in modern geometry optimization, while on the other hand also advancing the field of Geometry Processing by posing interesting geometric challenges. Modern optimization capabilities are not only an important tool to enable new architectural concepts, but also to extend classical, well-proven construction systems to modern *freeform* designs. In contrast to regular surfaces such as spheres and conics, *free* forms refer to more general shapes such as machine parts or humanoid characters not representable by simple functions or combinations of regular shapes. The coalescence of talented artists and modern modeling software continuously contributes novel ideas to an ever growing set of rich and novel architectural designs. The realization of such complex and continuous shapes typically necessitates that they first be divided into smaller "discrete" pieces, e.g., by a polygonal tessellation.

Two fundamental aspects of modern Architectural Geometry research are related to problems arising when wanting to generalize classical structural principals to freeform designs: First, the physical rules defining the stability of a classical construction system might not be straight-forward to transfer to the freeform setting. An example being the classical, tessellated geodesic dome – made popular by Buckminster Fuller – a convex shape, where force is distributed among the inherently stable triangular elements. For tessellations of more general shapes, demonstrating also concavities, new theories must be developed. This has led to a cohort of recent research on lightweight and even self-supporting freeform structures. Second, venturing outside the realm of simple, regular shapes, *rationalization* must be performed in a clever way to avoid skyrocketing fabrication costs. Returning to the geodesic dome for an example: the regular dome shape could be approximated by a small set of different triangular elements, enabling efficient, mass production of, e.g., glass panels. Tessellations of freeform surfaces, however, typically have all different, not necessarily only triangular, elements, making element *planarity* and *diversity* central geometric challenges for enabling efficient fabrication.

In practice the two problems of *structural* and *geometric* optimization are strongly related: optimizing for structural properties alone might not yield a plausible geometry and simply being geometrically well shaped does not guarantee structural stability. Naturally, the finalization of an architectural design must entail both aspects. In general, however, the direct optimization for several, possibly even opposing, goals is very difficult, motivating separate exploration of the degrees of freedom of the respective sub-problems, in order to enable better understanding of the joint setting.

The focus of this thesis is on geometric optimization for tessellated, architectural freeform shapes, where the polygonal faces are to be clad "efficiently" with panels. The three parts of this work deal with different aspects of this rationalization problem: From planarization of panels to enable economical panel-claddings, over geometric optimization and anti-diversification for double-layer support structures – consisting not only of a single surface but two interconnected layers – to computing tessellations of freeform designs using only a constrained set of construction elements.

- *Part I – Polygon Mesh Planarization.* Based on [ZCHK13] this part presents a novel approach for optimizing given polygon meshes for planarity of faces. Planarity is an important property for enabling efficient manufacturing of panels of different materials such as glass. The technique is based on a generalization of the concept of tangent plane intersections and guarantees planarity of panels by construction. This formulation is in a sense "dual" to state-of-the-art planarization techniques and inherently allows for a rich set of guiding energy functionals and constraints.

- *Part II – Rationalization of Dual-Layer Support Structures.* Based on [ZCHK13] and [ZCBK12] this part deals with geometric optimization for two related types of double-layer support structures. Both structures are made up by two, combinatorially dual, mesh layers with different properties.

  For computing *dual-layer* space frames with planar faces the optimization formulation from Part I is extended by a set of intersection constraints to enable separate layers, free of inter-layer intersections.

  So-called *point-folded* structures are topologically equivalent to dual-layer space frames, but rely on a fundamentally different type of structural elements between the layers – *point-folded* elements. With no repetitivity optimization techniques available, recent realizations of freeform point-folded structures inevitably resulted

in the manufacturing of all different elements. A novel anti-diversification approach is presented here, which by a carefully designed combination of problem-tailored adaptive discretization and hierarchical sampling can achieve very high accuracy and enables rationalization gains of over 90%. Furthermore, it is demonstrated how the method can readily respect various, e.g., aesthetic- and production-based hard-constraints.

- *Part III – Freeform Zometool Tessellation.* Based on [ZLAK14] and [ZK14] this part presents two automatic algorithms for tackling two different problems related to efficient freeform surface tessellations based on a fixed construction system – the Zometool system. For tessellating closed freeform surfaces of arbitrary topology, an efficient algorithm is proposed which, based on a set of topology preserving modification operators, explores the shape space of Zometool models to find good approximations. Then, specifically for applications where planarity of panels is important (e.g., in architecture), a novel advancing front-based meshing algorithm is presented for rationalizing freeform surface patches with planar Zometool panels.

2. Paneled Architectural Surfaces

The focus of this thesis is on architecture-motivated geometric problems for different types of support structures based on polygon meshes. A *support structure* refers to some kind of supporting (or carrying) framework, where the mesh edges are replaced by load bearing beams or struts. Typically, the mesh faces are covered by panels for aesthetic and/or practical reasons (such as weather resistance), in some cases the panels are even structural members of the system, this is, e.g., the case of the "point-folded"-panels considered in Part II. In particular the issues regarding panel (or, more generally, element) repetitivity and quality are central here. This chapter presents a (non-exhaustive) collection of related work and concepts from the fields of Geometry Processing and Architectural Geometry.

Polygon Meshes Polygon meshes are the de-facto standard discrete representation of surfaces in Computer Graphics and Geometry Processing. Let $\mathcal{M} = (V, F, E)$ denote a polygon mesh consisting of a set of vertices (or nodes) V, a set of polygonal faces F and a set of edges E. The geometry of the mesh is defined by a 3D position $\mathbf{p}_i := \mathbf{p}(v_i) \in \mathbb{R}^3$ for each vertex $v_i \in V$. The term *vertex* is often used ambiguously to refer to both the entity v and its position $\mathbf{p}(v)$, sometimes also denoted $\mathbf{v}$. In this work only meshes that are two-dimensional manifolds (2-manifold for short) are considered, i.e., each point on the surface has a local disk-like neighborhood (or half-disk on the boundary). For a mesh $\mathcal{M}$ let $\mathcal{M}^* = (V^*, F^*, E^*)$ denote the (combinatorial) *dual* mesh, where each vertex $v \in V^*$ corresponds to a face $f \in F$ of the primal mesh and vice versa (special care must be taken at the boundaries). The geometry of the dual mesh $\mathcal{M}^*$ is not uniquely defined, but is often expected to be "similar" to $\mathcal{M}$, e.g., by placing the dual vertices to lie at the barycenters of the primal faces. Refer, e.g., to [BKP*10, Ede01] for more details on these fundamentals.

Continuous surfaces are typically discretized based on generalization of the regular plane tilings (triangular, quadrilateral and hexagonal). The generalization being that irregular vertices are allowed (and by the Euler characteristic even required) to represent

general surfaces of arbitrary genus. We refer to these meshes as tri, quad and hex meshes respectively. Irregular vertices in a mesh lead to irregular polygons in the dual mesh, e.g., a valence 5 vertex in a triangle mesh leads to a pentagon in the dual mesh. Assuming that the majority of primal vertices have regular valence k (e.g., $k = 6$ in the case of triangle meshes) such a dual mesh is referred to as k-gon-*dominant* (e.g., hex-dominant). Where it is clear from the context *dominant* is often left out for brevity. Once a continuous surface has been discretized into a mesh, there exists a wide range of *remeshing* techniques for converting between different surface types or for modifying and optimizing meshes of the same type. [BKP*10, BLP*13, AUGA08, Ede01] form a comprehensive overview on state-of-the-art in mesh generation, processing and remeshing.

Mesh Quality In [PBCW07] Pottmann et al. list a number of properties which should be fulfilled by polygon meshes in order to enable efficient geometric support structures:

- Planarity of faces
- Static properties
- Low vertex valence
- Constraints on the arrangement of supporting beams

The first two properties are inherently fulfilled by triangles meshes[1], explaining their widespread usage in modern support structures. Yet, such triangle-based structures come at the cost of not fulfilling the last two properties: the high valence can lead to non-trivial node configurations, which pose difficult constraints on the merging of beams at the nodes, and in fact excludes a whole class of useful support structures having so-called torsion-free nodes (cf. Section 2.2). In contrast, higher-order polygons enable fruitful optimization of such structures which can additionally have lower weight (cf. [PLW*07]). However, extra care must now be taken to guarantee face planarity and good structural behavior. A further important mesh property not mentioned in the above work is: low diversity of elements to support efficient (mass) manufacturing.

Twist or torsion in a structure is by no means a purely geometric problem, also from the point-of-view of structural engineers (e.g., [SSAK]), the realization of non-optimized

[1] Triangles are considered inherently stable in the sense that, given three fixed-length edges connected by hinge-joints, the shape of the triangle is well-defined (pre-determined). However, in a corresponding setting with quads or higher-order polygons, extra attention might be required as the elements display one or more kinetic degrees of freedom.

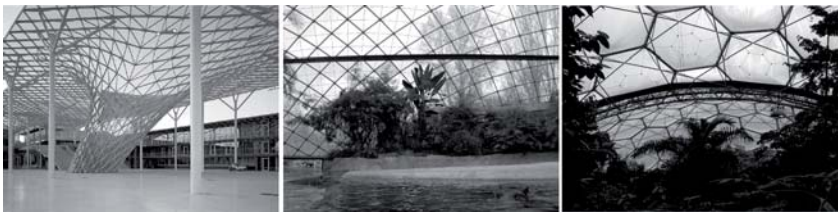
freeform designs is complex. The optimization of structural properties is outside the scope of this thesis (focusing on geometric problems), but nevertheless the statics of lightweight and self-supporting support structures is an active research field, recent publications of which are briefly summarized in the next paragraph.

Structural Optimization Modern Architectural Geometry research on mesh-based, self-supporting structures is typically based on the *thrust network analysis* [BO07, Blo09]. Here modern optimization capabilities are utilized to enable venturing outside the well-known construction shapes and concepts of classical lightweight systems and enable self-supporting *freeform* constructions. Recent publications include optimization of self-supporting quad meshes with planar faces [VHWP12] and triangle based structures where adaptive tessellations are used to allow for better force distribution [LPS*13]. Work is also done on the ancient topic of self-supporting masonry designs [DGAOD13, PBSH13] including procedural modeling of stable masonry buildings with rules based on physical constraints [WOD09].

2.1. Meshes with Planar Faces

The quadratic approximation power of piecewise linear representations make triangle meshes well suited for the task of surface approximation. However, for general polygon meshes the vertices of a face typically do not define a unique plane and the described surface geometry of the mesh is not well-defined. While still useful as, e.g., subdivision base meshes in animation or NURBS control meshes in CAD/CAM scenarios [Bom12], for applications directly relying on the given mesh and not some underlying smooth interpretation, a non-ambiguous face geometry is often needed. In particular, for cladding tessellated architectural designs, planarity of faces is often essential to allow for efficient manufacturing and functioning of panels, e.g., made out of glass (cf. [Sch03, GSC*04]) or foil cushions (cf. [KSAZ01]). A polygon mesh with planar faces is called P-mesh. For the standard polygon types, let PT, PQ and PH mesh refer to a tri, quad and hex mesh with planar faces. Figure 2.1 shows tessellated architectural designs based on PT, PQ and PH meshes. It is important to note that, besides the tri mesh in subfigure (a), the PQ and PH meshes are not true freeforms – the Hippo House being based on a translational surface and the Eden Project consisting of (intersecting) spheres. Directly extracting P-mesh tessellations from non-trivial shapes can be considered impossible and, in general, given quad or hex mesh discretizations of continuous *freeform* surfaces do not have

planar faces. To achieve face planarity, a *planarization* optimization must be carried out. Before discussing planarization algorithms in the next subsection, the following paragraph briefly notes an interesting connection between meshes with planar faces and (continuous) differential geometry.



(a) (b) (c)

Figure 2.1.: Paneled architectural tessellations of smooth surfaces. (a) The Milan Trade Fair – a glass-clad structure using triangular panels in curved areas (PT/PQ mesh). (b) The Hippo House at the Berlin Zoo – a glass-clad quad mesh with planar faces (PQ mesh). (c) The Eden Project, Cornwall – a mesh with air filled foil cushions in hexagon-shaped planar faces (PH mesh). From left to right, the images are courtesy of M. Fuksas (`www.fuksas.it`), Nayrb7 (`www.flickr.com/photos/nayrb7`), and Karen Roe (`www.flickr.com/photos/karen_roe`).

Connection to Differential Geometry In [BS08] Bobenko and Suris provide a discretization of concepts of classical differential geometry based on PQ meshes as the discrete surface representation, with some ideas dating back to R. Sauer, cf. [Sau70]. In particular, PQ meshes can be shown to be a discrete analogon of (continuous) conjugate curve networks on surfaces. This connection is noted and also explicitly utilized in different quad planarization approaches (e.g., [LPW*06, ZSW10, LXW*11]). In fact, in [ZSW10] the authors also explicitly note the converse case: that a well behaved planarization cannot be expected on meshes where the quads are not aligned along a network of conjugate curves. Here the quad case was explicitly mentioned, however, discretizations of various differential geometric concepts exist also for tri and hex meshes.

Planarization Algorithms

We first consider the PQ case in more detail. State-of-the-art quad remeshing algorithms (converting tri meshes to quad meshes) are often motivated by applications in animation

or simulation and typically strive to align the quads along directions of principal curvature on the input surface for reasons of approximation quality and aesthetics [Bom12]. The network of principal curvature lines is a conjugate network on the input surface – meaning such remeshing methods can already yield appropriate initial solutions for further planarization optimization. Planarization methods are typically based on nonlinear numerical optimization of functionals expressing the planarity of the faces on the one hand and geometric faithfulness to the input on the other hand. Also, more complex optimizations problems can be defined by prescribing additional functionals and even enforcing planarity as hard-constraints. Usually the vertex positions are used as natural degrees of freedom for the optimization – to be faithful to the original geometry these positions shall then be minimally perturbed to obtain a P-mesh.

Planarity of faces is expressed and optimized for differently by state-of-the-art methods. Liu et al. [LPW*06] and Wang et al. [WLY*08] use *cubic* polynomials describing the volumes spanned by combinations of four face corner points, i.e., if for a face all such volumes are zero, then the face is flat. Other formulations include minimizing the distance between diagonals of quad faces as done by Zadravec et al. [ZSW10] and optimizing interior angles of quad faces (to sum up to 2π) [LPW*06]. However, enforcing high order polynomial hard-constraints during optimization is a complex matter. In practice, for large problems planarity is often enforced by penalty methods instead, i.e., the relative weight of a planarity functional is increased until sufficiently planar faces are obtained. Our VTPI (variational tangent plane intersection) framework described in [ZCHK13] is different – here a dual formulation over the plane equations is used, i.e., the variables describing the face planes (of the resulting planarized mesh) are unknowns in the optimization and the planes are optimized such that the resulting intersection points remain geometrically faithful to the input. This formulation only requires *quadratic* hard-constraints, and the planarity is implicitly guaranteed for all solutions due to the dual formulation over plane equations.

Besides techniques based on numerical optimization, there are also different "non-polynomial" formulations of planarity, which use entirely different methods for the minimization. Two such methods are the *Planarizing Flow*, which is a technique based on the Laplacian operator for general polygon meshes defined by Alexa and Wardetzky [AW11], and the *Shape-Up* framework by Bouaziz et al. [BDS*12] based on geometric projections, which allows for defining complex, abstract functionals in a geometric manner without given closed form expression.

It should be noted that quad meshes obtained by state-of-the-art remeshing methods, although typically aligned to direction fields based on principal curvatures, are not necessarily optimal for planarization. The principal curvatures are only one possible set of conjugate directions, additionally they are unique, leaving no additional degrees of freedom for influencing the geometry of the resulting PQ mesh. For this reason Zadravec et al. [ZSW10] and Liu et al. [LXW*11] do not rely on given quad meshes as input, but design PQ meshes by first optimizing the underlying *conjugate direction fields* (discrete counterpart of conjugate curve networks on continuous surfaces) before extracting and planarizing the corresponding quad meshes.

For planarizing hex meshes similar optimizations can be used as in the quad case. However, for covering a given freeform surface by planar hexagons, additional effort is typically spent on generating good initial solutions, in a sense similar to [ZSW10, LXW*11]. Both C. Troche [Tro08] and Wang et al. [WLY*08] note the connection between local Gaussian curvature of the input surface and the shape and quality of the resulting planarized panels – correspondingly both start by computing "well-suited" triangulations (aligned along conjugate directions on the surface) from which then good initial hex-dominant meshes can be obtained (by dualization). While [WLY*08] follows up with a planarity optimization step, [Tro08] directly computes a PH mesh by intersecting tangent planes on the triangle mesh. In fact our VTPI framework is a generalization of this idea. However, by posing the intersection computation as an optimization problem instead, additional computational robustness and design control can be gained.

Variational Surface Approximation The goal of the above techniques was to planarize polygon meshes approximating freeform surfaces. Irregular vertices are required when tessellating freeforms, but the meshes were assumed to have regular faces, e.g., quads or hexagons. We now mention a different approach for covering architectural freeform surfaces with meshes consisting of arbitrary planar polygons, which, however, generally have valence 3 vertices. First, for the task of surface approximation the *Variational Shape Approximation* method [CSAD04] by Cohen-Steiner et al. partitions similarly oriented parts of a freeform surface and replaces them by approximately planar polygons. Similar to a Voronoi Diagram, this partitioning generally leads to polygons meeting at valence 3 vertices. In [CW07] this method was extended to allow for perfectly planar panels based on explicitly computing plane intersections similar to [Tro08]. However, as noted by the authors, forming such plane intersections can be numerically problematic. Part I of this thesis shows how a variational (VTPI) formulation alleviates these problems.

2.2. Geometric Support Structures

The above optimization algorithms worked on polygon meshes with dimensionless entities, such as zero-width edges and zero-thickness faces. Naturally, for the actual manufacturing of parts and assembly of a structure, real-sized entities must be considered. Intuitively, a simple support structure could be derived from a polygon mesh by using cylindrical struts for the edges which are "somehow" joined in spherical nodes at the vertices. It comes as no surprise that this principle is in fact a classical connector system adopted by several support structure manufacturers, such as MERO TSK[2], cf. [SSAK]. Still, the "somehow" can be quite complex, each node can end up requiring special attention, due to the arrangement of incident edges being geometrically different in general. Furthermore, to accommodate for the non-zero dimensionality of the struts and nodes, non-trivial parts would need to be cut out from the assumed regular (e.g., quad or hexagon) panels in order not to intersect with the support structure itself. One way around this problem is to extend the support structure by a second (or more) connected layers to hold the panel cladding and/or improve static properties. One real-life example being the hex-tri/hex double-layer structure of the Eden Project shown in Figure 2.1(c), for more information cf. [SSAK, KSAZ01].

Based on a certain kind of offset meshes, H. Pottmann and co-authors (e.g., [PBCW07, PLW*07, LPW*06]) define different *multi-layer* support structures with optimized beams and nodes, suited for paneling architectural meshes. In these methods, as discussed in the next section, further layers are obtained by extruding (offsetting) a base mesh, hence the layers have the same connectivity. Following that discussion the so-called dual-layer support structures are introduced. These are multi-layer structures, where the layers are not offsets of the same mesh but have dual connectivities (similar to the Eden Project).

Multi-Layer Support Structures

Following [PBCW07] we restrict to considering PQ meshes for the introduction. At the end the corresponding important concepts for tri and hex meshes are briefly discussed.

Bobenko and Suris [BS08] define a PQ mesh $\mathcal{M}^+$ to be a *parallel mesh* of the PQ mesh $\mathcal{M}$ if the two are related by a discrete Combescure transformation, i.e., all vectors of corresponding edges between the meshes are parallel (or equivalently, all faces are parallel [PBCW07]). Intuitively, the fact that the corresponding parallel edges between

[2] `http://www.mero.de`

two such parallel meshes are also co-planar can be used to define a support structure: *beams* can be formed by extruding the edges of $\mathcal{M}$ and connecting them to the respective edges of $\mathcal{M}^+$. These beams have the advantage of planarity, i.e., exhibit no twists (or torsion). Still, at the nodes the arrangements of meeting beams can be complex. Nodes at which the (planes of) incident beams do not align nicely along a single (cylindrical) axis are said to have *torsion*. In [LPW*06, PBCW07] the advantages of torsion-free (or optimized) nodes for freeform structures clad with glass panels is discussed. Meshes with torsion-free nodes are called *conical*, as the planes incident to a node are tangent to a common right cone (cf. Figure 2.2). Conical meshes were introduced by Liu et al. in [LPW*06]. In their work, the authors present an angle condition defining conical vertices (refer to Figure 2.2 for notation):

$$v \text{ is conical } \Leftrightarrow \beta_0 + \beta_2 = \beta_1 + \beta_3.$$

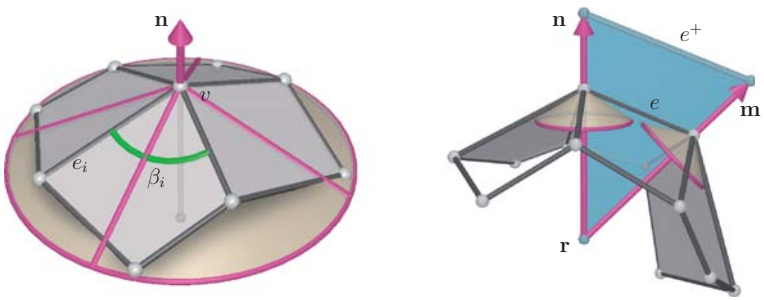


Figure 2.2.: A conical vertex is defined by having all incident face planes tangent to a right cone. In (a) the pink lines from the apex denote the common face/cone tangents. The inner face angles incident to a vertex v are denoted by β_i, the incident edges by e_i, and the cone axis by $\mathbf{n}$. In a conical mesh, the cone axes $\mathbf{m}$ and $\mathbf{n}$ of two adjacent vertices intersect at a point $\mathbf{r}$, e^+ denotes a parallel offset of the edge e.

Conical meshes admit a special kind of parallel offset meshes having constant face-to-face distances. Consequently, beams of a torsion-free support structure can be constructed in a straightforward fashion by offsetting the faces about a constant distance and connecting the correspondingly offset edges, yielding thin-plate beams. Additionally the beams can be thickened by an extrusion orthogonal to their respective planes.

The introduction of conical meshes marks an important theoretical and practical advance in multi-layer structures for freeform geometry. Note, however, that while the offset faces are at constant distances from each other, the edges have varying heights. This is typically more an aesthetic, than a functional problem, as all beams could still be manufactured with the same (tallest) height. This would not influence the ability of the beams to align in one axis at the nodes and they can still be leveraged to give a smooth impression on the "visible" side of the structure.

To achieve beams of constant height, in [PLW*07] H. Pottmann and co-authors investigate multi-layer structures based on constant edge/edge offsets. This turned out to be a quite restrictive class of meshes, which, in contrast to conical PQ meshes, cannot approximate arbitrary shapes. A further class of PQ meshes are *circular* meshes (e.g., [BS08, PW08]) which corresponds to offsets with constant vertex/vertex distance. Circular meshes are PQ meshes where the corners of each face lie on a circle, this is in a sense dual to conical meshes.

The high valence of nodes in tri meshes make them too rigid to allow for non-trivial conical meshes: only planar and spherical shapes are possible. However, circular tri meshes are trivial as three points define a circle. Conversely, for hex meshes: all PH meshes are conical, since three planes (in general position) define a right cone at the common intersection point, circular PH meshes are restricted to planes and spheres.

Dual-Layer Support Structures

Part II of this thesis deals with multi-layer support structures consisting of two polygonal layers with dual connectivities. Such support structures can be seen as a generalization of classical 2D trusses, which obtain stability by two layers interconnected by triangular elements. A real-life example of a multi-layer structure involving combinatorially dual layers is the hex-tri/hex arrangement seen in the Eden Project, cf. Figure 2.1(c).

Figure 2.3(a) shows a classical 2D truss. Hinted by the green arrow in the figure, the top and bottom layer are "dual" in the sense that each beam on the upper layer corresponds to a node on the lower layer. Trusses are still popular in modern lightweight construction, e.g., of efficient, low weight bridges (e.g., [LeM08]). Figure 2.3(b) shows a space frame (3D truss) consisting of pyramidal elements. Space frames are popular for roof constructions spanning large areas at low weight. In a freeform setting, again care must be taken to enable an efficient paneling for weather resistance and insulation.

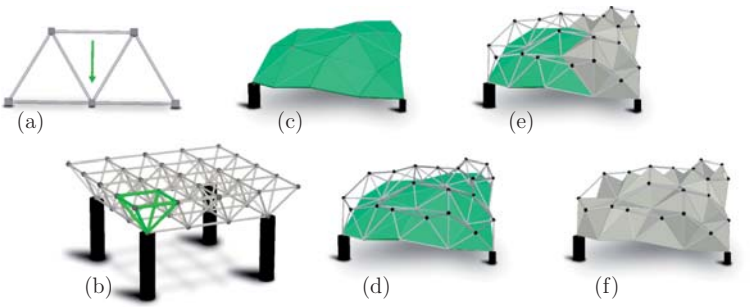


Figure 2.3.: (a) regular 2D truss, (b) 3D truss (or space frame), (d) a mock-up freeform space frame to stabilize a tri mesh design (c). (e,d) demonstrate so-called point-folded structures, with the structural elements consisting of metal sheet pyramids.

Dual-Layer Space Frames with Planar Faces Figure 2.3(d) shows an example of a more general space frame based on a freeform tri mesh. Here, instead of the quad-based pyramids used in the regular example in Figure 2.3(b) a tetrahedral element was erected on each triangle. One advantage of such a construction is the stability provided by the triangular elements. Unfortunately, in practice, paneling the tri mesh (as shown here in green) is rather involved due to the non-trivial node configurations. Also the dual layer cannot be directly be clad with panels as the (hex-dominant) faces are not planar in general. However, if the second dual layer can be planarized, an efficient paneling is made possible by the conical property of the nodes (all having valence 3). The computation and planarization of the second layer cannot be done independently of the first layer – inter-layer intersection must be prevented. This is the topic of Chapter 5.

Point-Folded Structures In Part II also another type of dual-layer structure is considered, which provides both watertightness and structural stability without necessitating a planarization: "point-folded structures" can have the same topological structure as the space frames, but instead of steel struts connecting the two layers, triangle-based pyramidal elements are used, where the creases assume the structural properties of the space frame struts and the faces provide a covering (cf. Figure 2.3(e,d)). Just like a simple crease can transform a sheet of paper from a fluttering to a much stiffer state, folding or creasing of, e.g., thin metal sheets can be used to create elements of highly increased inherent stiffness without adding mass. Hence, they are appealing building blocks in

the construction of light-weight, self-supporting structures without the need for heavy beams or additional interior support structures. By the folding analogy the pyramidal elements used here are also referred to as "point-folded" elements, a term used in work by M. Trautz and co-authors (e.g., [TH09, HT11, TA11]).

In principle this idea is not new and dates back to ideas developed by Buckminster Fuller in the early 20th century. Figure 2.4(a) shows a fulleresque dome with metal pyramids, here referred to as a *triangle-based point-folded structure*, (b) shows a cylindrical *quad-based* point-folded structure with transparent polycarbonate pyramids, and (c) shows a dome-shaped *hexagon-based* point-folded structure with steel sheet pyramids. Although this type of paneling generally does not requires any planarization, the realization of *freeform* designs can quickly be hampered by high production costs relating to the different pyramid geometries. In the regular, simplified setting of dome and cylinder shapes, the diversity of elements can be more easily controlled. Point-folded elements and their production are further detailed in Chapter 6. In Chapter 7 we present a geometric optimization for reducing the number of different base pyramids, allowing

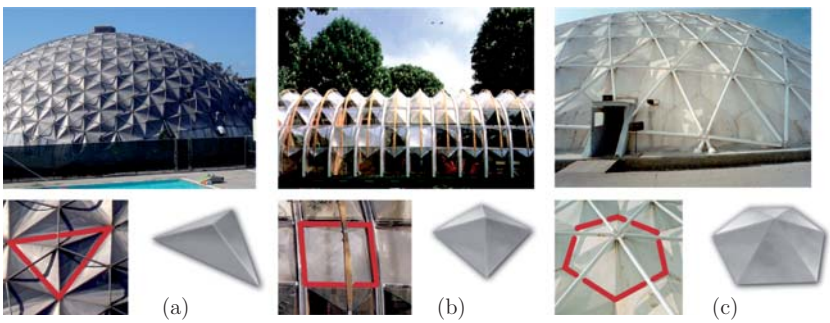


Figure 2.4.: Three (polygon-based) point-folded structures based on regular geometries: a triangle-based dome-shaped structure (a), a quad-based cylindrical structure (b) and a hexagon-based dome-shaped structure (c). Subfigure (a) is the dome gymnasium of Palomar College, image courtesy of a San Diego Reader photographer. Subfigure (b) portrays the IBM traveling pavilion designed by Renzo Piano [PN89], image courtesy of Martin Lisnovsky (`http://arquitecturamashistoria.blogspot.com`). Subfigure (c) is the Wood River railcar service facility, image courtesy of Karl Hartig (`http://www.karlhartig.com`).

for efficient, high accuracy production of such elements. This method is classified as an *Anti-Diversification Optimization* technique and introduced in Section 2.3 below.

More General Structures

The types of support structures introduced above where based on (possibly multiple layers of) polygon meshes with straight edges and typically planar faces. To briefly summarize a few different types of recently investigated freeform rationalizations: In [BPK*11] Bo et al. optimize multi-layer type of structures with circular arc edges, Schiftner et al. derive support structures from circle and sphere packings on tri meshes in [SHWP09]. In [DPW11] Deng et al. investigate more general types of aesthetically and functionally motivated curve-webs on surfaces. Stepping away from the typical tessellations altogether, in [PSB*08] Pottmann et al. optimize for and cover surfaces by developable strips. In [KFC*08] curved folded (developable) surfaces are analyzed and computed for various designs.

2.3. Anti-Diversification Optimization

The techniques mentioned above dealt primarily with reducing construction costs relating support structure construction and assembly (optimized, conical nodes), and enabling efficient panel production by optimizing for planarity. However, besides geometric panel properties, the diversity of elements is another key aspect, which can heavily influence the fabrication costs when realizing architectural freeform designs. *Anti-Diversification* describes the goal of a range of approaches, which by modifying the shape to maximize panel similarities aim at maximizing the number of serializable parts, or equivalently, minimizing the number of parts requiring custom manufacturing. Such an optimization can be critical, e.g., for scenarios dealing with curved panels, where molds (i.e., master-shapes) are used for shaping the panels, e.g., by slumping (glass) or deep-drawing (steel sheets). In the worst case each unique panel relies on a custom-made mold, which can quickly lead to skyrocketing production costs.

Recent anti-diversification attempts in Architectural Geometry reveal two fundamentally different approaches to the problem. In the following called *Object-Derived Anti-Diversification* are methods which, based on the input geometry, derive an (in some sense) optimal set of least diverse construction elements, which can be used to approximate the original shape. The advantage being that very good approximations can be

obtained. The disadvantage is that the parts are tailored for one particular shape and can generally not be re-used for realizing another design. The term *Fixed-System Anti-Diversification* is taken to refer to methods, which enable even better reutilization, by using a fixed set of construction elements (independent of the current input) to approximate any given design. Such a system can also be allowed a wider basis of molds/parts, as they are not only reused for constructing multiple elements of one specific design, but also across multiple designs.

Object-Derived Anti-Diversification In 2010 three research papers addressing similar element diversity problems were presented [FLHCO10, SS10, EKS*10]. The methods by Fu et al. [FLHCO10] and Singh et al. [SS10] are methodologically quite similar, but operate on different classes of input, quad and tri meshes respectively. Both rely on iterative, non-linear optimization involving clustering of similar panels (faces), computing a representative, canonical panel for each cluster, and a global optimization for incorporating the geometry of the canonical panels in the mesh. In [SS10] all the above steps are iterated until a user-specified tolerance has been reached, meaning the solution relaxes from iteration to iteration and the clustering adapts to the changed geometry. In contrast, in [FLHCO10] all steps are performed only once, here more effort is spend on the initial clustering, considering not only the geometric similarity of panels, but also costs relating to the compatibility of incident panels. Finally, a single step of numerical optimization of the geometry is applied to accommodate for the new panel shapes. Note that this method optimizes for minimal diversity of quad panels, planarity is not considered.

Unlike the methods above, the method by Eigensatz et al. does not directly aim at minimizing the diversity of panel geometries, but rather focuses on the underlying molds, typically used to create curved panels. The admissible types of molds (and panels) are limited to a set of, also double-curved, surface classes (e.g., planar, cylindrical and toroidal) and the method not only optimizes for panel geometry and continuity, but also for maximization of mold reuse. Molds are the master-shapes from which the panels of the different classes can be fabricated. Note that even geometrically quite different panels of the same class can be created from one and the same mold by positioning the panel blank differently on the mold. The method is based on an iterative optimization consisting of two parts: (1) a discrete optimization where a weighted Set Cover problem is solved for computing assignments between surface panels and possible molds of different classes/costs, and (2) a continuous part where the surface shape is updated to

accommodate for the changed mold (panel) geometries. This approach shares conceptual similarities with our approach [ZCBK12], which deals with the task of computing a minimal set of molds (or dies) for efficiently manufacturing "point-folded" elements. We cast the problem in an angle-space, where, similar to [EKS*10], the search for an optimal set of molds is formulated as a Set Cover problem. However, in contrast to the other approaches, in our setting the geometry of the given input design is considered fixed and our optimization only relies on degrees of freedom derived from design or fabrication related tolerances pertaining to the second layer. As noted by Feige in [Fei98], a greedy algorithm for set covering is the best-possible approximation algorithm with polynomial time complexity – a fact utilized in both approaches. This algorithm chooses subsets in the order of decreasing cardinality (or cost) until the whole universe is covered.

Due to the inherent discreteness, solving anti-diversification tasks, as the ones described above, is computationally very expensive. The above methods all share similar limitations, where the input mesh complexity is restricted to a couple of thousand faces and timings are typically in the range of several hours.

Finally, for the problem of (extremely) simplifying 3D models by a small set of significant, textured planes, the *plane-space* discretization and greedy Set Cover strategy used by Décoret et al. in [DDSD03] is in principle similar to the greedy search in *angle-space* used in [ZCBK12].

Fixed-System Anti-Diversification The first question that arises in scenarios, where a fixed construction system is to be used, is: *which system is appropriate?* With "appropriate" depending heavily on the application at hand. Manufacturers of support structures typically have company-specific systems for realizing with different classes/-types of surfaces. E.g., MERO TSK has a wide range of different *node* or connector types (some of which are discussed in [SSAK]), while the corresponding (lengths of the) beams vary with the design and, consequentially, so do the shapes of the panels in general. Now the question arises, if this can be taken even further? Meaning, *are there practically useful systems, which consist of a fixed set of nodes, beams and possibly even panels?* The answer is somewhat affirmative: in fact, there are some construction systems (originally intended as toys) with potential to be of interest also for architecture.

The classic LEGO (`http://www.lego.dk`) system consists of a small set of simple brick-like pieces and thus, has great anti-diversification potential. There has been recent work done using the LEGO system in the fields of Computer Graphics and Geometry

Processing (e.g., [SPC09, TSP13, Nak13]). However, the goal of these approaches was not architecture – one problem is that the block structure of the system is not well suited for realizing smooth designs and furthermore does not map well to the here considered tessellated/paneled surfaces.

A fixed construction system, which is not only a toy, but is also used in research and teaching, is the Zometool system (`http://www.zometool.com`). It consists of a single node or connector type and (in the standard version) only uses a total of 9 different edge (or strut) types. The system is based on icosahedral/dodecahedral symmetry with 62 fixed directions prescribed by the slots in the node. In each slot a corresponding type of strut can be attached (and fixed) – this yields a stability in the system, enabling single-layer shapes (even without using panels) in contrast to systems based on magnetic nodes, as discussed further below.

Part III of this thesis details our two recently presented methods for applying the Zometool system for different freeform surface tessellation tasks, these methods are briefly summarized in the following. In [ZLAK14] we present a general approximation technique for tessellating surfaces of arbitrary genus with a polygon mesh consisting of Zometool nodes and edges, a *Zome mesh*, guaranteed to be 2-manifold and consisting only of triangles and quads. The technique is based on a simulated annealing-type method, which efficiently explores the space of Zome meshes around a given shape. High quality results can be obtained within few minutes, and it is possible to compute also quite coarse tessellations using only a low number of elements, while still preserving thin surface features. Due to the used approximation energy functional, the resulting quad faces are often close to being planar, however, there are no guarantees.

For the restricted, but architecturally essential, case of (disk topology) freeform patches, in [ZK14] we present a fundamentally different, deterministic approach, which enables computing an approximating Zome mesh with planar, convex panels. The method implements an advancing front growing, where, in each step, a surface piece along the current front is conquered and covered by Zometool panels in an optimal way. The used set of panels consists of the 29 different triangles available in the Zometool system and 118 different convex, planar quads. While a total of 147 panels might seem a lot at first, one recalls that the fundamental idea of fixed-system approaches is not to use these panels only once for tessellating a certain input design, but, on the contrary, enable mass production and application to any appropriate input.

Finally, as an outlook, another type of interesting construction systems which could find use in similar tessellation scenarios are systems based on a single magnetic node

type and (typically) a single edge type. One such system is GeoMag (`http://www.geomagworld.com`), which additionally consists of a very small set of four different panels (an equilateral triangle, two quads and a pentagon). While being more restrictive in the sense of having only a single edge, the system allows for a continuous range of different directions as the struts can slide freely over the magnetic nodes. This, however, requires additional care to enable structural stability – either by using double layers or the panels available in the system.

Nevertheless, the properties of these kinds of fixed systems and the initial research that has been carried out already indicate potential for tasks such as freeform paneling, making this an interesting direction for future research. At time of writing and to the best of our knowledge, there have been no other algorithmic approaches dealing with the Zometool system for freeform approximation. Also for GeoMag-type systems, there is no known research in such topics. However, the *Connectivity Shapes* of [IGG01] are in a sense similar to a GeoMag system with "rigidified" nodes – in their paper Isenburg et al. optimized for 3D embeddings of shapes defined only by their connectivity graph and the goal that all edges have constant length, i.e., be equal, as in the GeoMag case.

2.4. Conclusion

For efficiently paneling architectural designs, the recently developed theories on optimized nodes in general and multi-layer structures in particular mark important milestones in the field of Architectural Geometry. In particular for efficiently paneling single-layered shapes, the approach by Eigensatz et al. [EKS*10] proved efficient and general. Nevertheless, when dealing with new or special types of construction systems, such as the "point-folded structures" in [ZCBK12], specialized problem parametrizations and solutions are required. Concerning anti-diversification techniques dealing with fixed systems, not much work has been done so far. We believe this field to have great practical potential and it is also predicted to yield many worthwhile theoretical problems.

In general, all problems discussed here are very hard to solve, relying not only on non-linear continuous optimization, but typically also dealing with discrete degrees of freedom. This complexity can even further drastically increase for practical applications, where not only a single property is to be optimized for, but where combinations of different qualities and simultaneous optimization of their respective problems is required.

Part I.

Polygon Mesh Planarization

The faces of general polygon meshes are typically not *planar*, meaning the k corner vertices of a k-gonal face need not lie in a common plane. Polygon mesh planarization, or simply *planarization* for short, is the process of converting such a mesh with non-planar faces to one in which all faces have well-defined supporting planes, while the shape deviates only minimally from the original.

Often motivated by architectural considerations the type of input meshes to planarization methods are usually smooth surfaces without sharp features. The discussion in Chapter 2 revealed that a planarization endeavor cannot be expected to succeed on every tessellation of the input, e.g., optimally, quads should be aligned along conjugate directions on the surface. Still, even with recent advances in quadrilateral meshing, care has to be taken, since the quads in the resulting meshes seldomly conform 100% to this paradigm. For this reason certain planarization techniques rely on direct control also over the tessellation generation part of the process, e.g., first modifying the underlying direction fields and subsequently extracting meshes optimized for planarization purposes. However, in the process of rationalizing a design such explicit control over all stages of the pipeline is not always given.

This part deals with the planarization of given, fixed connectivity polygon meshes. The goal is to control the planarization process and compute well-behaved outputs even on inputs which are sub-optimal in the above sense. Chapter 4 presents our planarization technique. The technique is based on plane equations in the form of controlled tangent plane intersections, and, in contrast to methods simply penalizing non-planarity using energy functionals, inherently guarantees planarity of the mesh faces. Furthermore, the constraints arising in our optimization formulation are of lower polynomial degree (quadratic) than in related works, where planarity is expressed, e.g., via 3D volumes (cubic).

Tangent plane intersections are a special case of (general) plane intersections, where the configuration of planes involved in an intersection is typically derived from the (local) connectivity of a polygon mesh. E.g., assuming tangent planes have been defined for each vertex of a mesh, then an intersection arrangement can be computed for each face by intersecting the tangent planes of the incident vertices. Note the use of the word "arrangement" – since, except for triangular faces, the so computed intersections are not single, well-defined points in general. Even for pure tri meshes, the three tangent planes of the vertices incident to a face need not define a point of intersection, e.g., in the case of co-planarity of the tangent planes. In other words, this "standard" setting of tangent plane intersections is quite rigid and is of little use for planarizing general polygon

meshes. To this end Chapter 3 first generalizes the concept of tangent plane intersection and embeds it into an optimization setting, thereby eliminating the rigidity and enabling controlled intersection behavior, even between an arbitrary number of planes. The resulting formulation allows for a rich set of constraints and guiding functionals and can be applied in various scenarios involving planar polygon meshes. Applications are demonstrated in the chapters of the current part as well an in Part II of this thesis.

3. Variational Tangent Plane Intersection

This chapter reviews the basic concepts of tangent plane intersections (TPI) and presents our variational formulation called *Variational Tangent Plane Intersection* (VTPI). By taking an optimization approach to tangent plane intersections, VTPI enables more control over the intersection arrangements: by additional degrees of freedom, stable computations can be performed in co-planar regions and even with a number of planes different from 3. By properly constraining the problem and guiding the result by different energy functionals a useful, general tool applicable to a range of different problems involving planar polygon meshes is obtained.

We start by reviewing the concepts of ordinary 3D plane intersections. Let π_i be a plane in 3-space defined by a normal $\mathbf{n}_i \in \mathbb{R}^3$ ($\|\mathbf{n}_i\| = 1$) and a point on the plane $\mathbf{v}_i \in \mathbb{R}^3$. All points $\mathbf{x} \in \mathbb{R}^3$ on the plane fulfill the plane equation:

$$\mathbf{n}_i^\top \mathbf{x} = \mathbf{n}_i^\top \mathbf{v}_i. \tag{3.1}$$

The common points (intersection) between a pair of non-parallel planes is a line in 3-space defined by the two corresponding plane equations. Three planes lying in general position intersect at a single point $\mathbf{x}$. This point of intersection is defined by three plane equations and can be expressed by a simple 3×3 linear system as follows:

$$N\mathbf{x} = \mathbf{b} \;\Leftrightarrow\; \begin{bmatrix} \mathbf{n}_0^\top \\ \mathbf{n}_1^\top \\ \mathbf{n}_2^\top \end{bmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{n}_0^\top \mathbf{v}_0 \\ \mathbf{n}_1^\top \mathbf{v}_1 \\ \mathbf{n}_2^\top \mathbf{v}_2 \end{pmatrix}. \tag{3.2}$$

By inverting the system matrix the intersection point is obtain as $\mathbf{x} = N^{-1}\mathbf{b}$. Note that if the planes are not in general position then N is singular and cannot be inverted (no unique intersection point exists).

Above the three planes were assumed fixed and their intersection point was unique and pre-determined. Now, reversing the argument, any arbitrary point $\mathbf{x} \in \mathbb{R}^3$ can be

defined by an appropriate choice of planes. In fact, there are infinitely many triplets of planes yielding the same intersection point. This under-determinedness is the key to obtaining the necessary degrees of freedom for the variational formulation of tangent plane intersections (VTPI) we presented in [ZCHK13].

3.1. Tangent Plane Intersections

The advantages of polygon meshes with planar faces and PH meshes in particular was discussed in the introduction, e.g., conical vertices. With tri meshes being a standard mesh representation in Computer Graphics and since hexagonal tilings are dual to regular triangular tilings [Cox73], a natural way to obtain PH meshes is by intersecting the vertex tangent planes of tri meshes. This way the vertices of the hex mesh (dual to faces of the tri mesh) arise as the intersection points of the tangent planes situated at the three corners of each triangle, and the hexagonal faces (dual to tri mesh vertices) are naturally planar, since the vertices of each hexagonal face have the tangent plane of the central vertex in common (cf. Figure 3.1). Note that if the tri mesh is not regular, i.e., it has vertices with valence $\neq 6$, the corresponding faces in the dual mesh consequently do not have 6 corners. Such meshes are referred to as hex-dominant.

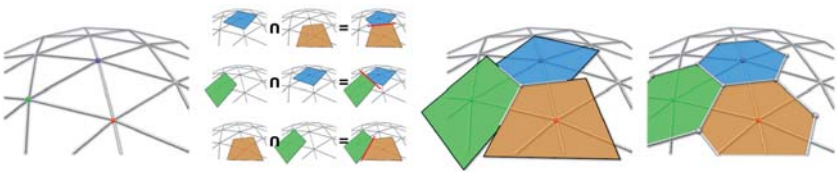


Figure 3.1.: The pairwise intersections (red lines) between the tangent planes of the three corner vertices of a triangle form the outgoing edges of a vertex in the (dual) PH mesh. Continuing this for all triangles cuts out planar hexagonal pieces.

PH meshes by TPI In 2008 Christian Troche presented an idea to compute PH meshes based on the tangent plane intersections [Tro08]. Basically, for each triangle of a given tri mesh the corresponding 3×3 equation system (cf. Equation 3.2) is inverted to yield the intersection point (vertex of PH mesh) as mentioned above. While this method works well in certain scenarios, its overall applicability is hampered by unstable and unexpected results on general shapes:

1. Singular N. Where the Gaussian curvature is zero and two planes are co-planar the matrix N does not have full rank and the solution is no longer a unique intersection point $\mathbf{x}$ but an entire solution space (e.g., a line).

2. Unwanted $\mathbf{x}$. Even where the intersection point is well-defined, it might lie at an unexpected and unwanted position. Looking at the graph duality in the regular case, one expects the dual points (hex mesh vertices) to lie near the center of gravity of the primal faces (triangles). However, by slightly tilting one tangent plane the intersection point can suddenly lie far away from this expected position.

These two problems, exemplarily demonstrated in Figure 3.2, motivated developing the VTPI approach explained next.

Variational Formulation

By adopting an optimization based approach to tangent plane intersections (as opposed to the direct inversion of N) a more stable and more general method is obtained. The idea is based on the observation that $N\mathbf{x} = \mathbf{b}$ does not have to be explicitly inverted but can instead be used directly as linear equality constraints. For an input tri mesh with n_F triangles let $N_j\mathbf{x}_j = \mathbf{b}_j$ denote the intersection constraint for triangle j. A simple formulation of the optimization problem is then:

$$\text{minimize } E(\{\mathbf{x}_j\}) \quad \text{s.t.} \quad C_{\text{int}} := \{N_j\mathbf{x}_j - \mathbf{b}_j = 0\}_{j=1}^{n_F}, \tag{3.3}$$

where $\{\mathbf{x}_j\}$ is the set of n_F unknown intersection points, E an energy functional and C_{int} the set of intersection constraints. The number of unknown variables is $3n_F$ (the 3D coordinates of the intersection points). With this formulation the above problems can now be solved as follows:

1. Singular N. The constraint formulation already alleviates the stability problem related to the inversion of N, but the constraint is still fulfilled by a whole space of solutions. We exploit this effect by defining an energy functional $E_{\text{pos}}(\{\mathbf{x}_j\}) = \sum_{j=1}^{n_F} \|\mathbf{x}_j - \hat{\mathbf{x}}_j\|^2$ to allow specifying preferred positions $\hat{\mathbf{x}}_j$ for the intersections in case of such ambiguities. Now, where the intersection point is not unique the minimization can choose as $\mathbf{x}_j$ the point in the solution space closest to $\hat{\mathbf{x}}_j$ in the least-squares sense. Note that by the nature of the constraints the energy functional has no effect where N has full rank.

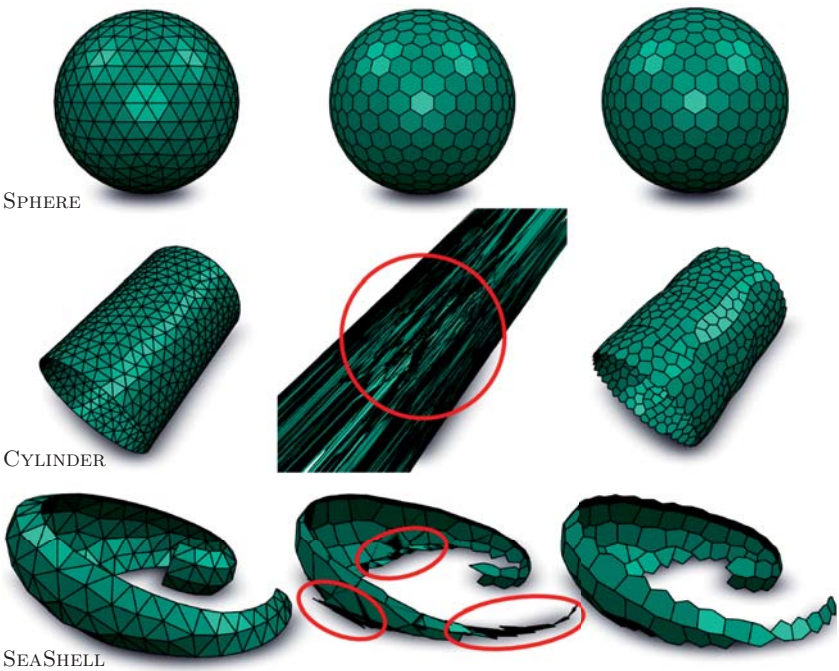


Figure 3.2.: PH meshes are computed for the three input tri meshes in the first column. The middle column shows the result using the standard TPI method and the last column shows the results obtainable by VTPI. Problematic cases are highlighted with red circles. The TPI Cylinder result demonstrates the numerical instability of inverting the matrix N on a surface with zero Gaussian curvature and the SeaShell result shows how, even on doubly curved surfaces, the direct TPI suffers from degeneracies caused by "unwanted" positions. Using the VTPI formulation the intersection points can be kept close to "preferred" positions on the input surface. The SeaShell model is courtesy of the Chair for Structures and Structural Design (`http://trako.arch.rwth-aachen.de`) at the RWTH Aachen.

2. Unwanted $\mathbf{x}$. To solve the second problem, i.e., prescribing preferred positions also where the intersection points *are* well-defined, more degrees of freedom are

needed in the optimization. This can be done by considering all terms in the plane equations to be unknown variables in the optimization. First, the normals $\mathbf{n}_i$ of the n_V tri mesh vertices $\mathbf{v}_i$ can be "set free" to allow the tangent planes to rotate around their respective fix-points $\mathbf{v}_i$. However, while this can lead to better intersection points for some triangles, keeping the planes pinned at the vertices is still quite rigid. By additionally decoupling the planes from the mesh vertices a much richer formulation is obtained. This is done by adding an additional offset variable $h_i \in \mathbb{R}$ along the normal direction for each plane. Each plane equation (Equation 3.1) can now be rewritten as: $\mathbf{n}_i^\top(\mathbf{x}_j - (\mathbf{v}_i + h_i\mathbf{n}_i)) = 0 \Rightarrow \mathbf{n}_i^\top\mathbf{x}_j = \mathbf{n}_i^\top\mathbf{v}_i + h_i$, where only the mesh vertex $\mathbf{v}_i$ is a constant.

Variational Formulation of Tangent Plane Intersection The above solutions lead to a richer and more powerful formulation but at the same time increases the complexity of the optimization problem in Equation 3.3. Counting the total number of unknowns now yields $3n_F + 4n_V = 3n_F + 3n_V + n_V$ (the 3D coordinates of the intersection points and normals plus the scalar offsets). With variable normals the (previously linear) intersection constraints C_{int} turn into quadratic constraints and additionally a new set of quadratic constraints is required to guarantee $\|\mathbf{n}_i\|^2 = 1$. The VTPI optimization problem is thus a quadratically constrained quadratic program of the following form:

$$\begin{aligned} \text{minimize } E(\{\mathbf{x}_j\}) \quad \text{s.t.} \quad & C_{\text{int}} := \{N_j\mathbf{x}_j - \mathbf{b}_j - \mathbf{h}_j = 0\}_{j=1}^{n_F} \\ & C_{\text{norm}} := \{\|\mathbf{n}_i\|^2 - 1 = 0\}_{i=1}^{n_V}, \end{aligned} \tag{3.4}$$

where $\mathbf{h}_j \in \mathbb{R}^3$ are the three offsets corresponding to the planes of the corner vertices of face j. Note that in the simple formulation where $E = E_{\text{pos}}$ only the intersection points are directly part of the energy functional, the normals and offsets are connected via the constraints. Naturally, different energy functionals can be used (and combined) to control the output. This is done for the mesh planarization application in Chapter 4 and for two other exemplary applications briefly discussed in Section 3.4 of this chapter to demonstrate the versatility of VTPI. Different energy functionals are described in Section 3.2.

Multiple Planes

Besides the additional control and stability, the variational formulation has one very important advantage over the standard TPI setting: It is not restricted to intersections

between triplets of planes. The VTPI formulation imposes the plane intersections via equality constraints and does not rely on inverting a (quadratic) 3×3 matrix N. This means that intersection points between *any* number of planes can be enforced simply by adding more rows (plane equations) to the constraints in C_{int} (cf. Equation 3.4):

$$N_j \mathbf{x}_j - \mathbf{b}_j - \mathbf{h}_j = 0 \Leftrightarrow \begin{bmatrix} \vdots \\ \mathbf{n}_k^\top \\ \vdots \end{bmatrix} \mathbf{x}_j - \begin{pmatrix} \vdots \\ \mathbf{n}_k^\top \mathbf{v}_k \\ \vdots \end{pmatrix} - \begin{pmatrix} \vdots \\ h_k \\ \vdots \end{pmatrix} = 0. \tag{3.5}$$

Thus, where the use-case for the standard TPI is restricted to creating dual, hex-dominant meshes with planar faces for given tri mesh inputs, VTPI can now deal with arbitrary topologies. Given an polygon input mesh the VTPI optimization outputs a dual mesh with planar faces.

3.2. Energy Functionals

To enable sufficient control over the intersection process the simple positional energy functional E_{pos} is generally not enough. The VTPI-based applications demonstrated in Chapter 4 and Section 3.4 rely on various linear combinations of the different functionals listed in the following. While for certain effects, such as normal smoothness, quadratic functionals suffice, some effects such as length-based rigidity require quartic expressions. Note that higher-order functionals can quickly deteriorate the performance of the optimization. The VTPI-based application presented later demonstrate the effects of the various energy functionals.

Quadratic Energies

Preferred Normals Energy Preferred normal directions $\hat{\mathbf{n}}_i$ with $\|\hat{\mathbf{n}}_i\| = 1$ can be prescribed by

$$E_{\text{norm}}(\{\mathbf{n}_i\}) = \sum_{i=1}^{n_V} \|\mathbf{n}_i - \hat{\mathbf{n}}_i\|^2.$$

Preferred Offsets Energy For creating structures at an offset from the given input surface, preferred offsets $\hat{h}_i \in \mathbb{R}$ can be specified by

$$E_{\text{off}}(\{h_i\}) = \sum_{i=1}^{n_V} (h_i - \hat{h}_i)^2.$$

Fairness Energy The layout of the dual vertices (intersection points) can be "regularized" by keeping each vertex close to the center of gravity of its neighbors:

$$E_{\text{fair}}(\{\mathbf{x}_j\}) = \sum_{j=1}^{n_F} \|\mathbf{x}_j - \frac{1}{|N_1(j)|} \sum_{k \in N_1(j)} \mathbf{x}_k\|^2,$$

where $N_1(j)$ is the 1-ring vertex neighborhood around the (dual) vertex j.

Smoothness Energy A smooth normal field can be obtained by minimizing the variation between normals of neighboring planes:

$$E_{\text{smo}}(\{\mathbf{n}_i\}) = \sum_{i=1}^{n_V} \frac{1}{|N_1(i)|} \sum_{k \in N_1(i)} (1 - \mathbf{n}_i^\top \mathbf{n}_k),$$

where $N_1(i)$ is the 1-ring vertex neighborhood around the (primal) vertex i.

Edge Vector Energy A simple form of (coordinate-based) rigidity can be expressed by prescribing edge directions $\hat{\mathbf{d}}_{jk} \in \mathbb{R}^3$ to edges jk in the dual mesh:

$$E_{\text{vec}}(\{\mathbf{x}_j\}) = \sum_{jk} \|(\mathbf{x}_j - \mathbf{x}_k) - \hat{\mathbf{d}}_{jk}\|^2.$$

Considering both lengths and directions of edge vectors, this energy functional implicitly leads to a global shape preservation.

Quartic Energy

Face Rigidity Energy Let $l_{jk} = \|\mathbf{x}_k - \mathbf{x}_j\|^2$ be the squared length between the points $\mathbf{x}_j$ and $\mathbf{x}_k$. Given initial lengths $\hat{l}_{jk}$ for face edges and diagonals, denoted jk, rigidity based on preserving these lengths can be expressed by:

$$E_{\text{len}}(\{\mathbf{x}_j\}) = \sum_{jk} (l_{jk} - \hat{l}_{jk})^2.$$

This energy functional is useful for preserving the shape of all (or individual) faces as it only considers lengths and not the coordinates of the embedding.

3.3. Optimization

The optimization problem in Equation 3.4 is a quadratically constrained quadratic program. To solve it we use `Ipopt` [WB06], an efficient, open source implementation of the *dual-primal interior-point method*. This method is similar to the barrier method (also an interior-point method) but commonly performs better, cf. [BV04] for more details on interior-point methods. A central component of such methods is the linear solve required for the Newton search direction. `Ipopt` offers interfaces to various free and commercial linear solver packages. For solving the VTPI problem we use the HSL `MA57` solver [HSL11]. This solver is optimized for sparse problems of small to medium size and can utilize the fill-in reducing ordering of `METIS` [KK95]. For the optimization we supply both the first and second order partial derivatives of the energy functionals and of the constraints. For ease of implementation and to simplify interfacing with `Ipopt` the `c++`-wrappers from [BZK12] were used. The effects of the different energy functionals regarding optimization performance and output quality is demonstrated in the scope of the applications shown in the following sections and chapters of this part.

3.4. Applications

Dual Meshes with Planar Faces

Open problems related to the computation of hex-dominant meshes with planar faces include dealing with complex, arbitrary genus input surfaces and guaranteeing valid (non self-intersecting) output (cf. [WL10, Tro08, WLY*08]). In this section we demonstrate the (multi-)plane intersection capabilities of VTPI for computing dual meshes with planar faces for given non-trivial, high genus (and also irregular) input tri meshes. Our experiments show that, although there are no formal guarantees, by using appropriate combinations of energy functionals (in particular utilizing the fairing energy) self-intersections could be avoided in all shown examples.

Tessellation (in)-dependence Generally, the tessellation (connectivity *and* geometry) of the input tri mesh can heavily influence the planarized dualization. Figure 3.3 shows two regular tri meshes of a torus with identical connectivities but different vertex positions. Dualizing and optimizing for face planarity leads to significantly different results due to the optimization starting from different initial solutions. The right example

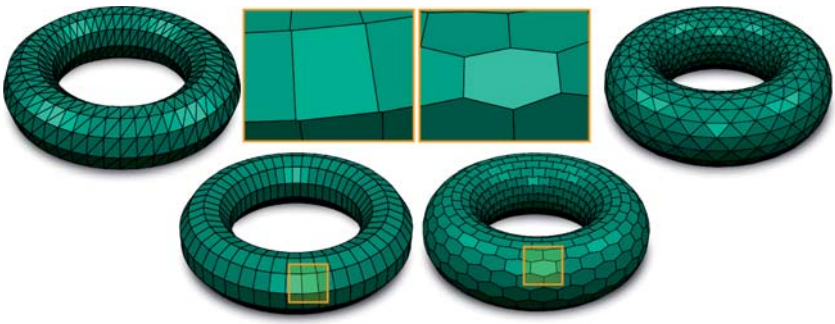


Figure 3.3.: Computing planar dualizations of two topologically equivalent (but geometrically different) tessellations of the same object can lead to different results due to different initialization of the optimization (energy functionals). From an opposite point-of-view this reveals the degrees of freedom supplied by the energy functionals for guiding the output.

shows the typically "expected" result with honeycomb and bowtie shaped hexagons in parabolic and hyperbolic surface regions respectively, whereas in the left example the hexagons have degenerated to quads. This tessellation dependency is also the foundation of state-of-the-art methods such as [Tro08, WLY*08], in which the first step is to compute (in a growing fashion) a suitable triangle tessellation (a process not trivial on complex shapes with arbitrary topologies). Our approach however, is based on another way of interpreting Figure 3.3: namely that the same output can be obtained from any of the two inputs by simply using the appropriate energy functionals. Hence, our method takes as input a (fixed-tessellation) tri mesh and uses a combination of various energy functionals to obtain good results.

Giving high importance to the positional energy leads to close, although not necessarily smooth, approximations in general. The positional energy is also stable in the sense that even when assigned arbitrarily high importance, the result is kept close to the given input. To influence the smoothness of the result a combination of fairness and normal variation energies can be used. These energies are similar in that they both have a smoothing effect and both can shrink and/or degenerate the surface when not regularized by other energies. The basic effects of these energies is demonstrated in Figure 3.4. Subfigure (c) shows the result of computing a planarized dual of the De-

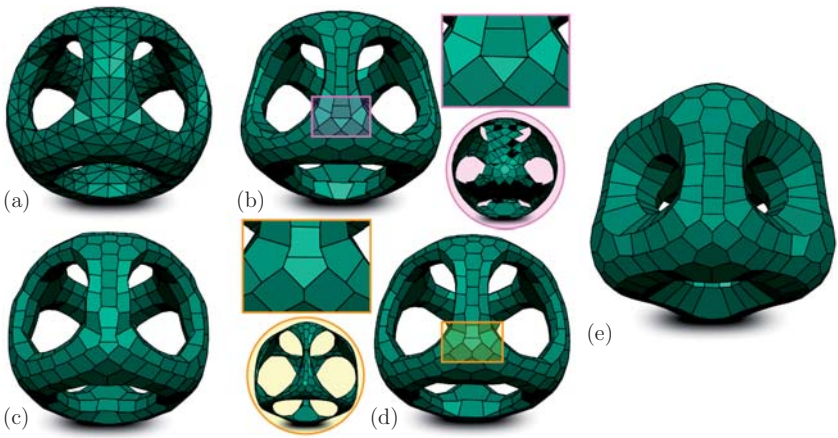


Figure 3.4.: Planarized duals of the input model in (a) computed using different energy functionals are shown in (b-d). In (c) only positional energy is used, (b) additionally penalizes normal variation for a smoother appearance and (d) adds a fairing term to help separate nearby vertices (cf. square close-ups). The circled images show the effects of assigning too high important to the normal variation and fairing energy. (e) demonstrates the offset energy. The here used DecoCube model is courtesy of Chi-Wing Fu et al. [FLHCO10].

coCube input (a), a tri mesh with $n_F = 960$ faces, using a positional energy. Although a very good approximation, the silhouette in (c) appears slightly more jagged than the original. A smoother result is obtained when additionally penalizing normal variation (b). Unfortunately, this can increase the occurrence of degenerate faces and edges, where several vertices end up at nearby positions (cf. pink square). Given too high importance, the normal variation energy can further degenerate the shape to a thin spherical shell (cf. pink circle). By also applying a fairness energy (d), nearby nodes can be pulled apart as shown in the orange square close-up. However, again care needs to be taken, as the fairness ultimately causes a shrinkage (cf. orange circle). Subfigure (e) demonstrates the offset energy. Run-times on the DecoCube where between $5s$ using only positional energy (c), $50s$ when additionally using normal variation (b) and $90s$ in combination also with fairing (d). The steep rise in computation times is due to the conflicting natures of

the positional energy, trying to keep the nodes in place, and the smoothing energies, in particular the fairing, which (in the limit) tries to contract everything to a single point.

The KITTEN, with $n_F = 3352$ faces, in Figure 3.5 demonstrates a similar setting. Here, in (b) an energy was used to preserve the normals on the input surface (a). However, these normals are not always appropriate for the faces of the planarized dual and can lead to overlapping faces (cf. close-up). By using a fairing energy these overlaps are avoided in (c). This example also confirms the run-time remarks from above: computing the KITTEN in (b) required $360s$, while for the overlap-free mesh with higher fairing $1300s$ are needed.

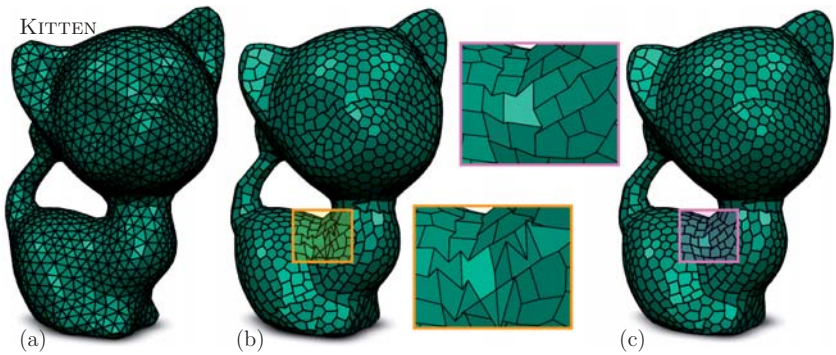


Figure 3.5.: (b) shows a dual planarization where an energy was used to prescribed preferred (face) normals from the (vertices) of the input mesh in (a). This can lead to overlapping and intersecting faces. Using a fairing energy the vertices could be pulled apart and an intersection free result is obtained (c). The KITTEN is courtesy of the AIM@SHAPE shape repository (`http://shapes.aim-at-shape.net`).

Connectivity vs. Symmetry Figure 3.6 demonstrates the importance of using symmetric input meshes when computing dual planarization of symmetric, man-made shapes. Naturally, irregular triangulations lead to irregular dual planarizations. However, while irregular triangulations themselves can still appear aesthetic, irregularities can often be more noticeable in a dual representation. The effects of such input irregularities on organic shapes such as the KITTEN are generally less apparent.

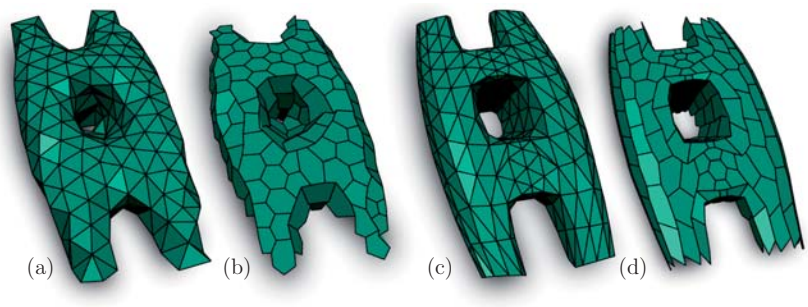


Figure 3.6.: Planarized duals (b,d) of two input triangulations (a,c). For man-made, symmetric shapes symmetric input meshes are usually required for qualitative results.

Variational Shape Approximation with Planar Faces

For the task of reducing the complexity of a model while minimally effecting its visual appearance, the *Variational Shape Approximation* (VSA) method [CSAD04] by Cohen-Steiner et al. first partitions the model's surface into connected clusters of similarly oriented faces that are then subsequently replaced by *nearly* planar panels. To each partition belongs a planar proxy geometry, and the partitioning is computed by iteratively (1) adjusting the proxy to best represent the orientation of faces in its partition and (2) modify (grow/shrink) the partitions by a Lloyd-based relaxation.

For architectural applications the VSA was extended by Cutler and Whiting in [CW07] to yield perfectly planar panels based on explicitly computed intersections of (proxy) planes. The authors note that the most challenging part of that approach is that not every partitioning allows for a qualitative planar paneling (when using explicit intersection computations). In particular the problems of co-planar planes not intersecting at all and slightly tilted planes leading to unwanted positions are mentioned. Note that these are exactly the problems, which motivated the development of, and which are solved by, VTPI (cf. Section 3.1).

Figure 3.7 demonstrates the applicability of VTPI to the problem of computing coarse shape approximations with (perfectly) planar panels. Here, a positional energy is used to keep the intersection points close to the partition corner points and the proxy normals are prescribed as preferred normals of the resulting polygonal faces.

Figure 3.7.: Variational Shape Approximation with perfectly planar panels. First column: Mesh triangles clustered by similar orientation. Second column: Extracting the polygons defined by cluster corners does not lead to planar faces in general. Third column: result of VTPI optimization. Models courtesy of the AIM@SHAPE shape repository (`http://shapes.aim-at-shape.net`).

3.5. Conclusion

VTPI is a generalization of the typical tangent plane intersections involving three planes. By a variational formulation further degrees of freedom were introduced to control the resulting output and to even allow (or rather enforce) multiple planes to intersect in a single point. One straightforward application was the computation of PH meshes from given tri meshes. This was even possible for complex surface with higher genus. Further-

more, it was showed how VTPI can be used for the more general task of approximating complex input shapes with anisotropic, planar panels. The next chapter presents the main topic of this part of the thesis: VTPI-based polygon mesh planarization.

Finally, it is interesting to note that similar variational plane formulation is used in [DPW11] to find optimal planes for defining families of planar space curves on surfaces.

4. Polygon Mesh Planarization

This chapter describes a polygon mesh planarization method based on the variational tangent plane intersection (VTPI) framework described in Chapter 3. Let $\mathcal{M}$, $\mathcal{M}^*$ and $P\mathcal{M}$ denote a polygon mesh, the dual of a mesh and a mesh with planar faces respectively. As mentioned in Section 3.1, for a given mesh, VTPI outputs a dual mesh with planar faces, i.e., $\text{VTPI}(\mathcal{M}) \rightarrow P\mathcal{M}^*$. Consequently, the problem of polygon mesh planarization, where the goal is to compute a planarized mesh with the *same connectivity* as the given input mesh, can be solved by simply using the dual of the given mesh as input to VTPI, i.e., $\text{VTPI}(\mathcal{M}^*) \rightarrow P\mathcal{M}$. Unfortunately, this intuitive solution is restricted to closed meshes, as the connectivity dualization is not well defined at boundaries. Luckily, this is only a practical problem related to the dualization process. As explained in the previous chapter VTPI can deal with any number of planes, even one or two (common for vertices on the boundary), by prescribing preferred positions for the intersection points. The general solution to planarizing meshes of arbitrary topology, is therefore to not explicitly form and use the dual mesh $\mathcal{M}^*$ as input, but rather use $\mathcal{M}$ and define the (dual) planes implicitly, e.g., one plane equation for each *face* of $\mathcal{M}$.

The planarization method presented here uses a combination of the energy functionals defined in Section 3.2 and is also based on the `Ipopt` framework for optimization. In the following section various energy functionals, appropriate for planarization, are evaluated and the planarization method is compared against two recent state-of-the-art techniques.

4.1. Planarization Energy

In the context of architectural geometry, mesh planarization is applied to enable efficient fabrication by making the faces of tessellated designs planar, while only minimally changing the aesthetics of the shape. To enable control and preservation of shape, here

the positional energy E_{pos} is linearly combined with the two rigidity energies E_{vec} and E_{len}, yielding the following planarization energy:

$$E_{\text{planarization}} = w_{\text{pos}}E_{\text{pos}} + w_{\text{vec}}E_{\text{vec}} + w_{\text{len}}E_{\text{len}},$$

defined by the triplet of weights $(w_{\text{pos}}, w_{\text{vec}}, w_{\text{len}})$. The functionals are initialized based on the entities of the given input mesh (the shape to be preserved), i.e., the preferred positions for E_{pos} are set to be the current vertex positions, the preferred edge vectors for E_{vec} are derived from the edges of the given mesh and for E_{len} the current lengths of edges and face diagonals are computed. In the same way, the unknowns in the optimization can be naturally initialized by setting the intersection points to be the vertex positions of the mesh, setting the normals to be the approximated normals of the non-planar mesh faces and setting the offsets to zero.

4.2. Evaluation

To investigate the behavior of the VTPI-based planarization, experiments were conducted on three different quad meshes (Fandisk, Fertility, Feline) computed by Mixed-Integer Quadrangulation (MIQ) by Bommes et al. [BZK09].

Energy

Figure 4.1 intuitively shows the behavior of the different functionals on these three input meshes by alternatingly setting one of the energy functional weights (w_{pos}, w_{vec}, w_{len}) to a high value and the rest low (denoted by "H" and "L" respectively). E_{vec} turns out to well preserve the aesthetics of the input and the best results are generally given by the LHL or HHH combination. Prioritizing either only position (HLL) or face rigidity (LLH) yield less smooth results.

Table 4.1 lists the corresponding timings, which are split into two parts: t_{opt}, measuring the time spent in the `Ipopt` optimization loop (without function evaluations) and t_{fcn}, measuring the time spent on function evaluations (including Jacobians and Hessians). The positional energy is the simplest and most efficient one, being quadratic and only involving one point at a time it has a sparse Jacobian/Hessian footprint and its evaluation requires less arithmetic operations than the other functionals. E_{vec} is also quadratic, but has a slightly denser footprint in the system as it involves two unknowns.

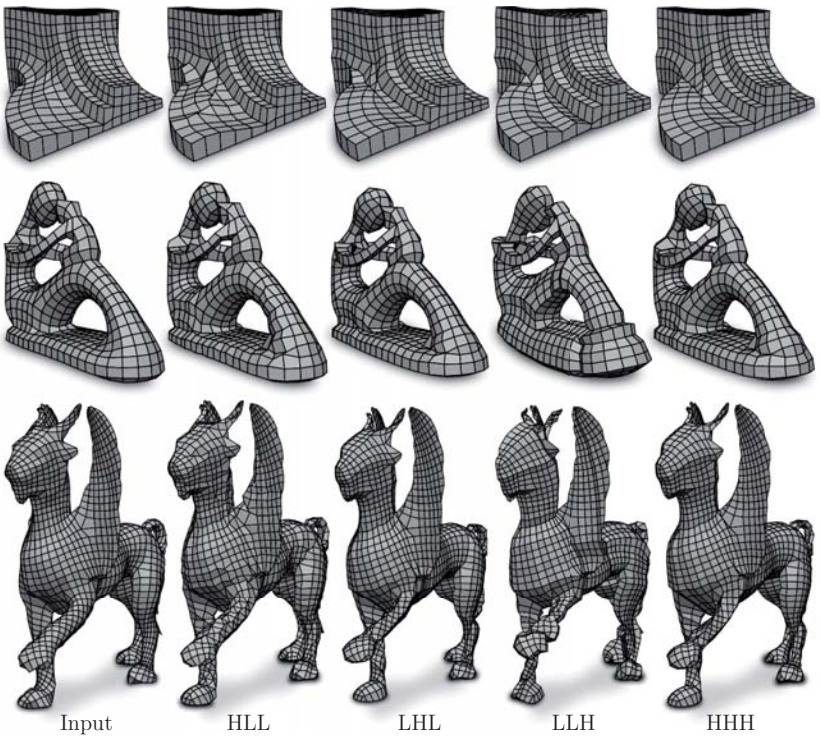


Figure 4.1.: Planarizations of three models (FANDISK, FERTILITY, FELINE) computed using different weights to define the planarization energy. The "L" and "H" combinations refer to the weights w_{pos}, w_{vec} and w_{len} being set either Low or High.

The most complex is the length based, quartic energy, which has a large impact on the performance.

Initialization

Different initializations can be shown to influence the convergence behavior of the optimization. The two different timings in the table (parenthesized and non-parenthesized) are the results of different normal vector initializations. The computation times naturally

Input			Timings (s)			
Model	n_F		HLL	LHL	LLH	HHH
FANDISK	764	$t_{\text{opt}} =$	9.0 (2.8)	5.0 (10.5)	11.0 (8.4)	10.0 (4.5)
		$t_{\text{fcn}} =$	0.0 (0.0)	3.5 (8.3)	31.0 (22.9)	21.0 (9.5)
FERTILITY	887	$t_{\text{opt}} =$	10.2 (10.1)	9.8 (11.2)	19.3 (17.2)	14.5 (14.2)
		$t_{\text{fcn}} =$	0.1 (0.1)	8.3 (9.3)	49.3 (47.2)	39.8 (38.2)
FELINE	3137	$t_{\text{opt}} =$	226 (209)	245 (311)	444 (496)	338 (415)
		$t_{\text{fcn}} =$	2.2 (2.0)	798 (1023)	5365 (7603)	3998 (4642)

Table 4.1.: Planarization timings for the quad meshes shown in Figure 4.1. t_{fcn} refers to the time spent evaluating functions (including Jacobians and Hessians) and t_{opt} is the optimization time (without function evaluations). The timings (seconds) in parentheses are for the initialization by "midpoint" normals based on the quads' midpoint-planes, the other timings correspond to the "standard" normals.

depend on the initial planarity of the faces. For non-planar polygons the normal vectors are not well-defined and typically an average over the polygon corner cross-products is used. The non-parenthesized timings in Table 4.1 refer to an initialization by such "standard", averaged normals, while the timings in parentheses refer to the so-called "midpoint" normals defined next. For quads, the midpoints of the quad edges can be shown to always define a plane. In the sense of *maximal projections* of polygons (cf. Lemma 2 in [AW11]) the normal of that plane can be thought of as a "natural" normal for the planarization of the corresponding quad, we refer to these as "midpoint" normals. Initializing all quad normals by the respective midpoint normals can lead to faster convergence of the optimization. The gain is the strongest for the HLL case (with up to 3 times faster convergence on the FANDISK). However, depending on the mesh geometry and the energy functionals also the opposite effect can arise. E.g., the misalignment between these virtual normals and the actual geometry of the (prescribed) edge vectors consistently worsens the timings in the LHL case. Note that the planarization is generally not unique and by using different initializations the optimization can converge to different minima.

Comparison

Our VTPI-based planarization [ZCHK13] using the functional defined by setting all weights high (HHH) was compared against two other state-of-the-art methods. For comparability these techniques also do not assume any control over the mesh generation process, but like our method, merely work with the given input mesh. The first method is the Planarizing Flow (PF) based on the Laplacian for general polygon meshes defined by Alexa and Wardetzky [AW11]. In the experiments reported here, the Laplacian was iteratively re-computed and applied to yield a planarized output (the smoothing component was switched off to avoid shrinkage). The second method is based on geometry projections using the Shape-Up (SU) framework by Bouaziz et al. [BDS*12]. SU allows for defining complicated, abstract functionals in a geometric manner, without giving a closed form. In the following experiments, the SU framework was used with the planarization operator described in [BDS*12] together with a BSP based projection to the input surface for closeness. Note that the results shown are of quad meshes optimized for planarity. Additional properties such as *conical vertices* have not been considered. For meshes having only valence three vertices, this property results automatically from the planarization. For other topologies it is possibly to add corresponding energy functionals (or projection operators) to VTPI and SU. It is, however, not straightforward to embed PF into an optimization framework due to the SVD involved in the computation of the Laplacian.

Shape Figure 4.2 shows the planarization results of the three methods on three input quad meshes computed by Mixed-Integer Quadrangulation (MIQ) [BZK09]. The colored areas in the input meshes highlight interesting regions handled differently by the methods. A strip of non-planar quads that undergoes a smooth rotation along its trajectory is marked in blue on the Fertility model. Such rotations tend to get concentrated at single points by our VTPI-based planarization, yielding triangle-like, degenerate quads. The PF better preserves the individual quad shapes, however, at the cost of introducing ripples and plateaus in the geometry. The results of SU are smoother than PF and more similar to VTPI. In orange, an area is marked where the input quads are not aligned to principal curvature directions. The necessary degeneracies again arise as ripples for the PF, while VTPI concentrates these effects, producing sharp lines separated by smooth areas. Without an additional rigidity term, the SU-based planarization can lead to noticeable loss of area and volume.

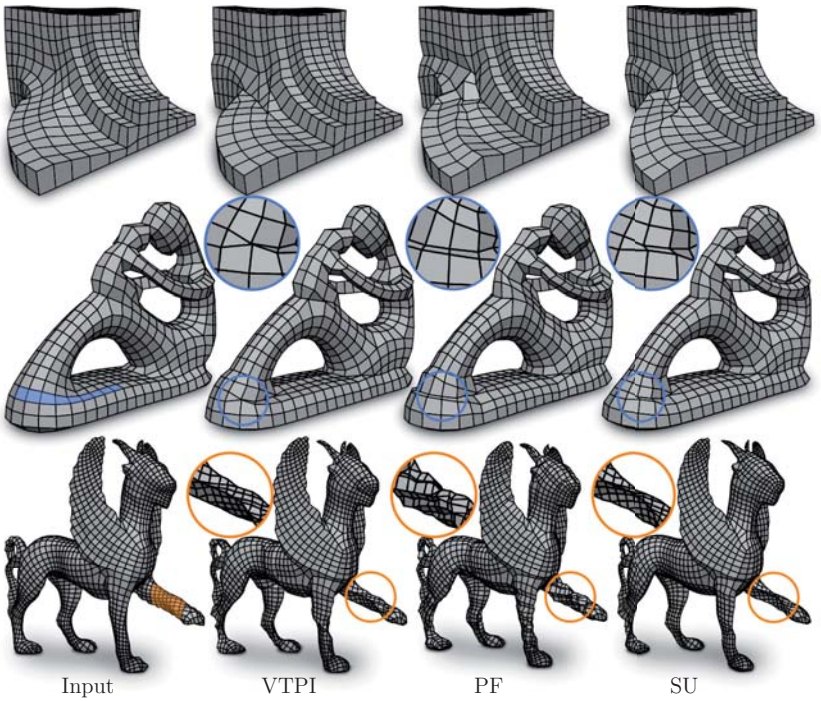


Figure 4.2.: Planarizations of three models (FANDISK, FERTILITY, FELINE) computed using three different methods: the VTPI-based planarization, the Planarizing Flow (PF) and a projection-based planarization using the Shape-Up framework (SU). Closeups are shown of some regions (highlighted) which are handled differently by the methods.

Timings and Planarity Table 4.2 lists resulting timings and planarity measures. While all timings (and planarity values) were measured on the same Intel i7-based PC, they can still only serve as coarse references, since their numbers heavily depend on the actual implementation, the chosen tolerances, the scale of the model and the parameters of the methods. To measure the planarity quality (or rather deviation from planarity) in quad meshes we use the $\delta_{PQ,n}$ measure from [ZSW10]. This measure is the maximum over all line-to-line distances of quad diagonals normalized by the mean diagonal length of the respective quad.

Input			Method			
Model	n_F		MIQ	VTPI	PF	SU
FANDISK	764	$t_{\text{tot}} =$	n/a	31s	15s	28s
		$\delta_{PQ,n} =$	0.21079	0.00000	0.00041	0.00043
FERTILITY	887	$t_{\text{tot}} =$	n/a	54s	20s	25s
		$\delta_{PQ,n} =$	0.53029	0.00000	0.00382	0.00023
FELINE	3137	$t_{\text{tot}} =$	n/a	4340s	66s	564s
		$\delta_{PQ,n} =$	0.72018	0.00000	0.01456	0.01409

Table 4.2.: Comparing the VTPI-based planarization using the HHH functional to the Planarizing Flow (PF) and the projection-based planarization in the Shape-Up framework (SU). In our experiments the VTPI planarity deviation was always less than 10^{-5}.

In regards to planarity, VTPI is fundamentally different from the other two methods. Here, planarity is not *minimized* as part of an energy functional, but implicitly guaranteed by fulfilling the constraints. This way the resulting degree of planarity does not depend on the final energy value as there is no race-condition with the other involved (weighted) functionals; if a solution is found, then the result is planar. In the presented examples, the VTPI planarity deviation was typically around 10^{-8} or less (depending on the tolerances used) and always less than the 10^{-5} accuracy used in Table 4.2. To obtain more planar meshes using SU or PF re-weighting and re-iterating is necessary, which would further contribute to the run-times of these methods. For the first two models the run-times of the VTPI-based planarization can be considered to be on par with the compared methods. Even for the more complex and less planar FELINE input mesh it is important to note that, while VTPI does display significantly higher timings, the results computed by PF and SU are still comparably far from being planar.

Other Techniques SU and PF are, similar to VTPI, not pure mesh planarization techniques but part of more general methods. The state-of-the-art in pure mesh planarization is often, similar to VTPI, based on numerical optimization. However, in contrast to the VTPI formulation over plane equations (leading to intuitive quadratic constraints for defining the planes), other methods are explicitly formulated over the mesh vertices and use, e.g., the distance between face diagonals [ZSW10], the sum of interior angles of quad faces (whose sum should be 2π) [LPW*06] or, for planarizing hex (or mixed-polygon)

meshes, the volumes of tetrahedra spanned by different combinations of four polygon corners [WLY*08]. Optimization based techniques are generally considered to have the advantage of *control*, i.e., by adding more functionals the behavior of the method and the result can be guided. However, this is also considered a weakness, since finding good weight combinations, especially in high dimensional spaces, is not trivial.

Outlook While the majority of planarization optimization techniques are formulated in a "primal" manner where vertex positions are relocated to achieve planarity, the "dual" formulation using planes not only has the advantage of implicitly guaranteed planarity, but the explicit representation of normals also allows formulating different types of energy functionals and constraints for more control. This is utilized for computing non-intersecting space frames consisting of two dual layers in Chapter 5. It is likely that also other applications can benefits from this formulation. However, relying on numerical optimization also poses limitations on the types of energies used, e.g., excluding energies which do not possess closed form expressions (and derivatives), such as a closeness energy based on point-projections to the input surface.

A possible direction for future research could be to investigate, to which extend the effect of VTPI to concentrate quad strip rotations and degeneracies to single points can be utilized for computing a planarization-optimized connectivity by locally remeshing around the degeneracies by introducing further irregular vertices.

Part II.

Dual-Layer Support Structures

This part concerns geometric optimization problems related to two types of similar architectural construction systems. Although quite different in the sense of the used "inner" structural elements, both structures deal with stress in related ways and both share a resemblance with space frame trusses. Furthermore, the systems display similar topologies, both consisting of a "primal" triangle mesh layer (the given input freeform design) and a second "dual" layer, connected to the first via the respective inner support structure. These similarities motivate collecting these types of structures under a common classification and we refer to them as *Dual-Layer Support Structures*.

Both research projects presented in this part were directly motivated by practical requirements from the field of architecture and structural design. The projects are a direct consequence of the fruitful interdisciplinary work in the *Fold-In* (`http://www.fold-in.rwth-aachen.de`) research conglomerate at the RWTH Aachen. In particular we would like to thank Karl-Heinz Brakhage, Martin Trautz and Ralf Herkrath for the inspiring discussions.

Chapter 5 is conceptually similar to the computation of (dual) polygon meshes with planar faces presented in the previous part with one major difference. It deals with a much more delicate setting, where not only a (single) polygonal surface is to be planarized independently, but as one part (layer) of a two layer structure. In detail, this chapter concerns the problem of inter-layer intersections. Geometric constraints are developed to allow for computing a planarized dual layer which does not intersect the given input mesh (primal layer). Furthermore, it is shown how the constraints can be described by simple polynomials and can, hence, be directly integrated into the VTPI optimization. The method is demonstrated by computing such so-called "Dual-Layer Space Frames with Planar Faces" on various architectural freeform designs.

The second type of construction system optimized in this part is referred to as "Point-Folded Structures". In principle, these are paneled space frames based on polygon meshes, where the panels do not only serve practical requirements (e.g., watertightness) but are also active supporting members of the system. The stability of the panels stems from their doubly-curved nature: Each panel resembles a pyramid with sharp creases meeting at an apex. With the creases resembling folds, the pyramidal elements are commonly referred to as *point-folded elements*. As is commonly the case for paneled structures, the diversity of the panels (here the pyramids) can be strongly linked to their fabrication costs. In Chapter 7 we define a similarity measure between pyramids and, based on this measure, develop an algorithm to minimize the diversity of base pyramid types. By careful design, the method can handle very high accuracy requirements and

readily incorporates various aesthetic and productional constraints. On a set of given architectural input designs rationalization gains of around and above 90% are shown to be realistic – meaning for computing point-folded elements on an input tri mesh with 100 unique triangles, where 100 unique pyramids are expected, only 10 different types of base pyramids were required, enabling significant reductions in fabrication costs in practice. The anti-diversification method is exemplarily based on a rather new production technique called *Incremental Sheet Forming* (Chapter 6), which has already been successfully employed for producing metal sheet pyramids for architectural applications.

5. Dual-Layer Space Frames with Planar Faces

This chapter concerns the geometric optimization and computation of architecturally inspired space frame constructions made up of two polygon mesh layers connected by an interior strut-based support structure for stability. Specifically two geometric properties of such structures are considered in the following:

- *Weather Resistance.* To enable insulation and watertightness, both layers shall admit efficient (planar) paneling.
- *Assemblability.* The layers must admit realization by an inner support structure and must not intersect each other.

In this chapter the input design (base layer) is taken to be a tri mesh and the second layer is a hex-dominant mesh with planar faces dual to the input, i.e., a PH mesh. The interior structure connects the two layers according to the duality between them, i.e., each primal node is connected by struts to the dual nodes corresponding to each of its adjacent faces. Figure 5.1 shows a mock-up rendering of such a structure with a glass-clad outer PH mesh (left) and a close-up of the interior support structure on the right.

For the base layer, the *weather resistance* aspect is covered by the inherent planarity of triangles. A second, weather resistant dual layer can easily be obtained by using the VTPI-based technique of Chapter 3, which dealt with the computation of dual meshes with planar faces for the purpose of (single-layer) architectural realization of freeform designs. However, the additional *assemblability* requirement poses non-trivial constraints on the layers to also guarantee separateness (a property generally not fulfilled by the simple dualization, even when prescribing preferred offset heights h_i), meaning the VTPI approach cannot directly re-used in a straight-forward fashion.

Finding a general solution to this problem is very hard. However, by restricting our attention to architecturally designed input meshes of reasonable quality (e.g., without

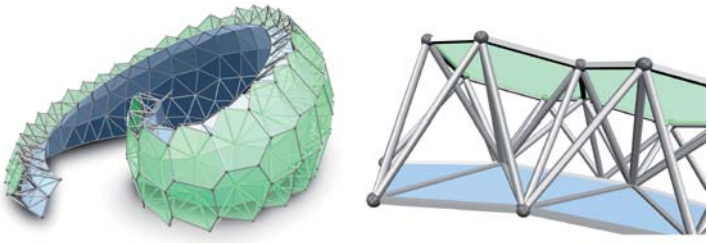


Figure 5.1.: Dual-layer support structure of the SeaShell model. The input design is shown in blue, the dual layer is transparent green. The model is courtesy of the Chair for Structures and Structural Design (`http://trako.arch.rwth-aachen.de`) at the RWTH Aachen.

extremely curved regions not allowing for meaningful offsets), a simplified but practically meaningful setting is obtained for which we devised a set of *local* geometric constraints to avoid inter-layer intersections (cf. Section 5.1). The constraints can be formulated as simple polynomials which can be directly integrated into the VTPI problem formulation and solved by the same optimization framework.

Energy Functionals Being based on the computation of dual meshes with planar faces described in Section 3.4 the same set of energy functionals are used here for computing the dual layer of the support structures: normal variation and fairing energies control the element quality and surface smoothness, while preferred positions (and also normals) are used to preserve geometric fidelity to the input. In this dual-layer setting, however, the offset-parameter h_i plays a more central part, as the thickness of the support structure can be shown to influence its structural abilities. Note that this work deals with aspects of geometric plausibility of the structure only, combining this with more detailed physical and structural aspects, ultimately necessary for real-life fabrication, is left as future work.

5.1. Inter-Layer Intersection Constraints

The constraints detailed below are based on the three intersection scenarios demonstrated in Figure 5.2.

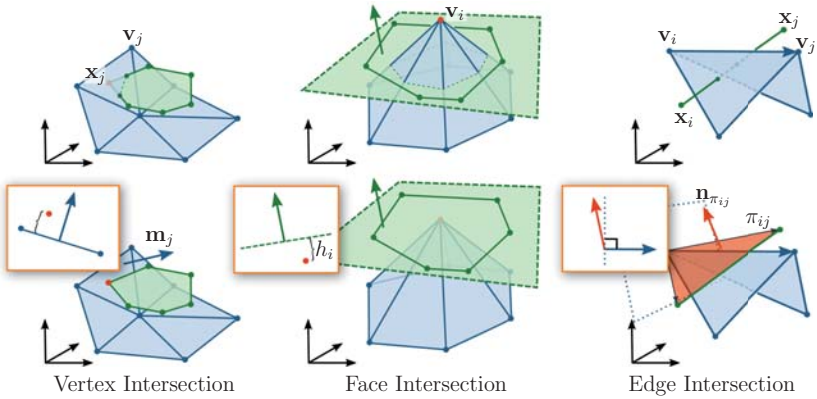


Figure 5.2.: Local intersection configurations for dual-layer space frames. The primal (tri) mesh is shown in blue and the dual PH mesh in green. The insets demonstrate the functionality of the constraints detailed in Section 5.1.

Vertex Intersection A *vertex intersection* occurs when a dual vertex lies behind the supporting plane of its corresponding primal (triangle) face (cf. Figure 5.2 (left)). Let $\mathbf{x}_j$ denote the VTPI intersection point (or dual vertex) corresponding to the jth triangle of the input mesh, further let $\mathbf{m}_j$ be the normal of that triangle and $\mathbf{v}_j$ one of the corners. To force the intersection points to lie in front of their respective primal planes the following inequality constraints, based on the plane equations of the primal faces, are introduced: $\{\mathbf{m}_j^\top(\mathbf{x}_j - \mathbf{v}_j) \geq 0\}_{j=1}^{n_F}$.

Face Intersection A *face intersection* means that a primal vertex lies in front (pokes through) a dual (planar) face (cf. Figure 5.2 (middle)). This is dual to the vertex intersection case. In the VTPI formulation a parameter already exist for influencing the preferred offsets of the dual planes from the primal vertices $\mathbf{v}_i$, i.e., h_i. By simple inequalities of the form $\{h_i \geq 0\}_{i=1}^{n_V}$, the dual planes are forced to be at a positive distance along normal direction from the corresponding primal vertices.

Edge Intersection An *edge intersection* refers to a configuration where a dual edge lies "below" its corresponding primal edge and intersects the two triangles (cf. Figure 5.2 (right)). However, the two edge vectors alone do not suffice to define a unique above/-

below relation. To formulate constraints for this intersection case we additionally rely on the orientation inherent in the input tri mesh. Let $\mathbf{v}_i$ and $\mathbf{v}_j$ be the two end points of a primal edge e_{ij} and let $\mathbf{x}_i$ and $\mathbf{x}_j$ be the end points of the corresponding dual edge e^*_{ij}. Now consider the (orange) plane π_{ij} with normal $\mathbf{n}_{\pi_{ij}}$ spanned by $\mathbf{v}_i$ and the dual edge e^*_{ij}. Intuitively, the boundary case where the two edges just touch can be described by the dot product between $\mathbf{n}_{\pi_{ij}}$ and the edge vector of e_{ij} being zero. If the dot product is always less than zero no intersection occurs[1]. The following inequality constraints are introduced: $\{((\mathbf{x}_i - \mathbf{v}_i) \times (\mathbf{x}_j - \mathbf{v}_i)) \cdot (\mathbf{v}_j - \mathbf{v}_i) \leq 0\}_{e_{ij} \in \mathcal{M}}$, where $\mathcal{M}$ denotes the input tri mesh. Naturally, this formulation depends on the relative positions of $\mathbf{x}_i$ and $\mathbf{x}_j$ and might not work as expected in the atypical configurations where both dual vertices lie on the same "side" of the primal edge or have even switched sides. If needed, additional constraints could be added, e.g., to enforce dual vertices to remain within the prisms of their extruded base triangles. However, the formulation used here proved sufficient in our experiments.

Note that being based only on local entities (normals and vertices) and the implicit assumption of only local intersections, the above constraints could be expressed in simple, closed polynomial forms. Preventing global intersections in a similar setting is in contrast a *very* hard problem where such simple formulations are not possible.

5.2. Evaluation

Dual-layer space frames with planar faces where computed for three different architectural freeform designs. Results are shown in Figure 5.1 and Figure 5.3. The blue mesh shows the input tri mesh and the green mesh is the (non-intersecting) dual mesh layer with planar faces. For demonstration the two are connected by a mock-up support structure consisting of sphere-like nodes and cylindrical steel struts.

Somewhat surprisingly, the timings are on average much lower than in Section 3.4, all here shown results were computed in less than 30s. The asymmetric AlpineHut with 468 faces required the maximum time and the symmetric 368 faced Trainstation requiring merely $4s$. One explanation for this effect is that, although the additional

[1] This constraint can equivalently be formulated as the sign of the oriented volume of the tetrahedron spanned by the four points.

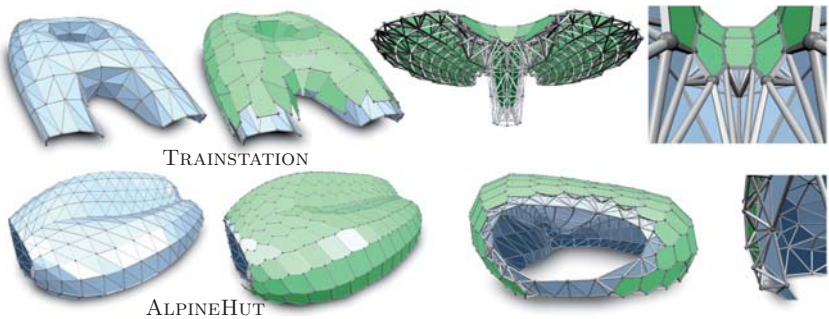


Figure 5.3.: *Dual-layer space frames with planar faces* computed for two architectural designs. The input tri meshes are colored blue and the dual-layers are green. The left two columns show the two layers and the right images show close-ups of a possible support structure. The Trainstation model was kindly provided by Evolute GmbH (`http://www.evolute.at`) and the AlpineHut is courtesy of the Chair for Structures and Structural Design (`http://trako.arch.rwth-aachen.de`) at the RWTH Aachen.

constraints may increase the complexity of the optimization problem, they also limit the search space and can speed up convergence.

On the front of the dual layer of Trainstation two valence 4 nodes can be seen. These are pairs of valence 3 nodes which naturally ended up at similar positions by the optimization. There are different options for handling this effect. The valence 4 nodes "proposed" by the optimization can be endorsed and made exact by updating the solution with the respective valence 3 vertices constrained to the same position. If a change in valence is not desired the energy functionals can be modified to pull the vertices further apart (e.g., by increasing the importance of the fairness energy) or edges can be flipped in the triangulation as proposed in [Tro08].

Limitations The layers of the computed structures are free of *inter*-layer intersections in the sense of the constraints defined above. However, prescribing unnatural heights in the vicinity of narrow concavities can lead to *intra*-layer intersections. An obvious example being the cylindrical hole in the middle of the Trainstation, prescribing offsets larger than the hole radius naturally leads to degeneracies. Typically, being of global nature, it is a very hard problem to efficiently safe-guard against such intersections.

In the shown results the inner support structure was exemplarily visualized as cylindrical struts connecting sphere shaped nodes. However, spatial dimensions of edges and vertices where not considered during optimization. Future work should include aspects of physical realization and the construction of non-zero thickness elements. Note that this is mainly an issue for the (non-conical) tri mesh, as these panels in general cannot be offset by a constant height to avoid intersections with the nodes and struts.

This section demonstrated another example of the versatility of the VTPI optimization problem. By integrating additional constraints, non-intersecting, planar offset surfaces could be computed.

6. Point-Folded Structures

Origami and folding-inspired techniques are becoming increasingly popular in architecture and structural engineering. While the stiffening effect of folds in thin sheets of a material, e.g., paper, is well known and "folds" have been used in construction and architecture since the early 19th and 20th century in form of corrugated metal and folded plate concrete, recent advances in materials research and geometric optimization has lead to an upswing in folding related research. For the purposes of this thesis it is helpful to differentiate between *traditional origami*, not allowing for cutting or gluing, and *generalized foldings*, which need not be foldable by the traditional rules but somehow utilize concepts of origami and folding for form finding or stability.

One important aspect of traditional origami are the kinematic degrees of freedom resulting from the actual (un-)folding process itself, with several works dedicated to the investigation and exploration of these degrees of freedom for deployable structures (e.g., [SDG10, Tac06, Tac10a, Tac09]). E.g., Schenk et al. [SDG10] analyzed the kinematics from a structural engineering point-of-view and T. Tachi [Tac10a] computed a retractable passageway based on a quad mesh folding pattern. While most works deal with such regular (polygon mesh-based) origami patterns, in [Tac10b] T. Tachi presented a method to compute folding patterns for more general tessellated freeform geometries and Kilian et al. investigated curved foldings in [KFC*08]. Venturing outside the scope of (assumed) zero-thickness materials (such as paper) additional care must be taken in the computation (and realization) of such designs. H. Buri et al. [BW10] rely on a similar origami pattern to the one in [Tac10a] in the form finding process of a temporary chapel, but realize the final design as a *rigid* structure made up of cross-laminated wood panels. This is an example of a generalized folding which is no longer foldable in the classical sense.

This thesis deals with a different type of generalized folding, where not the structure itself is the result of an origami-based form finding process, but rather consists of structural elements, which gain their stability from the stiffening effect of folds. In detail, the structures are made up of two polygon mesh-based layers connected by

pyramid-shaped structural elements. Naturally, since neither the whole structure nor the individual elements are foldable this is a kind of generalized folding. Recent publications by M. Trautz and co-authors refer to these elements as facet or point foldings (e.g., [TH09, HT11, TA11]). Structures made up of such elements were introduced in Chapter 1 and are referred to as *point-folded structures*. The shape of the structural elements (pyramids) depends on the type of polygon mesh used for the base layer, Figure 2.4 showed examples of structures with triangle, quad and hexagon-based pyramids. The above mentioned publications deal with generalizing the construction principle of point-folded structures, showed on regular geometries (domes and cylinder) in the figure, to freeform surfaces. A related interdisciplinary research project at the RWTH Aachen (`http://trako.arch.rwth-aachen.de/forschung/faltstrukturen-stahlblech/`) recently culminated in the prototypical realization of a freeform surface as a point-folded structure consisting of 140 all-different, hexagon-based, thin metal sheet pyramids. Figure 6.1 shows the finished structure and a close up of the hexagonal pyramids. Generally, freeform shapes do not exhibit the same regularity and inherent structural stability as geodesic domes and cylinders, making explicit control over the shapes of pyramids important to allow for aesthetic and structural adaption, e.g., by varying apex position and height (cf. [HT11, TA11]). The hexagon pyramids in the figure were generated by vertex offsets in normal direction, yielding all-different shapes.

Figure 6.1.: Left: Hexagon-based point-folded structure of the freeform TradeFair design. Right: Close-up of hexagon-based pyramids connected by triangle panels. Images courtesy of the Chair for Structures and Structural Design (`http://trako.arch.rwth-aachen.de`) and the Institute of Metal Forming (`http://www.ibf.rwth-aachen.de`).

Production

Metal pyramids such as the ones shown in Figure 2.4 could be produced by a series of cutting, folding and welding of metal sheets. Due to the regularity of the shapes (domes), only a small number of different such cutting/folding-patterns would be required. For injection molding the polycarbonate pyramids of the traveling pavilion even a single mold could suffice. The pyramids of the freeform surface shown in Figure 6.1 were produced by (single point) incremental sheet forming (ISF), a process where a blank sheet of metal is fixed in a holder and incrementally deformed by a CNC-controlled tool. ISF enables straightforward manufacturing of individually shaped elements, but, similar to the cutting/folding approach mentioned above, for each individual shape the machine must be re-set and re-rigged, making the reduction of element diversity an important cost factor [HT11]. Additionally, when high accuracy requirements necessitate the use of an individual *die* (a base-shape similar to a mold) per pyramid to prescribe accurate shapes, costs can quickly skyrocket.

The anti-diversification optimization presented in the next chapter is exemplarily based on an ISF production scenario where the number of different dies is to be optimized. However, it is clear that the problems of the different scenarios are related, e.g., the folding angles of a certain cutting/folding-pattern can also be found along the sides of the corresponding die. This implies that the advantages of such a minimal set of base-shapes can also imply advantages for other production scenarios. The next paragraph details some relevant aspects of ISF.

Incremental Sheet Forming In ISF (e.g., [JMH*05, HJW02]) metal sheets are shaped by a robot-arm, incrementally pressing down points of a blank sheet with a tool, either without a die, with a partial die, or with a full die (cf. Figure 6.2) – in the order of increasing accuracy of the resulting shape, but decreasing production flexibility. After the deformation is completed, the finished pyramid is cut out from the rest of the sheet. Being cost- and time-efficient for low volume production, the costs for ISF increase significantly for large volume production of unique elements, especially when accuracy demands a different die for each individual element. Optimizing the number of different parts can reduce this overhead drastically. In more detail, the next chapter considers the high accuracy setting, where each differently-shaped pyramid requires the manufacturing of an individual (full) die, and aims to minimize the number of such dies.

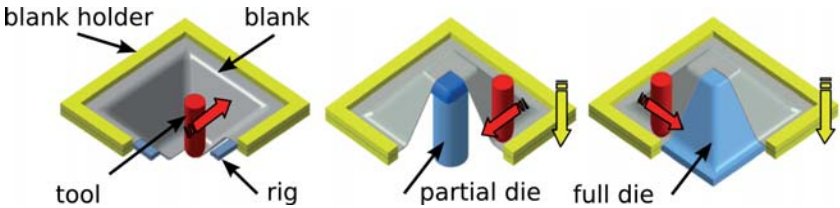


Figure 6.2.: ISF can be performed without a die, with a partial die and with a full die. Image courtesy of the Institute for Metal Forming (`http://www.ibf.rwth-aachen.de`).

The so called (maximum) *draw angle* is an important parameter, describing the material behavior as the maximum angle obtainable before material failure. The draw angle is measured between the production plane of the blank and the sides of the deformed metal sheet. Typically the maximally allowed draw angle is around 70 degree for various metals and steel [HJW02, JMH*05, Ame08]. Even before failure, significant thinning of the material can occur (related by the material thickness and the sine law) and, as other types of metal sheet forming, the process is influenced by spring-back, discouraging too low angles. For the architectural purpose investigated in this work, safely achievable angles are considered to lie in the range of $[20, 50]$ degrees. The anti-diversification approach presented in the next chapter is designed to respect such constraints and guarantee solutions within a prescribed angle range.

7. Anti-Diversification of Point-Folded Elements

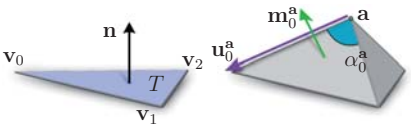
For rationalizing freeform point-folded structures the fabrication costs for the pyramids (point-folded elements) heavily depends on the element diversity (e.g., [HT11]). As discussed in the previous chapter there is great savings potential in reducing the number of "different" element shapes since it affects both (a) the times needed to re-setup the machinery for each separate shape and in a high accuracy setting (b) the expensive creation of an individual die for each different pyramid. Hence, for efficiently rationalizing freeform point-folded structures an anti-diversification of the elements is necessary.

In this chapter, based on [ZCBK12], we develop an anti-diversification method for reducing the die induced costs in a high accuracy ISF production scenario of triangle-based point-folded elements. For this, a measure of similarity between pyramids (or dies) is needed and, as already hinted in the previous chapter, one possibility for such a measure is a comparison based on angles, e.g., between the pyramid sides or creases. Unfortunately, this is not enough since pyramids can also have different sizes (heights). However, the ISF process inherently utilizes cutting at the end of the process to remove the deformed part of the sheet (the pyramid) from the rest. Hence, for the case of triangle-based pyramids, a die can be imagined as an very large trihedron (defined only by three angles) from which differently sized pyramids, having the same angles, can be obtained by different "cuts". This idea is similar to the anti-diversification method of Eigensatz et al. [EKS*10] working with various types of doubly-curved panels, where depending on the size and positioning of a blank on a larger mold, a differently shaped panel could be obtained.

7.1. Rationalization Scenario

The input to our anti-diversification method is a tri mesh $\mathcal{M}$ with n_F faces $\{T\}$ and n_V vertices $\{\mathbf{v}_i\}$, together with a set of requirements on the aesthetics and structural

properties of the resulting pyramids. These constraints are expressed over the allowed shapes of the resulting pyramids in the form of valid apex positions. The output of the anti-diversification is a minimal set of dies, which can be used to compute, for each triangle $T \in \mathcal{M}$ a pyramid which fulfills the given constraints.



Let the three associated corner vertices of a triangle $T \in \mathcal{M}$ be denoted $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ and the face normal $\mathbf{n}$. A pyramid basing on T is completely defined by the triangle corners and the pyramid apex $\mathbf{a}$. The input tri mesh is considered fixed and the pyramid entities are functions of the apex position alone. Let $\mathbf{u}_i^{\mathbf{a}} = \mathbf{v}_i - \mathbf{a}, i \in 0, 1, 2$ be the *crease vectors* (or "folds") which connect the apex to the corner vertices, let $\mathbf{m}_i^{\mathbf{a}}, i \in 0, 1, 2$ be the side normals of the pyramid and $\alpha_i^{\mathbf{a}}, i \in 0, 1, 2$ the angles between crease vectors $\mathbf{u}_i$ and $\mathbf{u}_{(i+1) \bmod 3}$. See the inset for an illustration.

Typical Requirements

Extensive discussions with architects and construction engineers[1] revealed a set of requirements that have to be fulfilled and properties that should be controllable in order to come up with desirable and realizable folded structures. These requirements can be due to structural as well as aesthetic considerations. In essence, given a (tessellated) free-form surface, design and rationalization tools for point-folded structures should be able to

- control **height** and **centricity** of each pyramid,
- respect **production constraints** for producible elements,
- prevent **collisions** between neighboring pyramids,
- **preserve** the given free-form shape (and even its often purposely crafted tessellation).

Centricity in this context refers to the relative position of the apex over some triangle center position, e.g., the barycenter or incenter.

[1] Chair for Structures and Structural Design (`http://trako.arch.rwth-aachen.de`) and Institute for Metal Forming (`http://www.ibf.rwth-aachen.de`) as part of the interdiciplinary Fold-In project at the RWTH Aachen (`http://fold-in.rwth-aachen.de`)

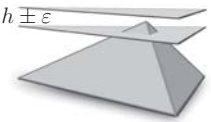
Validity Volumes Respecting the fourth point, i.e., considering the given tri mesh (and thus the pyramid bases) fixed, all pyramids are completely defined by the apex positions. Hence, handling all other conditions reduces to control over the apexes. In our approach we thus define *validity volumes* (VVs) for the apexes such that inlying (= *valid*) positions fulfill all requirements of a given scenario. It is convenient to think of these validity volumes in terms of indicator functions, which state whether an apex position $\mathbf{a}$ leads to a valid pyramid or not

$$\text{valid}^T(\mathbf{a}) = \begin{cases} 1 & \text{if } \mathbf{a} \rightsquigarrow \text{valid pyramid for T} \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

More precisely, a validity volume is defined as the kernel of the corresponding validity function:

$$VV^T := \text{kern valid}^T := \{\mathbf{a} \mid \text{valid}^T(\mathbf{a}) = 1\}, \tag{7.2}$$

i.e., the set of all valid apexes. The validity volumes are central to our approach. A simple validity volume which bounds the height and centricity of the pyramid is shown in the inset figure. Here the apex is restricted to lie in the space between two offset planes of the base triangle at offset distances $h-\varepsilon$ and $h+\varepsilon$ defined by a prescribed height h plus an allowed tolerance ε. To rule out pyramid elements with protruding apexes this volume is further clipped by the three planes spanned by the three edge vectors of T and its normal, yielding a prism. This is the basic setting used in the following – more complex constraints are considered later in Section 7.5.

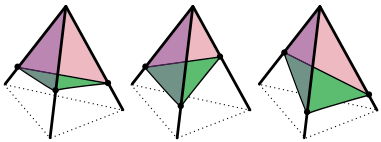


There is a connection between the height (and frequency) of folds and the structural properties of a folded structure (cf. [HT11, TA11]). In architectural applications desired heights h are typically in the range of $[10\%, 30\%]$ of the average edge length. The allowed tolerances depend on the specific application and on the material – hence, unless stated otherwise, in the following we rather aggressively assume allowed height deviations ε to be constrained to $\leq 2\%$ of the height, i.e., about 2-6mm in the case of 1m elements.

Rationalization of Triangle-Based Point-Folded Structures

For the rationalization problem at hand, similarity between pyramids (or dies) somehow needs to be quantified in order to compute common dies to enable efficient manufacturing of similar pyramids.

Pyramid Similarity To deduce such a measure it is useful to consider the definition of a pyramid given above from a reversed point-of-view: Instead of viewing a pyramid as being defined by the three fixed corner vertices of the base triangle together with the apex, the base triangle itself can be defined as the intersection between a trihedron (an infinite pyramid) situated at $\mathbf{a}$ and a plane with normal $\mathbf{n}$ passing through the positions $\mathbf{v}_i, i \in 0, 1, 2$. Now, as demonstrated in the inset figure, by using differently oriented "cutting" planes at different distances from the apex a whole class of triangular pyramids can be obtained. All such pyramids, cut from the same trihedron, are similar in the sense that they share the same triplet of angles at the apex. This is the key insight to efficient rationalization: if two triangles happen to have any two valid pyramids that can be cut from the same trihedron, one could simply produce a large enough representative pyramid of that class twice and cut it differently to cover both triangles.



Problem Statement Based on the above observation the rationalization problem can now be stated as:

> *Find a small set of trihedra such that for each triangle a valid pyramid can be cut from one of them.*

The smaller the set we find the higher the *rationalization gain*, meaning the percental gain compared to the trivial solution of using all unique dies. Let D be the set of all trihedra (or dies), then the rationalization gain can be defined as $\frac{n_F - |D|}{n_F}$.

Angle Parametrization of the Problem

We first note that the set of all trihedra is of dimension three. It can be parametrized by the three side facets' inner angles at the apex, i.e., a trihedron is uniquely defined by an angle triplet $\alpha := (\alpha_0, \alpha_1, \alpha_2) \in \mathbb{A}^3$, where $\mathbb{A}^3 := [0°, 180°]^3$. The angle triplet $\alpha^T(\mathbf{a})$ of the trihedron defined by an apex position $\mathbf{a}$ over a given base triangle T can be computed by

$$\alpha_i^T(\mathbf{a}) := \arccos\left(\frac{\mathbf{u}_i^{\mathbf{a}\intercal}\mathbf{u}_{(i+1) \bmod 3}^{\mathbf{a}}}{\|\mathbf{u}_i^{\mathbf{a}}\|\|\mathbf{u}_{(i+1) \bmod 3}^{\mathbf{a}}\|}\right),\ i \in \{0, 1, 2\}, \tag{7.3}$$

where $\mathbf{u}_i^{\mathbf{a}}$ are the crease vectors of the corresponding pyramid. This essentially allows us to map the validity volumes from $\mathbb{R}^3$ to $\mathbb{A}^3$ and we define the *angular validity volumes* $AVV^T := \alpha^T(VV^T)$.

Now, to determine if two given triangles T and t have any dies (trihedra) in common from which valid pyramids can be cut for both triangles, the following must hold:

$$\exists\, \mathbf{a}_T, \mathbf{a}_t : \text{valid}^T(\mathbf{a}_T) = \text{valid}^t(\mathbf{a}_t) = 1 \quad \text{and} \quad \alpha^T(\mathbf{a}_T) = \alpha^t(\mathbf{a}_t), \quad \text{i.e.,}$$

two *valid* apexes $\mathbf{a}_T$ and $\mathbf{a}_t$ must exist and both must yield the *same angle* triplet. This can be tested by mapping *all* valid apex positions of each triangle to $\mathbb{A}^3$ checking for overlap, i.e., $AVV^T \cap AVV^t \neq \emptyset$. In short, the search for trihedra, that can be cut to pyramids fitting multiple triangles while having valid apex positions, can be posed as an intersection problem on the AVVs. Naturally, the more AVVs of different triangles overlap the better the rationalization gain. Note that Equation 7.3 depends on the (random) ordering of the indices of vertices $\mathbf{v}_i$ of T, i.e., each apex position could actually be assigned three angle triplets. In practice, to be independent of this ordering and enable maximal rationalization gain, the union of three AVVs for each VV must be considered. Here, to simplify explanations and figures only one of these is considered.

Parachutes Figure 7.1 shows the kind of AVVs that arise when mapping the prismatic VVs, resulting from the basic height plus tolerance constraints, to $\mathbb{A}^3$. Due to their shape we refer to the AVVs as *parachutes*. As the height of the pyramid grows the apex angles decrease. Meaning, the apexes furthest away from the base triangle, i.e., on the "upper" offset plane, correspond to the smallest $\alpha \in \mathbb{A}^3$, i.e., lying on the "lower" side of the parachute. Of all positions in the prism, the top corners correspond to the smallest apex angles. This explains the shape shown in subfigure (b) where the three corners of the parachute seem to "point" towards a single position, the origin of $\mathbb{A}^3$. Subfigure (c) shows an arrangement of 11 such parachutes corresponding to different base triangles (with equal height constraints and large tolerances for visualization purposes). In practice, tolerances can be quite strict and parachutes correspondingly thin, posing high accuracy requirements on the intersection computation (cf. Section 7.2).

Optimal Rationalization Solution

In theory, the truly optimal rationalization solution respecting given constraints could now be obtained as follows:

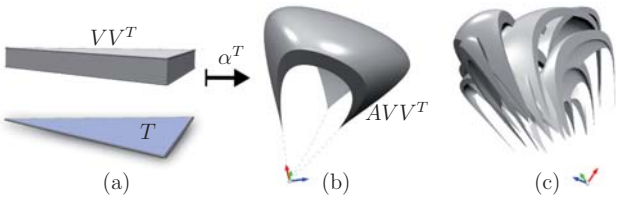


Figure 7.1.: (a) Validity volume (VV) defined over a triangle T by pyramid height constraint and tolerance. (b) the corresponding angular validity volume (AVV) (or *parachute*) mapped by Equation 7.3. (c) An exemplary intersection arrangement of AVVs in $\mathbb{A}^3$ resulting from a small mesh with 11 triangles.

1. Compute the intersection arrangement in $\mathbb{A}^3$ of the AVVs of all triangles $T \in \mathcal{M}$ and for each non-empty region $R \subset \mathbb{A}^3$ obtain the corresponding subset of triangles $S_R := \{T \mid AVV^T \cap R \neq \emptyset\}$.

2. Solve the set cover problem on the collection of all these subsets $\{S_R\}$ and pick an arbitrary representative angle triplet within each region corresponding to a subset of the result.

3. For each triangle map the corresponding representative angle triplet back to $\mathbb{R}^3$ to obtain a valid apex position within each VV.

The problems with this approach are that (1) computing the intersection arrangement of the AVVs can be considered computationally intractable – even for the simplest VVs the AVV boundaries cannot be described polynomially – and (2) the set cover optimization is known to be NP-hard and the number of subsets to be considered could be as large as 2^{n_F}, the power set of all triangles.

To remedy the first problem, we combine adaptive discretization and sampling techniques. This renders approximate determination of the intersection arrangement tractable. The algorithm is presented in detail in Section 7.2, where we also describe how we ensure that no intersections beyond a certain size are missed and at the same time no false positives violating any constraints are produced.

In order to solve the second problem, we use the greedy algorithm for set covering – a best-possible approximation algorithm for the set cover problem with polynomial time complexity [Fei98]. This algorithm chooses subsets in the order of decreasing cardinality until the whole universe (the set of all triangles T of $\mathcal{M}$) is covered. However, a naïve

implementation is effectively ruled out by the high time and space complexity induced by the accuracy requirements of the setting at hand (further detailed below). Our *memory efficient greedy set cover* algorithm is presented in Section 7.3.

How angle triplets are mapped back to $\mathbb{R}^3$ is explained in Section 7.4.

7.2. Efficient Intersection Arrangement Computation

We now proceed to (approximately) determine the intersection arrangement of the AVVs in $\mathbb{A}^3$. To achieve efficiency we discretize both the AVVs and $\mathbb{A}^3$ – the former using a conservative variant of prism refinement, the latter using an adaptive sampling of the space. However, care has to be taken when doing so, since (1) the risk of missing potential good solutions due to the discretization should be minimized and (2) the risk of producing invalid "solutions" should be zero – contradicting goals in the context of discretization with limited resolution.

The tolerance-induced thickness of the VVs naturally corresponds to the thickness of the AVVs and low tolerance (high accuracy) applications pose a significant technical challenge on the rationalization method. A simple computation shows that for a $1m$ long triangle with a $1cm$ thick VV, which is offset about $10cm$ from the base, even a sampling resolution of $0.5°$ in $\mathbb{A}^3$ might not be enough to distinguish between the upper and lower part of a parachute (cf. Figure 7.2). Meaning, a uniform sampling of $\mathbb{A}^3$ with a resolution of $180°/2^9 \approx 0.35°$ would barely be enough to place any samples on a single parachute. The problem is that the intersection regions between *several* parachutes are typically even much thinner. These, however, are exactly the sought regions, as they can maximize reusability. Hence, for optimal rationalization gain the angle space sampling resolution should be maximized. By using an on-demand geometry representation for memory efficiency and a multilevel discretization strategy our greedy set cover algorithm (cf. Section 7.3) is able to work with angle resolutions of up to 2^{15} samples per axis. Below the parachute discretization and sampling of $\mathbb{A}^3$ are detailed.

Discretizing Parachutes

Even the simple prismatic VVs introduced so far map to curved AVVs. We represent them using a piecewise linear boundary representation for efficiency. This can be done quite naturally to any desired accuracy by repeatedly performing 1-to-4 splits on the prismatic VVs in $\mathbb{R}^3$, then mapping the generated sub-prism vertices to $\mathbb{A}^3$. Unfortunately,

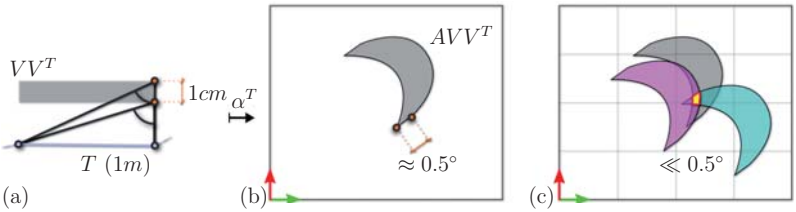


Figure 7.2.: (a) 2D view the $1m$ long side of a triangle T and its VV, defined by a $10\% = 10cm$ offset and $1\% = 1cm$ tolerance. (b) at the VV corners the (mapped) difference in $\mathbb{A}^3$ is about $0.5°$. (c) The regions overlapped by several parachutes are typically even smaller, requiring high accuracy intersection computations.

these representations are far from being conservative on coarse levels of the refinement – and we want to exploit also these coarse levels of the inherent multilevel hierarchy of prisms, as described in the following sections. To this end a *conservative* prism-based discretization was developed, which always contains the whole parachute and guarantees that no valid overlaps are missed on coarse levels. This is done by modifying each (sub-)prism to ensure that the corresponding parametric part of the continuous parachute volume is completely enclosed within. Figure 7.4 illustrates the difference between the naïve "lossy" representation (pink) and our conservative representation (blue), which is compatible with the (hierarchical) sampling of $\mathbb{A}^3$ described in the next subsection.

Conservative Prisms First of all, in order to obtain prisms with *planar* sides in $\mathbb{A}^3$ (to simplify intersection tests) we do not simply map the corresponding VV prism's vertices, but take as the AVV prism the volume enclosed by the five planes tangential to the AVV boundary at the images of the five sides' centers as depicted in Figure 7.3. The corresponding plane normals are computed by restricting Equation 7.3 to the (triangulated) sides $\triangle_{ABC} \subset \mathbb{R}^3$ of sub-prisms, yielding three restricted coordinate-maps $\mathbf{f} := (f_0, f_1, f_2)$:

$$f_i : \triangle_{ABC} \to \mathbb{A},\ (\lambda_0, \lambda_1, \lambda_2) \mapsto \alpha_i(\lambda_0 A + \lambda_1 B + \lambda_2 C)$$

with barycentric coordinates λ_j (with $\sum_j \lambda_j = 1$), from which the directional derivatives $D_{\mathbf{d}}(\mathbf{f})|_{(\lambda_0,\lambda_1,\lambda_2)}$ can be computed for directions $\mathbf{d} = (\delta_0, \delta_1, \delta_2)$ (with $\sum_j \delta_j = 0$) in $\triangle_{ABC}$. Computing the cross-product of the directional derivatives of two directions $\mathbf{d}_0$ and $\mathbf{d}_1$

oriented counter-clockwise in $\triangle_{ABC}$ yields the normal of that prism side at the point $(\lambda_0, \lambda_1, \lambda_2)$ mapped to $\mathbb{A}^3$.

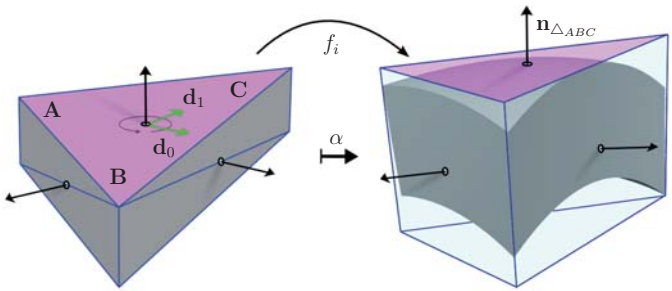


Figure 7.3.: For computing *conservative* AVV prisms with *planar* sides, the AVV prisms are defined as the volume enclosed by five planes in $\mathbb{A}^3$, which are positioned to contain the corresponding part of the (continuous) parachute.

For *conservativeness* we then shift these planes outwards such that the (corresponding part of the) actual AVV volume is contained in the (deformed) prism defined by these planes. Due to the absence of inflections on the AVV boundaries, the amount of shifting necessary could be determined using gradient ascents along the edges and within the face of each side. For simplicity, in our current implementation we shift to the maximum of several samples.

Tightness of Discretization The prism-representation must not only be conservative in order to not exclude good solutions, to avoid unnecessary overhead, it must also be *tight* enough not to generate too many false-positives (at least on the maximum refinement level, where the overlapping regions are extracted). As evident by Figure 7.4 neither representation (lossy nor conservative) is even visually accurate after 5 iterations of 1-4 split refinement. However, as a 1-4 split operation basically halves the edge lengths in each step, the quadratic approximation power (of piecewise linear representations, cf. [BKP*10]) leads to rapid quality improvement with each additional level. At level 6 the result is practically already visually indistinguishable from the continuous shape. Our experiments revealed that prism refinement above level 7 never improved the results, hence, unless stated otherwise, we use level 7 as the maximal refinement for an accurate and tight parachute discretization.

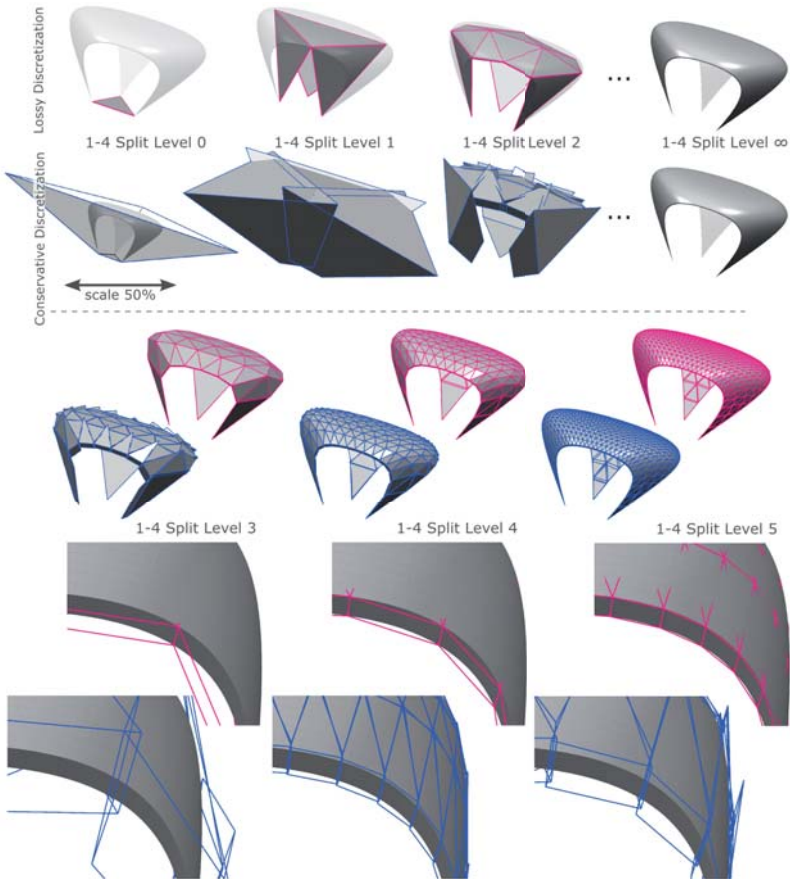


Figure 7.4.: The AVVs (or parachutes) are represented by a prism-based discretization. For computing exact intersections between parachutes in an hierarchical way, where information about possible intersections are propagated from coarse to finer levels, the simple "lossy" discretization shown in pink cannot be used. In blue our proposed "conservative" discretization is shown, which at all times encloses the whole parachute.

Hierarchical Sampling of $\mathbb{A}^3$

To identify overlapping parachute positions, we could now naïvely sample $\mathbb{A}^3$ regularly in $(2^{15})^3$ points, perform pairwise inclusion tests for these with the sub-prisms of all parachutes, and (in the manner of a greedy set cover algorithm) pick the one sample included in the largest number of parachutes, as it corresponds to the first best representative trihedron. This could be iterated for the remaining parachutes until all base triangles are "covered".

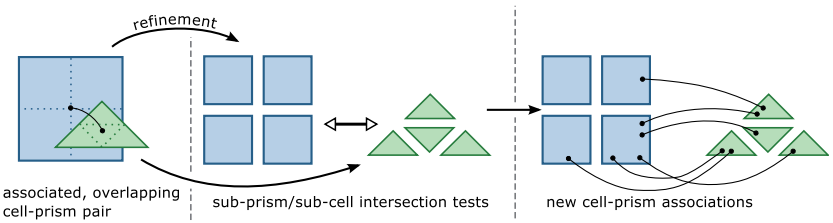


Figure 7.5.: Associations between intersecting prisms (representing parts of parachutes) and cells (representing parts of $\mathbb{A}^3$) are computed in a hierarchical manner: intersection computations only need to be performed between sub-prims and sub-cells which are children of prisms and cells already associated on the next coarser level.

It is immediately clear, that this is computationally utopistic. Our solution is to use an adaptive multilevel discretization of both the AVVs and the space $\mathbb{A}^3$ (as illustrated in Figure 7.5): We perform a simultaneous octree-based subdivision of $\mathbb{A}^3$ and a 1-4-split-based refinement of the AVVs in an interlocked manner. The idea is to propagate and refine intersection information from coarse to finer levels. Initially all level 0 conservative parachute prisms are associated with the octree root cell (encompassing all parachutes). Then, iterating over the subsequent levels, for each associated pair of octree cell and prism, the eight child cells and four child prisms are checked for intersections and associated accordingly. It is clear that a cell can intersect many prisms (and vice versa). By subdividing space adaptively in this way and performing the intersection tests in this doubly-hierarchical manner, spatial regions contained in many parachutes can be located with practicable performance. By the conservative discretization only false-positives disappear between levels, no valid intersections are lost. At the final level the cell centers are taken to be the sample positions (cf. Section 7.3).

As discussed in above the parachutes only require 6 to 7 levels to be accurate, whereas the angle accuracy required on the sampling should be much higher than 9 levels of "octree" refinement. Hence, after level 7, the above multilevel refinement only refines the octree cells further while leaving the prisms at their highest (level 7) resolution.

Efficient Prism-Cell Intersection Computation The octree cells in $\mathbb{A}^3$ naturally have 6 planar sides and by construction the conservative prisms consist of 5 planar sides. To establish an association between a cell and a prism in $\mathbb{A}^3$ the two are checked for intersections. Note that there is no need to compute the actual volumetric intersection arrangement between the two – association merely depends on the existance of a non-empty intersection. The intersection test can be reduced to a set of efficient tests based on the separating axis theorem (SAT) for convex polytopes [GLM96]. The idea is to project both objects onto certain axes and check if the resulting intervals are disjoint. The SAT states that only a finite number of such axes need to be checked. Similar to Bischoff et al. in [BPK05] we reduce the volumetric intersection tests to simpler tests between the octree cells and the (triangulated) sides of the prisms. For one triangle-box test 13 separating axes suffice [AM02]. Additionally, the trivial cases of complete containment of the prism in the cell (or vice versa) must be handled.

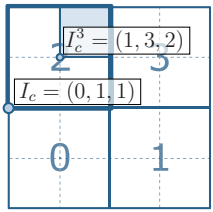
7.3. Memory Efficient Greedy Set Cover

The hierarchical sampling basically avoids pairwise comparisons between all parachute prisms on the finest refinement level. However, further measures need to be taken to reduce memory requirements to an acceptable level – the storage needed for (1) the geometry of prisms and cells as well as for (2) all cell-prism associations would by far exceed common main memory sizes at levels beyond 10.

We address the first issue (storage of prism and cell geometry) by simply encoding cells and prisms by an index (integer) from which the corresponding geometry can be created *on-demand* for the intersection tests and subsequently be discarded. The second issue (size of associations) is addressed in the following way: the refinement is performed in the hierarchical manner described above until the memory budget is nearly exhausted (we call this *phase I*), then we switch to an extremely memory-friendly depth-first search (called *phase II*) to be able to traverse the remaining sampling levels (usually up to level 15). Pseudo-code for the main functions of the algorithm is collected in Figure 7.7.

On-Demand Geometry

Creating octree cells on a regular grid from an index is straight forward. Let an octree cell c at refinement level l be indexed by the grid point (i, j, k) at its lower left corner and collect this information in one integer index $I_c = [i, j, k, l]$. Now the eight subcells on the next refinement level $l + 1$ are obtained by multiplying the grid position (i, j, k) by 2 and adding 1 where appropriate: $I_c^{0..7} = [i \cdot 2\,(+1), j \cdot 2\,(+1), k \cdot 2\,(+1), l + 1]$ as illustrated on a 2D example in the inset figure. The actual cell geometry (corner point positions) in $\mathbb{A}^3$ are obtained by an appropriate scaling.



Creating a sub-prism at a certain refinement level is slightly more involved and requires indexing functions for navigating through a triangle based refinement hierarchy. Our method is based on a simple row-by-row enumeration of prisms $p \in [0, 4^l]$ on level l (cf. Figure 7.6). Assuming that the geometry of the VVs on the most refined level is given, i.e., two tri meshes with 4^7 triangles each (for the top and bottom of the prism-shaped VV), using an appropriate mapping function, the geometry of a sub-prism on any intermediate level can be obtained by looking-up the corresponding corners points in the high resolution representation. The figure exemplarily demonstrates this for the blue prism $p = 3$ on level $l = 2$ (here the highest level is assumed to be 4 for visualization purposes). This way each vertex is stored only once (and not separately for all intermediate levels). Note that for simple constraints, such as the prismatic validity volumes defined by height+tolerance, no explicit geometry needs to be stored at at all, but the points can be obtained on-the-fly by simple barycentric combinations. However, an explicit representation at the highest level can make sense for validity functions defining more advanced constraints, where the VV shape is not trivial. By basing these volumes on deformed versions of the simple prism shaped VVs, the same indexing scheme can be re-used.

Finally, as generating the geometry of an octree cell is a trivial matter compared to constructing a conservative sub-prism (cf. Section 7.2), we here let the outer refinement loop run over the prisms and the inner loop over the associated cells. Meaning that the associations are actually directed and associate each prism with a set of cells (not the other way around). This avoids the need to create the same sub-prism geometry multiple times (cf. `phase_I` in the pseudo-code). However, for the `depthTraverse` in phase II the maximum refinement level of prisms is already reached and the oppositely directed associations are computed and used.

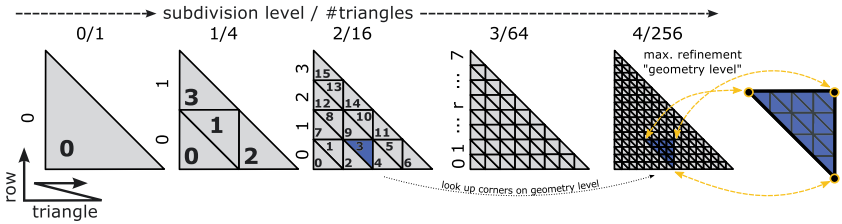


Figure 7.6.: An exemplary on-demand geometry hierarchy with 5 levels. The corner points (geometry) of the blue prism on level 2 are looked-up on the highest refinement level.

Two-Phase Processing

Phase I basically performs a breadth-first computation of all prism-cell associations. Note that the approximation accuracy of the prisms grows quadratically with the refinement level, in contrast to the linear increase in octree resolution. Thus, even if no new prisms are generated after level 7 (when the parachutes are considered accurate) the memory consumption for storing the prism- cell associations still grows rapidly as more and more cells are created. When the memory budget has been depleted the non-empty cells generally still need to be refined further to find accurate intersections. Since phase II typically sets in after level 7 (depending on input mesh and memory budget) we can efficiently rely on fixed prims for the remaining levels and only traverse the octree hierarchy further. This is done by, in a greedy manner in descending order w.r.t the number of associated parachutes, calling the depth-first search on each of the leaf cells. However, even on the finest refinement level, the association between cells and prisms only approximately and conservatively signifies an intersection in the corresponding region. To ensure that all constraints are respected, the center points of the cells on the final refinement level are mapped back to $\mathbb{R}^3$ (cf. Section 7.4) and then tested for validity to *exactly* determine how many input base triangles are validly covered by the corresponding trihedron.

Pseudo-code for `phase_II()` is given in Figure 7.7, where the depth-first search in a cell `C` is specified as `depthTraverse(C)`. Three global variables are used to keep track of the currently best found overlap in $\mathbb{A}^3$: `bestTrihedron`$\in \mathbb{A}^3$ is the best found triplet of apex angles and `bestCovers` is the set of base triangles for which pyramids can be generated using `bestTrihedron`. `bestBound` is the cardinality of `bestCovers`.

```
phase_I():
for each level l from 0 until memory full ...
  for each prism P of level l
    for each subprism p of P
      pg = on_demand_geometry(p)
      for each associated cell  C of P
        for each child cell c of C
          cg = on_demand_geometry(c)
          if intersects(pg,cg):
            associate(p,c)
```

```
phase_II():
  until all base triangles have pyramids do:
   sort leaves {C} desc. (wrt #parachutes(C))
   bestBound      = 0
   bestTrihedron  = none
   bestCovers     = empty set of triangles
   for each cell C of {C}
     depthTraverse(C)
   for each triangle t in set bestCovers
     a = reconstruct_apex(bestTrihedron,t)
     build_pyramid_for_triangle(a,t)
     delete all prisms P associated with t from {C}
```

```
depthTraverse(C):
if #parachutes(C) ⩽ bestBound: return
if finalLevel(C): updateBound(C)
for each child c of C
  cg = on_demand_geometry(c)
  for each associated prism P of C
    pg = fixed_geometry(P)
    if intersects(pg,cg):
      associate(P,c)
sort cells {c} desc. (wrt #parachutes(c))
for each cell c of {c}
  depthTraverse(c)
  delete c and its associations
```

```
updateBound(C):
{T} = trianglesValidlyCovered(C)
if |{T}| > bestBound
  bestBound     = |{T}|
  bestTrihedron = centerOfGravity(C)
  bestCovers    = {T}
```

```
trianglesValidlyCovered(C):
{T} = empty set of triangles
trihedron = centerOfGravity(C)
for each associated prism P of C
  t = base_triangle(P)
  a = reconstruct_apex(trihedron,t)
  if valid^t(a)
    add triangle t to set {T}
return {T}
```

Figure 7.7.: Pseudo-code of important parts of the algorithm: `phase_I()` constructs an initial octree. In `phase_II()` for each of its leaf cells `C` (in descending order w.r.t the number of associated parachutes) `depthTraverse(C)` is invoked. Afterwards the best angle triplet found (`bestTrihedron`) is used to construct pyramids for the covered triangles, their associated parachutes are removed from the cells and the process is repeated until all base triangles are covered by a pyramid.

Branch-and-Bound The search efficiency can drastically be improved in a branch-and-bound manner, i.e., if the best point (angle triplet) found so far is included in k parachutes, all sub-trees of cells associated with prisms of no more than k parachutes can safely be skipped subsequently (in the pseudo-code this current bound k is kept track of in the global variable `bestBound`). This is due to the fact, that the number of associated parachutes (`#parachutes(C)`) provides an upper bound on the number of parachutes that might overlap any common point contained in the cell. At the end of the traversal, the point that established the last such bound (i.e., `bestTrihedron`) signifies

the trihedron that is valid for the largest number of base triangles. Following the greedy approach to set cover, the parachute prisms corresponding to these base triangles are then removed from the octree leaves that resulted from phase I, and phase II is repeated to find the next best trihedra until all base triangles are covered.

7.4. Folded-Element Construction

Having found a representative angle triplet in $\mathbb{A}^3$, we map it back to $\mathbb{R}^3$ to determine the apex positions and construct the pyramids for the corresponding base triangles. The map in Equation 7.3 is non-injective and a global closed-form inverse is not available, but the construction of the pyramid on a triangle T given an apex angle triplet $(\alpha_0, \alpha_1, \alpha_2)$ is equivalent to the perspective three-point pose (P3P) problem well-known in Computer Vision. In [FB81] Fischler and Bolles describe how the position of the camera (here: the apex $\mathbf{a}$) is reconstructed from three sighted points (here: the vertices $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$) respecting the angles between the corresponding sight rays by finding the real roots of a quartic polynomial. Some care must be taken as the polynomial possesses up to four real roots and unsound solutions have to be filtered out. In our case validity is defined by the indicator function (cf. Section 7.1) and can directly be checked by evaluating $\text{valid}^T(\mathbf{a})$.

7.5. Advanced Constraints Handling

Having described our general rationalization pipeline, we now take a closer look at how further constraints can be incorporated. So far, production and construction constraints have not been taken care of. Also, explicit control over the positioning of the apex and shape of the elements can be desirable. This calls for differently defined (shaped) validity indicator functions (volumes). However, the rationalization relies on adaptive refinement of on-demand prism geometry based on the regularly refinable triangular structure of the VVs. Hence, to gain further control over the rationalization process, we only modify the prismatic VVs to suit our needs, the rest of the pipeline remains unchanged. There are basically two ways to modify the prismatic VVs: (1) for simple, small volumes, sub-prisms falling outside the volume can just be deactivated and (2) for more complex cases the VVs can also be deformed or projected to the desired shape.

Collision Prevention

In non-convex regions of the base mesh, the simple VVs (cf. Section 7.1) of neighboring triangles might be non-disjoint, potentially leading to solutions with intersecting pyramids that could not be assembled physically. Unless unusually high pyramids shall be placed in extremely curved concave regions or narrow passages, we may safely assume that intersections might only happen between pyramids on faces sharing a common edge or vertex. However, since tri meshes generally are not *conical* there are no unique vertex axes (or edge planes) that can be used to offset the faces in way that they are not intersecting. Hence, we apply a more brute-force approach and, after offsetting the triangles in normal direction, clip the sides of the offset triangles not to intersect their neighbors. By, for each edge of a triangle, using a clipping plane going through the edge and containing the edge normal (averaged incident face normals), intersections between the VVs of the adjacent (offset) triangles can be prevented. However, intersections across vertices potentially remain. These are additionally ruled out by, for each edge of a triangle, instead taking the *innermost* plane of the base-orthogonal clipping plane and the two that contain either incident vertex's normal. Here *innermost* means the plane whose outwards normal has the largest dot product with the base triangle normal, i.e., leans the most in over the base triangle. We refer to these clipped prisms as *collision prevention* constraints (CP). Note that, as also discussed in previous chapters, efficiently preventing intersections at a global level is outside the scope of this work.

Centricity Control

The prismatic VVs can be modified further to gain control over the positioning of the resulting apexes. We here consider the case that apexes shall be restricted to lie no further than some distance r from the base-orthogonal line through some center of the base triangle, e.g., the in-center. Obviously, the corresponding VV is a cylinder.

Since usually collision constraints (CP) are to be considered additionally, we are interested in the intersection of this cylinder with a clipped prism obtained as above. Hence, we keep the prismatic VV with its refinement structure and simply mark those sub-prisms as *inactive* that lie outside the cylinder. These are then ignored in the intersection computation/refinement process. Prisms partially on the outside are kept active to not miss any solutions – the final validity check (cf. Section 7.4) rules out false positives. We refer to these constraints as *centricity control* (CC).

Production Constraints

Depending on the folded element manufacturing method different constraints on the attainable element shapes might be given. Here we exemplarily consider the ISF production method to illustrate how such constraints can be incorporated. The previous chapter mentions the problem of excessive material thinning and the possible spring-back related to high and low *draw angles* respectively. Assuming a given range $[min_{ISF}, max_{ISF}]$ of safely achievable draw angles the VVs must be modified correspondingly to guarantee that the point-folded elements resulting from the rationalization can be produced. Note that one cannot just measure the angles to the supporting plane of a triangle T for a given apex $\mathbf{a}$, as this plane is generally not the actual production plane of the blank sheet, due to production of multiple, differently cut pyramids from one representative master – we must base our considerations on a virtual production plane.

We consider the optimal production plane for a given trihedron to be the one having equal angles to all trihedron sides – it simultaneously minimizes the maximum angle and maximizes the minimum angle, resulting in best-possible production quality. The normal $\mathbf{n}$ of this virtual production plane for an apex position $\mathbf{a}$ can be found as the vector $\mathbf{a} - \mathbf{x}$ for the center $\mathbf{x}$ of an insphere of arbitrary radius r of the trihedron. Such a point $\mathbf{x}$ is found by solving $\mathbf{m}_i^{\mathbf{a}\mathsf{T}}\mathbf{x} = r - \mathbf{m}_i^{\mathbf{a}\mathsf{T}}\mathbf{a}, \quad i \in \{0, 1, 2\}$. Figure 7.8 (a,b) show the resulting boundaries of the volumes defined by apexes fulfilling the max_{ISF} and min_{ISF} angles respectively. Now that we have a point-wise test for ISF compatibility, combining the ISF constraint VVs with other VVs can be performed by deactivating sub-prisms whose vertices lie in invalid regions as described in the last section. Subfigure (d) shows the resulting volume of ISF-manufacturable apex positions over a triangle additionally respecting the height and local intersection constraints. Note that outside vertices of partly active prisms can furthermore be projected to the valid region in order to reduce the number of false positives.

Combining Constraints

Each additional constraint decreases the size of the solution space, in fact, a combination of constraints does necessarily even result in non-empty VVs. E.g., depending on the valid range of ISF angles and the prescribed height, the volumes in Figure 7.8 might not overlap. Being based on hard constraints in form of VVs, empty VVs in the rationalization effectively mean that covering the corresponding base triangles will require customized solutions. In certain cases, however, one constraint can be considered

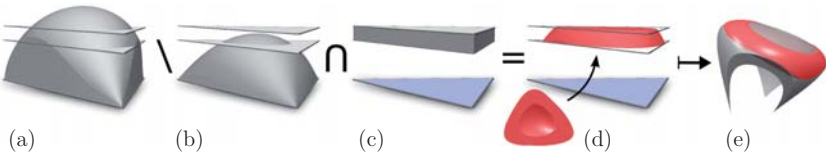


Figure 7.8.: (a,b) Visualization of the surfaces implicitly defined by a min_{ISF} and max_{ISF} constraint. (c) The basic height constraint VV. (d) The resulting VV respecting ISF and height constraints. The inset shows a bottom view – the "carved out" region contained apex positions that would lead to base angles $< min_{ISF}$. (e) AVV of a standard height constraint (grey) with the sub-volume that remains when additionally considering the ISF constraints highlighted in red.

"softer" than the other, and VV modifications to readmit a solution might be acceptable. E.g., in the height+ISF case above, we consider the production constraints to be harder than the height constraint. This allows us to compute a smooth height field over the input mesh triangles, where the prescribed heights are preserved as well as possible, while being restricted to lie within the range of the ISF constraints. Now, if the so computed heights still fulfill the aesthetic and structural requirements posed, the rationalization can be carried out in a straightforward manner, if not, a customized solution is required.

Figure 7.9.: The four base meshes used in the experiments.

7.6. Evaluation

We evaluate the proposed rationalization method on four architect designed models shown in Figure 7.9. SeaShell ($n_F = 249$), TradeFair ($n_F = 270$) and AlpineHut ($n_F = 468$) were kindly provided by the Chair for Structures and Structural Design

(`http://trako.arch.rwth-aachen.de`) and TrainStation ($n_F = 1038$) is courtesy of Evolute GmbH (`http://www.evolute.at`). All examples have been processed on a modern standard PC (Intel i7-920 CPU).

Table 7.1 shows the rationalization results for these models and compares different constraint configurations. Except in the ISF cases the apex height was fixed to $0.2m$ (ca. 20% of the average edge length) and $\varepsilon = 2\%$ was used. For the ISF cases, smoothly varying heights from a range around 20% were automatically set in a way to adapt to the ISF constraints and guarantee non-empty VVs. For comparability, the switch from phase I to phase II of the algorithm has been fixed to after level 7. The specified rationalization gain is defined as the percentage of the number of folded elements of the trivial solution (a unique element per base triangle) made obsolete by rationalization. Figure 7.10 illustrates the results.

	SeaShell, $n_F = 249$			TradeFair, $n_F = 270$		
	CP	CP+ISF	CP+CC	CP	CP+ISF	CP+CC
Unique Dies	13	9	32	10	11	21
Rationalization Gain	95%	96%	87%	96%	96%	92%
Runtime Phase I (min)	9.1	8.6	8.2	8.9	9.1	8.2
Runtime Phase II (min)	11.0	5.0	2.3	7.5	8.8	10.0
Peak Memory (GB)	3.2	2.9	2.5	3.1	3.4	2.1

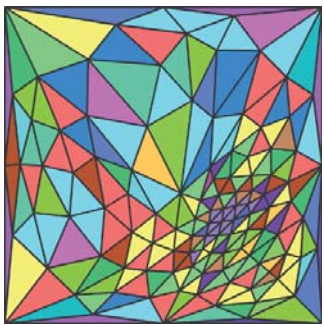
	AlpineHut, $n_F = 468$			TrainStation, $n_F = 1038$	
	CP	CP+ISF	CP+CC	CP+ISF	(level 6)	(level 7)
Unique Dies	22	16	50		11	11
Rationalization Gain	95%	97%	89%		99%	99%
Runtime Phase I (min)	15.0	14.8	13.5		16	83
Runtime Phase II (min)	39.0	51.9	14.8		440	337
Peak Memory (GB)	4.9	5.1	4.1		4.1	11.5

Table 7.1.: Statistics of the folded element rationalization for our examples. Columns show results for processing with collision prevention constraints only (CP), additional ISF production constraints (ISF), as well as additional centricity control (CC).

Achieved rationalization gains range from 87% to 97% for the first three models, whose tessellations have been provided by architects. For the TrainStation model, which has

been quite uniformly meshed based on [BK04], even 99% were achieved. As expected, tighter constraints usually lead to lower gains since AVV intersections tend to be rarer for smaller VVs. The exceptions seen in the table when comparing collision prevention constraints only with additional ISF constraints are due to the variable height constraints applied in the ISF case, hence incidental. The last two columns exemplarily illustrate the effect of the transition from phase I to phase II of the algorithm: switching to phase II earlier (here at level 6 of the octree refinement) leads to lower memory consumption but higher runtime compared to switching later (here at level 7).

Architecturally crafted tessellations, like the ones used in our experiments, usually have "nice" elements, e.g., round, similar triangles. Still, it is interesting to see to which extent the rationalization depends on this circumstance: we exemplarily applied our method to the rather irregular mesh depicted on the right. As could be expected, the gain was lower, but still 88% were achieved (using CP).



With our current research implementation, computing a full resolution rationalization on a standard PC is limited to scenarios with less than about 2000 folded elements. Optimization of the employed data structures and redundancy reduction will likely be able to further raise these bounds.

On the production side of things current research topics include the exploration of novel uses of point-folded structures and the development of efficient folding element production techniques. The here presented method makes a big step towards usability of metal sheet pyramids in large scale, free-form production scenarios – as the cost of producing the molds, dies, or tools for element production can be drastically reduced, depending on the production scenario. However, further aspects in this context still remain to be explored, as outlined in the following.

Soft Constraints The presented anti-diversification technique is completely "discrete" – there are no soft-constraints or "more or less preferred" solutions; always *some* valid solution (possibly out of several similarly good alternatives) is found. When soft constraints are desired, e.g., for mixing aesthetic preferences with hard production constraints, one could switch to using VVs augmented by weighting fields. While defining such weighted VVs is straightforward, some work would have to be done to steer the

adaptive discretization accordingly, e.g., considering (approximate) weighted integrals to choose the next cell for refinement.

Tessellation The TrainStation example demonstrates that uniformly meshed input meshes facilitate high rationalization gains. We considered the case of purposely designed input meshes that shall not be altered. An interesting direction for future work is the exploration of a combined rationalization and modification of the base mesh. Naturally, similar triangles can potentially improve the rationalization gain, and additionally optimizing the geometry of the base mesh in such a way (e.g., [SS10]), possibly also allowing for topological modifications (e.g., [LZKW10]), could be explored to further improve rationalization.

Dual Surface A deeper analysis of the properties and aesthetics of the implicitly generated dual surface spanned by the apexes could be useful. For instance, structural properties were not explicitly considered but only assumed implicitly defined by appropriate heights and apex positions. Also the aesthetics of the dual surface are of importance, one could think of defining a smoothed offset band over the base surface and constrain the apexes accordingly. This has already rudimentarily been explored in our experiments for adjusting the heights to meet ISF production constraints. Furthermore, additionally enforcing a (planar) panelization of the dual surface, i.e., combining the anti-diversification idea of this chapter with the dual-layer space frames of Chapter 5, yields a very hard problem, which would require a combined continuous and discrete optimization possibly along the lines of [EKS*10].

Polygonal Bases In principle, the greedy rationalization part of our algorithm could also be extended to point-folded structures based on, e.g., quad, hex or mixed meshes. However, as noted in [FB81], challenges could be posed regarding the inverse mapping due to non-planarity or non-convexity of the base polygons. Additionally, the (curse of) dimensionality of a 4 or 6 dimensional search space would dramatically increase memory consumption, yielding the here obtainable resolutions of 2^{15} samples per axis in $\mathbb{A}^3$ impossible, possibly calling for a different approach all together.

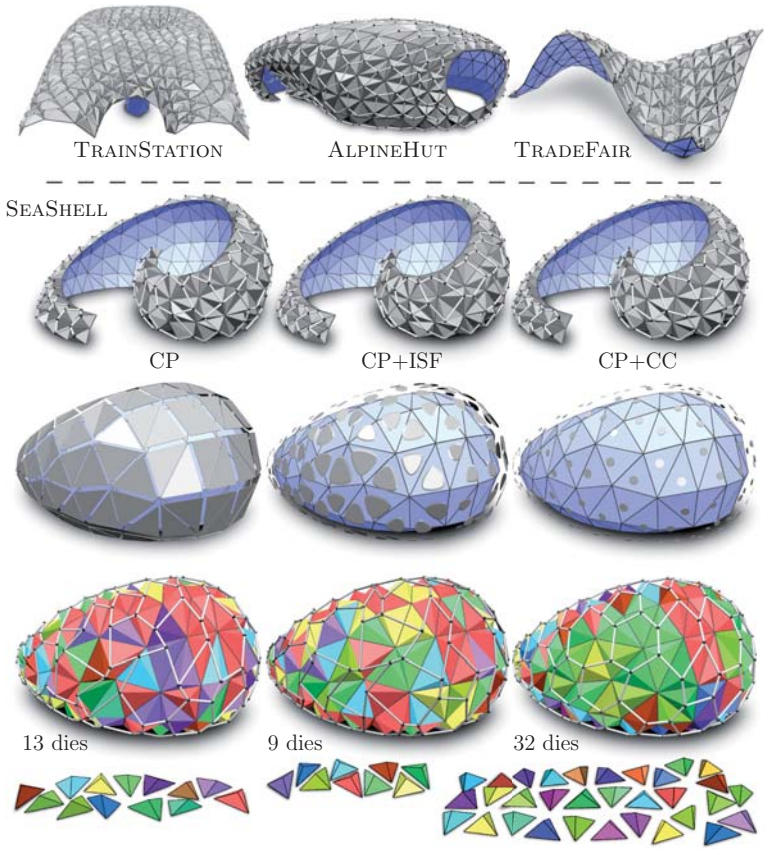


Figure 7.10.: The top row shows resulting point-folded structures of the TrainStation, AlpineHut and TradeFair rationalized by our approach. For the SeaShell three different cases are detailed: From left to right, the columns depict the CP, CP+ISF and CP+CC constraints respectively. The resulting point-folded structures are shown in the first row, with the two following rows depicting the used VVs and a color-coded visualization of the rationalization. It is noticeable how the dual support mesh becomes increasingly regular as the VVs shrink and get more and more centralized.

Part III.

Freeform Zometool Structures

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. This section presents two methods enabling physical realization of freeform designs without the typical limitation of state-of-the-art rationalization approach requiring manufacturing of custom parts.

Both approaches are based on a popular, tangible construction system: Zometool. The properties of the system make it inherently well suited for modeling symmetric structures such as molecules and crystal lattices. However, it does not only find use in various branches of science for research and teaching but also recreationally for personal fabrication. While being limited in the sense of having only a small, discrete set of available angles and edges the Zometool system in fact allows for a very rich set of structures, as will be demonstrated by the two approaches presented in this part. For digitally modeling and designing Zometool structures there are two software systems available for point-and-click-based Zometool modeling. However, prior to the work presented in this part, there existed no algorithmic approaches for assisting in the Zometool realization of *freeform* designs. A *free-styling* approach to such constructions can quickly be hampered by the fact that, when venturing outside the known symmetries of the system, one easily encounters situations where there are suddenly no appropriate slots or struts available to form a certain desired connection.

Chapter 8 more formally introduces the Zometool system and the approximation problem to be solved. In Chapter 9 an efficient algorithm to support efficient approximation and creation of freeform Zometool structures of arbitrary genus is presented. Here, the Zometool-model-space around a given input design is explored in an efficient manner to find a fitting approximation. Chapter 10 then considers the more restrictive setting of *panel-aware* Zometool rationalization, where the resulting panels shall be guaranteed planar and convex to enable efficient fabrication.

8. Zometool Shape Approximation

The Zometool system is a node and strut-based construction set, consisting of a single type of node having 62 different holes (called *slots*) and 3 different edge types (called *struts*), with each strut coming in 3 different lengths. The system inherent symmetries make it popular for hands-on visualization of, also higher dimensional, geometric structures and symmetries in different branches of science and in teaching. It is also used recreationally and a variety of astonishing, large-scale structures are designed and built by enthusiasts. Some complex, real-life examples are shown in Figure 8.1. In their book [HP00], Hart and Picciotto present a good introduction to Zometool and more examples of various structures can be found on the book's accompanying website (`www.georgehart.com/zomebook/zomebook.html`) and on the manufacturer's website (`www.zometool.com`).

Digital design and modeling of Zometool structures is supported by two available software system [Sch, Vor], both allowing for simple point-and-click adding of new nodes and struts, but also adding of pre-defined polyhedra and exploration of, and auto-completion based on, the system symmetries. However, even despite its intriguing mathematics and the potential advantages of a construction system with a fixed set of elements, e.g., for architectural applications, there has to date been no algorithmic approaches dealing with freeform Zometool realizations. From an optimization point-of-view the combinatorics and the inherent discreteness of the system rule out the use of efficient numerical solvers and pose difficult constraints on such endeavors. Hence, although mathematically well-founded and beautiful, typical recreationally built Zometool structures are still often (a) geometrically and topologically simple or (b) highly symmetric and regular. This chapter lays the foundation for the two freeform approximation approaches presented in the next two chapters. The Zometool fundamentals are presented in Section 8.1 and Section 8.2 poses the general form of the corresponding freeform approximation problem to be solved.

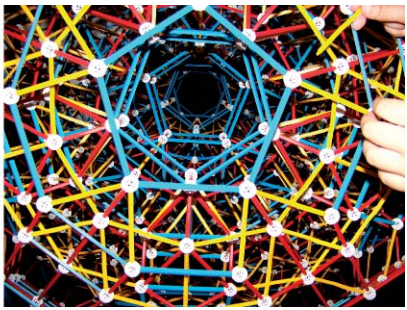
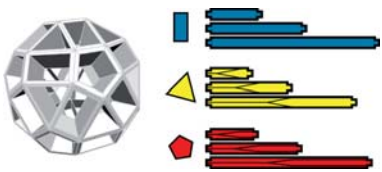


Figure 8.1.: Complex, real-life Zometool structures. The structure in (a) was designed by Chris Kling and the image is courtesy of George Hart (`http://www.georgehart.com`). The image in (b) is courtesy of Tanya Khovanova (`http://blog.tanyakhovanova.com`).

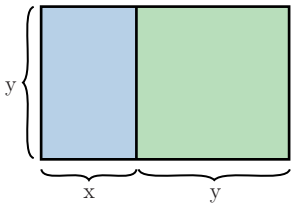
8.1. The Zometool System



The three different strut types of the standard Zometool system are colored blue, yellow and red. Each strut comes in three lengths. Let b_0, b_1, b_2 to refer to the lengths of the three different blue struts and analogously y_0, y_1, y_2 and r_0, r_1, r_2 for the yellow and red struts. The geometry of the Zome node corresponds to that of a slightly modified rhombicosidodecahedron, with the square faces expanded to golden rectangles. Each of the 62 slots is restricted to a single type of strut: there are 12 pentagonal slots for red, 20 triangular slots for yellow and 30 rectangular slots for blue struts.

Besides the comprehensive *Zome Geometry* book [HP00] by Hart and Picciotto, other documents dealing with and detailing various parts of the Zometool system exist. E.g., the official Zometool manual [Zom13] provides an introduction to the system with getting-started examples and *The Mathematics of Zome* by Tom Davis [Dav07] mathematically derives the lengths of and relationships between the different struts. Below we summarize facts useful for the algorithms presented in the next two chapters.

The Golden Ratio The Zometool system inherently relies on the golden ratio γ for the geometry of the nodes (i.e., also the symmetry of the system) and for the lengths of the struts. Geometrically, the golden ratio is defined as the ratio $\gamma := \frac{y}{x}$ for which $\frac{y}{x} = \frac{y+x}{y}$ holds. This is visualized in the inset on the right: by placing a $y \times y$ square (green) along the y side of the $x \times y$ rectangle (blue), the ratio of the sides of the (transposed) new triangle is again $\frac{y}{x}$. From this the equation $\gamma^2 - \gamma - 1 = 0$ can be derived and solved. Being a ratio, γ equals the positive solution: $\gamma = \frac{\sqrt{5}+1}{2} \approx 1.618...$ The golden ratio is an irrational number with several interesting properties, one of which is particularly important in the present context:



- *Powers of γ:* It is clear that $\gamma^0 = 1$ and $\gamma^1 = \gamma$. Not immediately clear, but trivial to show, is that adding one to γ is equivalent to squaring γ, i.e., $\gamma^2 = \gamma + 1$. Similarly, subtracting one is equivalent to the reciprocal of γ, i.e., $\gamma - 1 = \frac{1}{\gamma} = \gamma^{-1}$. This can be continued to a table of γ^k going in both directions ($k \in \mathbb{Z}$), where each power γ^k can be written as a linear, *integer* combination of γ and 1:

$$\forall\, k \in \mathbb{Z},\, \exists\, a, b \in \mathbb{Z} \,:\, \gamma^k = a \cdot \gamma + b \cdot 1, \tag{8.1}$$

showing that each γ^k can be imagined as a 2D integer coordinate (a, b) in the basis $(\gamma, 1)$. This enables a useful, exact representation of the irrational node coordinates of the Zometool system as described further below.

Properties

In the following important properties of the nodes and struts of the Zometool system are listed. Additionally, planes and faces representable by the system are discussed. In particular, details on the subset of planar faces and symmetry planes used by the method in Chapter 10 are given. Finally, an exact representation of the node coordinates as 6-dimensional integers is presented. This proves useful for equating constraints exactly without the need for ϵ tolerances.

Strut Properties For describing and representing the coordinates of the different node locations reachable in the Zometool system, first the possible lengths and directions of struts must be detailed.

- *Strut Lengths:* The shortest blue strut is typically considered to be the unit length in the system, i.e., $b_0 = 1$. The different lengths of same colored struts are related as follows $b_{i+1} = b_i \cdot \gamma$ (analogously for yellow and red). The lengths of differently colored struts are related by $y_i = \sqrt{3}/2 \cdot b_i$ and $r_i = \sqrt{2+\gamma}/2 \cdot b_i$.

- *Zome Vectors*: By combining the 3 different strut lengths with the 62 different node slots (or directions) a total number of 186 positions can be reached from a starting node. Here, these 186 vectors are referred to as the set of *Zome vectors* $\mathcal{V}$, where each $\mathbf{v} \in \mathcal{V}$ is a 3D vector corresponding to a unique slot/strut length combination. The 3D coordinates of each vector $\mathbf{v}$ can be described by 6 integers:

$$\forall\, \mathbf{v} \in \mathcal{V},\, \exists\, \mathbf{a} = (a_0, \ldots, a_5) \in \mathbb{Z}^6 \,:\, \mathbf{v} = \left(\tfrac{a_0\gamma + a_1}{2}, \tfrac{a_2\gamma + a_3}{2}, \tfrac{a_4\gamma + a_5}{2} \right) \in \mathbb{R}^3. \tag{8.2}$$

Node Properties The slots of the nodes define the symmetries of the Zometool system and, more specifically, the directions of the Zome vectors. In turn, combinations of Zome vectors define the reachable node positions.

- *Node Symmetry*: Due to the symmetry of the rhombicosidodecahedron, there is for each slot an opposite slot of the same type and, as $\gamma^2 = 1 + \gamma$, the longest struts, i.e., r_2, y_2, b_2, can be built by combining the two shorter ones of the same type, e.g., $b_2 = b_0 + b_1$. Furthermore, the blue, yellow and red struts of the system correspond to 2-, 3- and 5-fold symmetry axes respectively. This means, e.g., that three symmetric, indistinguishable node configurations are obtainable by rotating a node around a yellow strut by 0°, 120° and 240°.

- *Zome Node Coordinates*: Assuming a starting node at the origin and disregarding possible strut intersections, the set of all reachable 3D positions P of Zome nodes is made up of all linear, integer combinations of the 186 Zome vectors:

$$P := \left\{ \mathbf{p} \in \mathbb{R}^3 \;\middle|\; \mathbf{p} = (0,0,0) + \sum_{i=0}^{185} \mathbf{v}_i \cdot c_i \text{ with } \mathbf{v}_i \in \mathcal{V} \text{ and } c_i \in \mathbb{Z} \right\} \tag{8.3}$$

- *Fixed Node Orientation*: Implicitly used in the definition of the Zome node coordinates above is the fact that all struts (or Zome vectors) only cause a *translation* of nodes, i.e., the orientation of Zometool nodes remains fixed.

Planes Any two non-parallel Zome vectors describe a plane Γ with normal vector $\mathbf{n}_\Gamma$. Collecting all planes with identical normal vectors (disregarding orientation) leads to a total number of 121 different planes. These planes can be divided into 6 different types. The first three types are orthogonal to the direction of a blue, red or yellow strut respectively, the last three types are not orthogonal to any direction in the system (cf. Figure 8.2). Note that 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 elements.

Face Properties 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. The approximation algorithms presented in the following chapters deal exclusively with triangle and quad faces. 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 (in fact, non-planar quads can even be shown to exhibit super-convergence in certain cases [D'A00]), it can be critical for architectural scenarios as discussed in Chapter 2. There are 29 unique triangles shapes in the Zometool system and for $k = 4$ a set of 118 different planar and convex quads can be enumerated (this is explained in more detail in Chapter 10). For higher k the number of such faces steeply rises, making general k-gon-based Zome meshes unsuitable in anti-diversification scenarios.

Exact Zome Coordinates The irrational 3D coordinates in P together with floating point arithmetic make exact comparisons of the form $\mathbf{p}_i = \mathbf{p}_j$ unreliable. Still, such comparisons are often needed for defining various constraints in optimization tasks. To avoid working with ϵ thresholds in such cases, the above 3D floating point coordinates can be transformed to 6D integer coordinates, which can then be compared exactly. Equation 8.1 showed a 2D coordinate representation of powers of γ using a basis $(\gamma, 1)$ consisting of a "golden" part and an "integer" part. This form is very similar to the representation of the individual coordinates of the Zome vectors in Equation 8.2. By multiplying the vector coordinates by 2 or, equivalently using the basis $(\frac{\gamma}{2}, \frac{1}{2})$, each Zome vector in $\mathbf{v} \in \mathcal{V}$ has a unique 6D integer representation $(a_0, \ldots, a_5)$. These 6D coordinates can be added and multiplied by exploiting the equivalence $\gamma^2 = \gamma + 1$ and

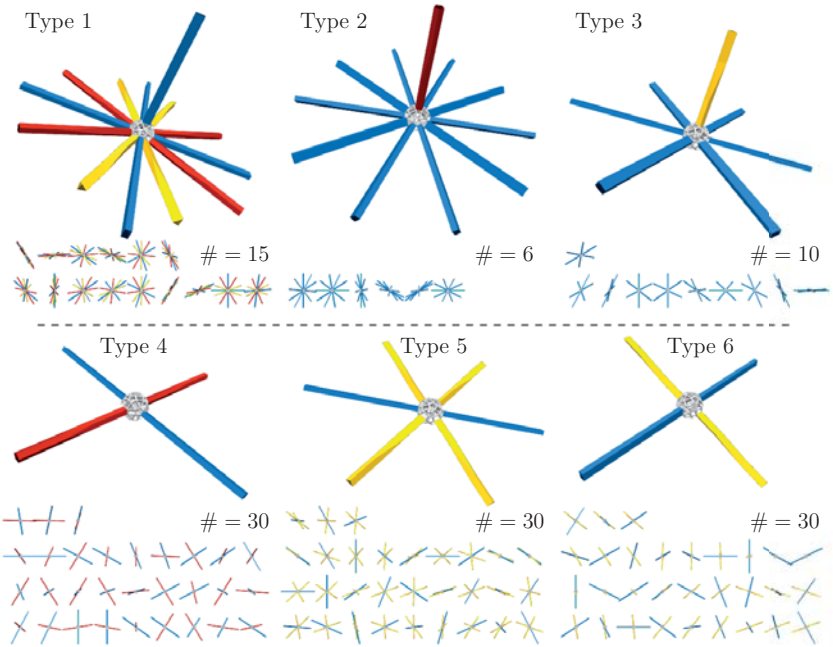
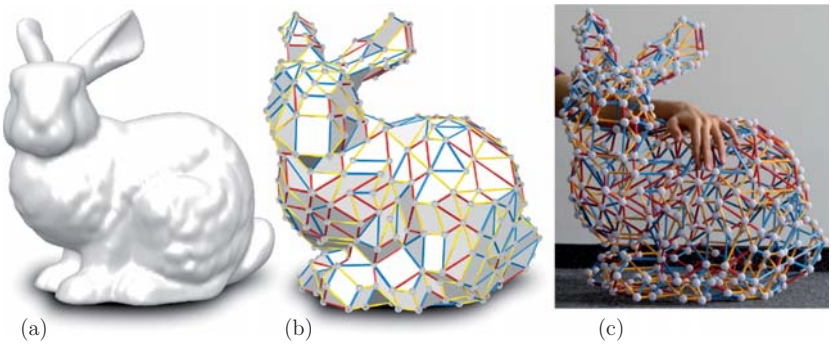


Figure 8.2.: The 121 planes of Zometool system can be divided into 6 types. A representative plane of each type is visualized as a node together with all struts lying in that particular plane. For the first three types additionally the orthogonal strut is shown. The small images below show the different variations (orientations) of planes of the respective type and # states the total number of variations.

compared exactly by their integer coefficients. Consequently, being linear, integer combinations of vectors, also all reachable positions in P posses 6D integer representations. In the following this 6D representation is assumed whenever exact comparisons between Zome coordinates are made.



(a) (b) (c)

Figure 8.3.: Zometool shape approximation: a given freeform input design (a) is approximated by a *Zome mesh* with the same topology (b). (c) shows a real-life Zometool rationalization of the Stanford Bunny.

8.2. The Zometool Shape Approximation Problem

This section introduces a general formulation of the problem of approximating a given freeform surface by a polygon mesh, called *Zome mesh*, whose vertices and edges correspond to Zometool nodes and struts. Figure 8.3 exemplarily visualizes the setting. This problem is the foundation of the following two chapters, where, different requirements motivate different approaches and solutions to the problem.

Common for both scenarios is that a best approximating 2-manifold Zome mesh $\mathcal{Z}$ shall be found for a given input freeform surface $\mathcal{S}$. The Zome mesh approximation is considered qualitative and consistent when it, not only well resembles the geometry of the input shape, but also does not deviate topologically. The geometric quality is defined by an energy functional, sometimes also called cost function, measuring, e.g., the distance difference between the input surface and the approximation. Since the geometric quality can be trivially improved by simply increasing the resolution (number of elements of the Zome mesh) an implicit goal is for the approximation to be as-coarse-as-possible. It is important to note that the constraint of topological equivalence, in contrast to object-derived anti-diversification, here does not imply a fixed connectivity of the approximation. The connectivity needs to be free in order to evolve and facilitate a good geometric resemblance. For consistency, the approximation is, however,

constrained to be topologically equivalent to the input shape. Topologically equivalent shapes are also called *homeomorphic* (cf. [Ede01]).

Surface Representations Note that the Zometool system itself does not imply any surface topology or orientation on the Zome mesh. These properties can be defined by basing the Zome mesh on an underlying 2-manifold halfedge-based mesh data structure – the oriented faces of the base mesh induce a surface topology and orientation on the Zome mesh. Freeform designs are generally instances of NURBS or subdivision surfaces resulting from modern modeling software. For the purposes of this part, the input surface $\mathcal{S}$ is, w.l.o.g., assumed to be a tri mesh of sufficient resolution. For the algorithms in the following chapters, this in particular has the advantage of allowing for efficient point-to-surface projections for distance measuring not based on numerical root-finding.

Problem Formulation Let $\mathcal{S}$ denote the freeform input surface, $\mathcal{Z}$ the Zome mesh and $E(\mathcal{Z})$ the energy functional measuring the approximation error. The approximation problem can now be more formally stated as:

Given $\mathcal{S}$, find $\mathcal{Z}$ such that $E(\mathcal{Z})$ is minimized and $\mathcal{Z}$ is homeomorphic to $\mathcal{S}$.

Not included in this formulation is the matter of geometric integrity, referring to the general assumption and wish that the approximation should not self-intersect (unless prescribed by the input). Being motivated by different applications (general freeform surface approximation vs. panel-aware rationalization) the approximation methods in the next chapters (must) rely on fundamentally different methods and also handle the issue of intersections very differently.

Intersections and Planarity Without the option of continuous, numerical optimization, possible approaches to the approximation problem are limited to making some kind of "discrete" decisions. These can, e.g., be in the form of updates, taking a valid, but possibly poor, initial approximation from one discrete configuration to another one (possibly improving the quality), or by building a valid solution from scratch by iteratively conquering the input shape by placing discrete pieces. The chosen strategy typically prescribes the type of control that can be expected over the results.

After computing an initial, rough approximation the method presented in Chapter 9 iteratively applies modification operators to change local areas of the approximation.

These operators are guaranteed to preserve the topology of the mesh, and allow for a very efficient exploration of the space of Zome meshes surrounding the input shape as no other consistency checks are carried out. It turns out, that by the nature of the used energy functionals the results can generally not only be made free of self-intersections, but the faces are often close to being planar. However, if the modification operators would have been restricted to always guarantee a planar state, the updates would have been much more rigid and the optimization would not be nearly as successful. I.e., there might be no chain of planarity-preserving modifications connecting two valid states.

This necessitates the use of a completely different strategy for the more restrictive setting of rationalizing designs with planar panels in Chapter 10. Here, a growing procedure is implemented, which in each step is guaranteed to place only planar elements not intersecting the already grown part. This extra guarantee, however, comes at the expense of higher complexity and run-time.

Related Work Having polygonal meshes describing both the input and output, the approximation problem can be viewed as a (constrained) remeshing problem from a fine triangle mesh $\mathcal{S}$ to the coarse Zome mesh $\mathcal{Z}$. Typically remeshing algorithms ([AUGA08, BKP*10, Bom12] provide an overview) are guided by continuous measures such as smoothness, inner-angles or alignment of the elements of the resulting mesh and seldomly deal with discrete criteria such as element diversity. In fact, we are unaware of any remeshing techniques dealing with constraints comparable to those posed by the Zometool system. While anti-diversification techniques in architectural geometry do deal with the element diversity, they typically differ to the setting here in two respects: (a) they are based on modifying a given initial solution (with a fixed tessellation) and (b) they allow for a continuous relaxation to make it fit. Neither of these assumptions hold in the Zometool setting.

9. Exploring the Zometool-Shape Space

This chapter deals with one aspect of the Zometool shape approximation problem posed in Chapter 8. In detail, the input here is a closed, 2-manifold freeform input surface $\mathcal{S}$ of arbitrary genus, which is to be approximated by a topologically equivalent Zome mesh $\mathcal{Z}$. The approach described here is based on, starting from an initial rough approximation, exploring the space of Zome meshes around $\mathcal{S}$ using a stochastic optimization method, which iteratively proposes local modifications to the current approximation. By a carefully designed set of local modification operators the exploration can both guarantee topological equivalence and be performed efficiently.

In the previous chapter two different approaches where proposed for dealing with the discrete degrees of freedom imposed by the Zometool system. While a growing process can be advantageous where full control is needed over the placement of every element, e.g., to guarantee planarity, it is rather unsuited for covering closed surfaces. In particular, in the case of a system with a limited set of fixed edge lengths, there is no guarantee that opposing front meeting at some point on the surface can be appropriately merged by any available element. Also, even if the fronts do happen to close up nicely, for higher genus surfaces the validity of the output, i.e., if $\mathcal{Z}$ has the same topology as $\mathcal{S}$, can only be decided once the last element is placed and the Zome mesh is closed. Hence, for dealing with closed surfaces, a method that can guarantee a consistent topology throughout the optimization is advantageous.

Here a set of topology preserving operators is used to update an initial approximation and reduce the approximation error. Consequently, as long as the initial approximation has the correct topology, so will the final result. Basically, any Zome mesh of correct topology, which roughly approximates $\mathcal{S}$, can be used as initial approximation. This chapter presents one possible approach based on a voxelization of $\mathcal{S}$. For efficiently updating the solution a parallelized, simulated annealing-based method is developed. An overview of the approximation algorithm is given in Section 9.1.

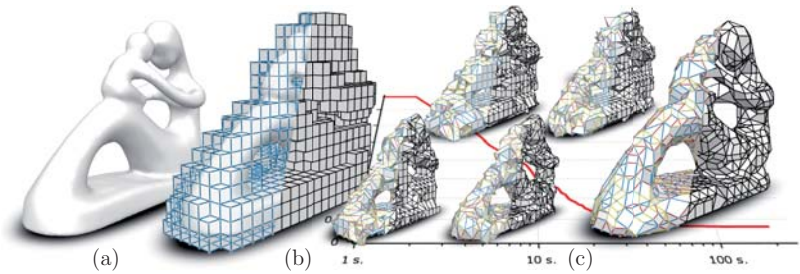


Figure 9.1.: Zometool Surface Approximation Pipeline on the FERTILITY model (a). First a rough Zometool approximation of the surface of the input shape is computed (b). Then, by applying modification operators to the initial approximation, the model-space around the input shape is explored to find an optimized Zometool representation which minimizes an approximation energy (c).

The huge search-space and discrete nature of the Zometool surface approximation problem maps well to the setting of simulated annealing (SA) [MRR*53, KGV83]. SA, often applied on discrete search spaces and complex problems not allowing for any straightforward analytical computation, has been successfully employed in various areas of Computer Vision and Graphics, e.g., for generating good building layout [BYMW13], for approximation of scattered data [KH01], structural reconstruction from images [LDZPD10] or triangle mesh repair [WLG03]. Note that the setting considered here is more constrained, having not only the manifoldness and topological requirements of polygonal meshes but also only a discrete set of possible connecting edges.

9.1. Algorithm Overview

As mentioned above the algorithm consists of two parts: (1) finding a valid, initial approximating Zome mesh and (2) modifying the approximation to minimize the energy and yield a valid output Zome mesh. Figure 9.1 presents a visualization.

The qualities of the initial approximation are critical to the success of the approach. Section 9.2 details how an initial approximation can be constructed as the 2-manifold surface mesh of a voxelization of the input surface. While techniques for handling volumetric data have been established for some time (cf. [KCY93]), the process of extracting consistent, 2-manifold surface meshes from a collection of voxels is not trivial. Non-

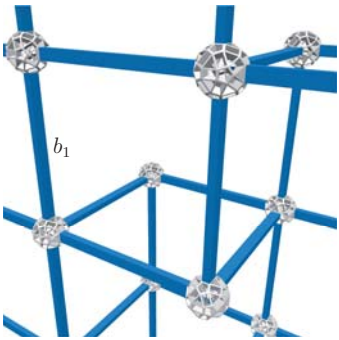
manifold vertices and edges pose problems for the extraction. We present a simple heuristic to alleviate these effects, based on local modification of the voxelization.

To obtain a final, optimized Zome mesh approximation $\mathcal{Z}$, a set of operators are iteratively applied to modify the approximation and reduce the approximation error. Here, the nature of the operators and their order of application is important. A greedy strategy can in general not be expected to find the best solution, the odds are further worsened by the fact that the discrete possibilities of Zometool system make the existence of a direct chain of modification operators, that monotonically decreasing the energy, leading to a useful minimum unlikely. To allow exploring the model-space properly, not only operators updating the geometry (positions of nodes) are required, but also modifications of connectivity and complexity. E.g., one operator moves an existing node to improve the approximation error while another splits a face by inserting a new node to refine the solution locally. Naturally, all such updates must respect and remain within the valid range of the Zometool elements. Section 9.3 introduces the basics of the used simulated annealing optimization method and set of modification operators used. In Section 9.5 implementation details are presented. The energy functionals used for measuring the approximation quality are detailed in Section 9.4. As shown in Section 9.6 not only an energy functional measuring the distance between $\mathcal{Z}$ and $\mathcal{S}$ is important, but also surface orientation and element fairness functionals can be essential to provide high quality approximations and to reduce artifacts such as self-intersections and lost surface details.

9.2. Initial Approximation

Before a voxelization can be computed, a similarity transform must be defined, relating the local coordinates of $\mathcal{Z}$ to those of $\mathcal{S}$. In particular, the "resolution" of $\mathcal{Z}$ is indirectly defined by selecting a scaling factor relating the size of $\mathcal{S}$ to lengths of the struts. We pick the lower left corner of the bounding box of $\mathcal{S}$ to be the origin of $\mathcal{Z}$'s coordinates, and the coordinate axes are defined by aligning three pairwise orthogonal (blue) directions along the axes of the bounding box. To set the scaling it is enough to fix the length of one strut, since the lengths of all struts are related to each other. Here, b_1 is used as the edge length of the voxelization and fixed by the user to set the resolution. b_1 has the advantage of being the most flexible edge length (between b_0, b_1 and b_2) in the sense that it allows for the largest number of local modification operations, i.e., the cardinality of

its SplitVector set (cf. Section 9.5) can be shown to be the largest. Being the mid length it is also the most visually intuitive choice for the user, since it better corresponds to the average strut lengths appearing in the final approximation $\mathcal{Z}$, where as b_0 and b_2 rather correspond to the minimal and maximal expected strut lengths instead. The choice of scaling defines the resolution of the voxelization and the real-life size of the final output $\mathcal{Z}$. An *appropriate* scale should optimally allow for preserving the genus of $\mathcal{S}$ while not being too fine to yield an overly tessellated result. However, there is unfortunately no rule on how to choose an appropriate scaling factor with such guarantees. Even if there in theory exists a voxel arrangement with the desired genus for every (non-degenerate) choice of b_1 this might have *very* little to do with the input shape. Also, even at an appropriate scale with the correct genus, the voxelization might contain non-manifold edges and vertices which hamper the extraction of a 2-manifold $\mathcal{Z}$. Luckily, such configurations can often be removed by applying local, topology preserving voxel operations, as explained in the following.



To compute the voxelization, the bounding box of $\mathcal{S}$ is first extended by $\frac{1}{2}b_1$ in each direction and split into b_1-sized cells. Then by iterating over all mesh primitives of $\mathcal{S}$ (this can be done in parallel) the cells covering the mesh or lying inside are tagged. The tagged cells define a *conservative* voxelization which completely covers the input.

To extract the 2-manifold boundary $\mathcal{Z}$ it is assumed that an appropriate scale b_1 has been chosen and that that a voxelization of the wanted genus has been computed. Then, as long as non-manifold configurations exist, *simple* voxels incident to these configurations are removed. A voxel is called *simple* if it does not change the genus (cf. [BK03]). If not all non-manifold configuration can be removed by these local operations, this happens very rarely in degenerate configurations, the user can adjust the scale factor slightly and retry. Finally, when no non-manifold configurations remain, the outer surface of the voxelization is a 2-manifold quad mesh, which can be trivially extracted.

This section presented one way to obtain efficiently obtain an initial approximating Zome mesh. However, note that the optimization approach described in the following, does not require a voxelization but can work with any valid Zome mesh roughly approximating the input.

9.3. Simulated Annealing

Simulated annealing (SA) optimization is adopted to let the initial configuration evolve, and to explore the space of Zome meshes. Simulated annealing can be seen as an optimization technique for non-convex energy functionals. Based on an iterative mechanism, a local modification of the current configuration is proposed at each iteration. This proposition of modification is then accepted or rejected depending on both a quality measure and a certain degree of randomness. Contrary to deterministic local optimization algorithms, SA can escape from local minima. A simulated annealing mechanism is specified by three important components:

- *Local Operators.* They are used to generate local modifications of the current configuration. The operator set must be rich enough for each configuration to be reachable from any other configuration in a finite number of steps. Also, a local modification by an operator has to be reversible, i.e., the inverse modification must be possible.

  In the current setting, three types of operators are required, operators for *increasing* model complexity, *changing* model geometry and *decreasing* model complexity (here, model complexity is measured by the number of nodes N). Simple node and face based operators can be applied to locally modify the geometry and complexity of the mesh by inserting, moving and removing nodes. To be able to explore different connectivities also strut-based operators are required. For example, a strut can be removed to merge two faces. We implemented a set of 7 different operators.

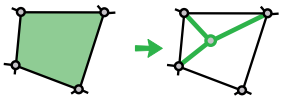
- *Energy.* It measures the quality of a configuration $\mathcal{Z}$ in the model space. Our energy, detailed in Section 9.4, is composed of terms evaluating (i) the geometric accuracy of $\mathcal{Z}$ with respect to the input shape $\mathcal{S}$, and (ii) the structure of $\mathcal{Z}$ in terms of complexity and fairness.

- *Cooling schedule.* It specifies the form of the relaxation parameter T_t, also called temperature, and its initial value T_0 that both control the degree of randomness of the simulated annealing. The temperature T_t is a decreasing series approaching zero as t tends to infinity. Note that a logarithmic decrease of T is necessary to ensure the convergence to the global minimum from any initial configuration. However, in practice, one uses a faster geometric decrease of the form $T_t = T_0 \cdot \alpha^t$ which gives a good approximate solution close to the optimum in general [HJJ03].

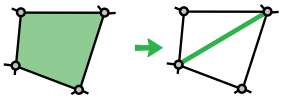
Set of local Operators

Below, the implemented operators are listed together with a brief description explaining their functioning and a figure of the respective support regions. The boundary of a support region is referred to as *link*.

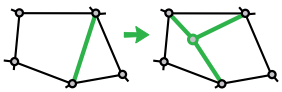
• InsNode(f) inserts a new node in the quad face f and adds all possible connections to the nodes of the link.



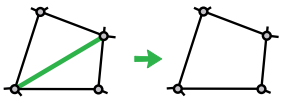
• AddDiag(f) splits the quad face f by adding a diagonal strut.



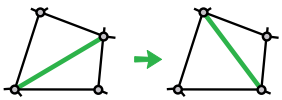
• SplitStrut(s) splits the strut s by inserting a new node and forms all possible new connections to the link.



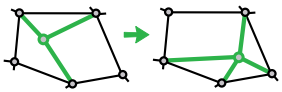
• RemDiag(s) removes the (diagonal) strut s separating two triangles.



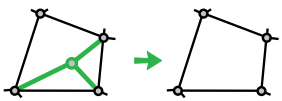
• FlipDiag(s) flips the (diagonal) strut s separating two triangles.



• MovNode(n) moves the node n and forms the possible connections to the link



• RemNode(n) removes the node n and its connections to the link.



Note that the operations InsNode and MovNode are not bound to any certain connections, but rather the new node (position) is simply always connected to as many link nodes as possible. For SplitStrut the new node is additionally required to connect to the initially adjacent nodes. Also, note that except for FlipDiag, RemDiag and RemNode the operators are generally not unique, e.g., there are typically several possible node positions which can be validly connected to the link. Details on how the

operators can be efficiently computed without enumerating all possible link connectivities are given in Section 9.5.

Operator Validity The operators are constrained to only yield triangle and quad faces. For operators adding or changing geometry (inserting a node, splitting a strut or moving a node) this is checked by first (virtually) forming all possible connections from the center node to the nodes in the link and then testing for faces of valence > 4. Also to support physical realizability of the result, (local) feasibility constraints reject operations where multiple nodes are at the same position and/or multiple struts are using the same slots. Some configurations require special care in order to guarantee 2-manifold results, details on this are presented in Section 9.5.

Parallelization The conventional simulated annealing performs successive local modifications on the current configuration. Such a mechanism is obviously long and fastidious. To speed-up the exploration, local modifications can be performed in parallel when located far enough apart. To allow for an efficient parallelization it is crucial that the influence area of an operator on the mesh is small and localized, as this area can only be processed by a single thread at a time to guarantee consistency of the underlying mesh. These regions, later "tagged" by different threads in the optimization, are referred to as *tag regions*. For the different types of operator entities (marked green) Figure 9.2 shows the different tag regions as red marked nodes. The outer oriented ring bounding the affected (support) region is referred to as the *link* and is marked by arrows. Note that the orientation is used to distinguish between a Zome vector $\mathbf{v} \in \mathcal{V}$ and the vector $-\mathbf{v}$ pointing in opposite direction.

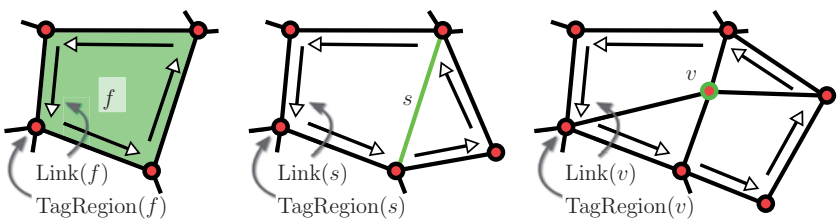


Figure 9.2.: The TagRegion (red nodes) and oriented edges of the Link (arrows) of different mesh entities (green).

9.4. Approximation Energy

The approximation energy $E(\mathcal{Z})$ is a linear combination of different energy terms, accounting for different aspects of the approximation,

$$E(\mathcal{Z}) = w_\mathrm{d} \cdot E_\mathrm{distance}(\mathcal{Z}) + w_\mathrm{o} \cdot E_\mathrm{orientation}(\mathcal{Z}) + w_\mathrm{f} \cdot E_\mathrm{fairing}(\mathcal{Z}) + w_\mathrm{c} \cdot E_\mathrm{complexity}(\mathcal{Z}),$$

detailed in the following. The first two terms measure the faithfulness to the input $\mathcal{S}$ and the last two are shape and structure priors for the output $\mathcal{Z}$. Evaluating the energy requires comparing properties of positions on $\mathcal{Z}$ with the properties of their nearest position on $\mathcal{S}$. To this end a projection operator $\pi : \mathbb{R}^3 \to \mathcal{S}$ is used for projecting positions $\mathbf{p}$ to their nearest points $\pi(\mathbf{p})$ on $\mathcal{S}$, let $\mathbf{n}_{\pi(\mathbf{p})}$ be the normal vector on $\mathcal{S}$ at this position. Note that in the following bold characters denote 3D positions and vectors, e.g., the 3D position of a node a is $\mathbf{a}$.

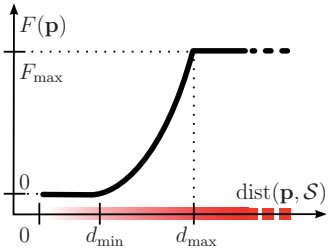
Distance

The distance from $\mathcal{Z}$ to $\mathcal{S}$ is integrated by samples $\mathbf{p}_i$ over all nodes, strut midpoints and face barycenters of $\mathcal{Z}$:

$$E_\mathrm{distance}(\mathcal{Z}) = \frac{1}{S \cdot b_0^2} \sum_{i=1}^{S} \|\mathbf{p}_i - \pi(\mathbf{p}_i)\|^2 \cdot (1 + F(\mathbf{p}_i))$$

where S denotes the total number of samples (#nodes + #struts + #faces) and b_0 normalizes the energy to the range $[0, 1]$ for positions with distances of less than b_0 from $\mathcal{S}$. The term $F(\mathbf{p}_i)$ is called *forbidden zone*, it is introduced to further penalize points lying too far away from the surface of $\mathcal{S}$.

Forbidden Zones In combination with thin surfaces details (e.g., arms on the FERTILITY model) high SA temperatures (typically in early stages of the exploration) can lead to configurations where a node gets displaced from one side of the arm to the other. Depending on the energy weights, the node might not be able to move back. To counter-act this effect and discourage mesh primitives from passing through the interior of $\mathcal{S}$, the interior is made more expensive (forbidden) for the optimization by introducing the *forbidden zone* term. $F(\mathbf{p})$ is a quadratically increasing function which depends on the distance



of the position $\mathbf{p}$ to the surface $\mathcal{S}$: It is equal to zero for positions lying close to the surface of $\mathcal{S}$, then increases quadratically after a certain distance $d_{\min}$ until assuming the maximal value $F_{\max}$ at $d_{\max}$. The standard weights used in the evaluation (cf. Section 9.6) focus on penalizing interior points to avoid degeneracies: $d_{\min} = b_0/3$, $d_{\max} = b_0 \cdot 1.5$ and $F_{\max} = 35$. For outside points the weights are set to: $d_{\min} = b_0/3$, $d_{\max} = b_0 \cdot 2.5$ and $F_{\max} = 15$. This choice works well in practice (cf. Figure 9.9), but can also further be adapted to object specific needs.

Orientation

The orientation energy consists of a *tangential* part measured on struts and a *normal* part measured at face corners. Both parts are in the range $[0, 1]$. Let s be a strut with endpoints $\mathbf{a}$ and $\mathbf{b}$, direction $\mathbf{d} = (\mathbf{b} - \mathbf{a})$ and midpoint $\mathbf{m} = (\mathbf{a} + \mathbf{b})/2$. The tangential part measures the deviation of the strut direction $\mathbf{d}$ from the tangent plane at the point on $\mathcal{S}$ closest to $\mathbf{m}$:

$$E^{\text{tangent}}_{\text{orientation}}(\mathcal{Z}) = \frac{1}{S} \sum_{s \in \mathcal{Z}} \frac{(\mathbf{d}^\top \mathbf{n}_{\pi(\mathbf{m})})^2}{\|\mathbf{d}\|^2},$$

where S is the number of struts in $\mathcal{Z}$. Now, let $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ be the three ccw oriented node positions at the corner b of a face f, the normal part measures the deviation of the normal defined by these points and the closest normal on the surface:

$$E^{\text{normal}}_{\text{orientation}}(\mathcal{Z}) = \frac{1}{2C} \sum_{f \in \mathcal{Z}} \sum_{b \in f} \left(1 - \mathbf{n}^\top_{\pi(\mathbf{b})} \frac{(\mathbf{c} - \mathbf{a}) \times (\mathbf{b} - \mathbf{a})}{\|\mathbf{c} - \mathbf{a}\| \|\mathbf{b} - \mathbf{a}\|}\right)^2,$$

where C is the number of corners in $\mathcal{Z}$. Note that while the tangential part only expresses co-planarity, the normal part also encompasses orientation of the normals. This property is later (cf. Section 9.6) shown to be crucial for avoiding fold-overs and self-intersections. Also, as the orientation of the normal flips for reflex angles, this energy additionally expresses face concavity. The two parts are combined as $E_{\text{orientation}} = 0.25 \cdot E^{\text{tangent}}_{\text{orientation}} + 0.75 \cdot E^{\text{normal}}_{\text{orientation}}$ which gives slightly more importance to the normal part.

Fairing

To increase element regularity we introduce a fairing energy based on the uniform Laplacian operator at each node p:

$$E_{\text{fairing}}(\mathcal{Z}) = \frac{1}{N \cdot b_0^2} \sum_{p \in \mathcal{Z}} \|\mathbf{p} - \frac{1}{|N_1(p)|} \sum_{p_i \in N_1(p)} \mathbf{p}_i\|^2,$$

where N is the number of nodes in $\mathcal{Z}$ and $N_1(p)$ is the set of neighboring nodes in the 1-ring of p. This energy also helps to reduce fold-over configurations.

Mesh Complexity

Let N be the current number of nodes in $\mathcal{Z}$ and N_{target} a specified target complexity. The mesh complexity energy is the quadratic deviation from the target complexity:

$$E_{\text{complexity}}(\mathcal{Z}) = \frac{1}{N_{\text{target}}}(N - N_{\text{target}})^2.$$

9.5. Implementation Details

The implementation is based on a standard halfedge-based mesh data structure (`www.openmesh.org`) to represent both $\mathcal{S}$ and $\mathcal{Z}$, and relies on OpenMP (`www.openmp.org`) for parallelization. The projection operator $\pi(\mathbf{p})$ is implemented as a BSP on $\mathcal{S}$ supported by a spatial hashing data structure in a crust around the surface to cache the result for previously queried positions.

The Operators

2-Manifoldness of Operators To keep implementation simple all operators are built upon combinations of the four basic functions supplied by most halfedge-based mesh data structures: *add/remove vertex* and *add/remove face*. While these functions themselves are safe, there are some pathological configurations which need to be handled. E.g., recall the MoveNode(n) operator from Section 9.3, which generally leads to a new connectivity inside the link of the node n. This operator can be implemented in three simple steps: (1) clear the old support region, i.e., remove all faces, (2) move the node and (3) add the new faces. However, even starting from a valid 2-manifold input, e.g., consider moving the green vertex in Figure 9.3(b), already the first step (removing old

faces) can lead to a non-manifold configuration, from which the following steps and the data structure may not be able to recover. The following prominent, problematic cases are visualized in Figure 9.3:

- Flipping certain edges (e.g., in a topological tetrahedron) can lead to pairs of nodes "doubly"-connected by two edges. While theoretically still 2-manifold, this configuration is problematic in practice, cf. Figure 9.3(a).
- Clearing the faces of support regions in which a single node is pointed to by more than one of the oriented link edges, yields a non-disk neighborhood around this node, cf. Figure 9.3(b).
- Removing an edge of a valence 2 node leads to a dangling valence 1 node and a non-disk neighborhood, cf. Figure 9.3(c).

Some of these cases are even manifold in theory, but, due to data structure limitations, usually are classified as not, and hence lead to problems in practice.

In short, all operations which lead to doubly connected vertices and operations where a node in the link is pointed to twice by the oriented edges are disallowed.

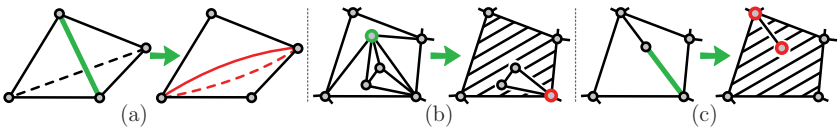


Figure 9.3.: When implementing the modification operators care needs to be taken to avoid problematic situations, such as, *complex*, doubly-connected edges (a) and non-2-manifold, dangling vertices (b,c).

Look-Up Tables of Basic Two-Strut-Operations Enumerating all possible (and valid) Zometool fillings for an arbitrary link can not be done efficiently. Even the much smaller setting where the filling is restricted to add only a single new node is complex. We reduce the complexity by not considering the whole link at once but by building the operators on combinations of simple atomic operations involving only *two* struts at a time. For this purpose we pre-compute three simple lookup tables based on the Zome vectors $\mathcal{V}$:

- The $\textsc{SplitVector}(\mathbf{v}) = \{(\mathbf{v}^*, \mathbf{v}')\}$ table maps a Zome vector $\mathbf{v}$ to a set of *pairs* of vectors $(\mathbf{v}^*, \mathbf{v}')$ with $\mathbf{v} = \mathbf{v}^* + \mathbf{v}'$, meaning a Zome strut with vector $\mathbf{v}$ can be replaced by two struts (connected by a new node) with vectors $\mathbf{v}^*$ and $\mathbf{v}'$.

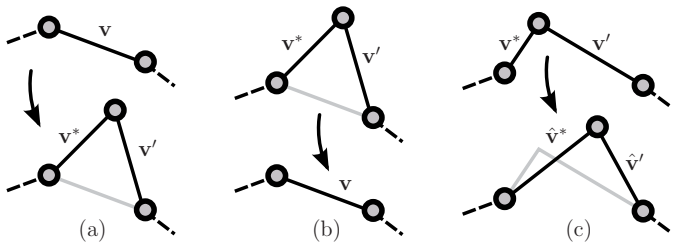


Figure 9.4.: All modification operators are based on combinations of three atomic operations: (a) SplitVector, (b) MergeVector and (c) ChangeDirections.

- MergeVectors$(\mathbf{v}^*, \mathbf{v}')$ is the table of inverse operations, mapping a pair of vectors $(\mathbf{v}^*, \mathbf{v}')$ to *at most* one direction $\mathbf{v}$ (not all pairs of vectors correspond to a single strut).

- The ChangeDirections$(\mathbf{v}^*, \mathbf{v}') = \{(\hat{\mathbf{v}}^*, \hat{\mathbf{v}}')\}$ table, maps a pair of vectors $(\mathbf{v}^*, \mathbf{v}')$ to a set of new pairs $(\hat{\mathbf{v}}^*, \hat{\mathbf{v}}')$, thereby effectively moving the center node between the struts.

Note that, contrary to the split and merge operations, the end-nodes involved when changing directions need not be connectable by a single strut. Also, there may be more than one possible split for a given strut, yielding an implicit "geometric" degree of freedom. The computation of the tables is straightforward and takes only a few seconds. In total there are 8472 split (and merge) operations and 686184 moves.

Generating an Operation Generating an operator is done by a set of (random) lookups in the aforementioned tables. The InsNode, MovNode and SplEdge operators are similar: (1) the support region is cleared (all faces removed), (2) a new node position is found by using either a SplitVector or ChangeDirections operation (random choice) on one or two oriented edges of the link respectively, (3) all connections between the new node and the link nodes are formed, (4) if all constraints are fulfilled the operator is valid. Furthermore, AddDiag is based on a random MergeVectors operation and FlipDiag, RemDiag, and RemNode are straightforward.

Parallelized Simulated Annealing

The simulated annealing mechanism used to explore the space of different Zometool meshes is introduced in Algorithm 9.1. Algorithmically all the local operators (cf. Section 9.3) have a common interface of functions regardless of their input entity (node, strut or face) and all hold a reference to the global energy and Zome mesh $\mathcal{Z}$:

- `rand_entity()` – gets a random entity handle from $\mathcal{Z}$
- `tag_region()`* – tags the local influence area of the operator on $\mathcal{Z}$ around the handle (if it has not already been tagged by another operator)
- `valid()` – tests if the operation is topologically valid
- `simulate()` – computes the energy difference ΔE by simulating the effect of the operation
- `commit()`* – performs the operation and updates the energy
- `untag_region()`* – untag the local influence area

The functions marked by an asterisk (*) modify the mesh data structure (by setting tags or removing/adding entities) and must be protected by appropriate semaphores in a parallelized setting, e.g., `critical` sections in OpenMP.

When a thread wants to perform an operation on some mesh entity (e.g., inserting a node in a face) the influence area of the operator on that handle is first tagged (not to be touched by another thread). Then the operator evaluates the topological validity and energy update which would be caused by performing the operation. If the operation is valid and the proposition of modification is accepted then the operation is carried out. Finally, the region is un-tagged and is again free to be used by other threads/operators. Note that for data structures based on lazy-updates, removing faces or vertices actually does not decrease the size of the data structure (which grows with every add-operation). In such cases an occasional *garbage collection* step might be necessary – this reorganizes the memory of the whole data structure and must only be performed in a synchronized state by a single thread while the others are idle.

Algorithm 9.1: SA for Zometool Approximation

Input: input surface $\mathcal{S}$, initial mesh $\mathcal{Z}$, set of operations $\mathcal{O}$, initial temperature T_0, min. temperature $T_{\min}$ and decrease factor α

Shared vars.: done, T_i, α, $\mathcal{Z}$, time

- - - Begin Parallel Region - - -;

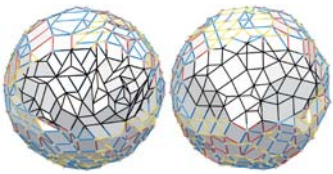
```
while done = false do
    o = random_operation(O);
    e = o.rand_entity();
    tagok = o.tag_region(e);
    if tagok then
        if o.valid() then
            ΔE_o = o.simulate();
            p = uniform_prob[0,1];
            if p > exp(-ΔE_o / T_i) then
                o.commit();
                cnt++;
            end
        end
        o.untag_region();
    end
    T_i = T_0 · α^cnt;
    if time up or T_i < T_min then
        done = true;
    end
end
```

- - - End Parallel Region - - -;

9.6. Evaluation

Zometool shapes computed by the presented method are shown in Figure 9.9. The used models are courtesy of AIM@SHAPE shape repository (`http://shapes.aim-at-shape.net`) and McGill 3D Shape Benchmark (`http://www.cim.mcgill.ca/~shape/benchMark/`). For comparability all results were computed using 7 threads on a standard i7-PC using the *same* parameter settings (unless stated otherwise). Note that the examples have intentionally been computed with coarse resolutions to challenge the method. Naturally, increasing the resolution of the voxelization trivially yields even better approximations.

Weights and Parameters In the experiments the weights and parameters have been fixed to: $w_\mathrm{d} = 10$, $w_\mathrm{f} = 5$, $w_\mathrm{o} = 100$, $w_\mathrm{c} = 1$, $T_0 = 0.1 \cdot Stdev(\Delta E(\mathcal{Z}))$, $T_\mathrm{min} = 0.00001$, $\alpha = 0.999995$. The target complexity is $N_\mathrm{target} = N^0$, with N^0 being the number of nodes of the initial $\mathcal{Z}$. A high orientation and fairing energy help avoid degenerate and fold-over configurations (especially) during the critical (high temperature) phase of the optimization. The orientation energy prevents fold-overs by penalizing flipped normals and can in general be chosen generously. It can furthermore be utilized to mesh sensitive configurations not possible with the standard parameters (discussed below). By "pulling" nodes apart the fairing energy regularizes the result. The inset shows a sphere mesh with low (left) and high (right) w_f. However, care must be taken as, in combination with a coarse resolution, such a regularization can smooth out small surfaces details. While with these settings very good results were obtained within the range of a couple of minutes, they are not optimal for all shapes. The paragraphs below discuss the influence of different weights and how they can be used to obtain good results even at very coarse resolutions.



Energy Evolution and Convergence Since the orientation energy is always in the range of $[0, 1]$, whereas the distance energy is much bigger than 1 for distances outside the range $[0, b_0]$ (cf. Section 9.4), this means that distance minimization is prioritized in the beginning of the optimization, while orientation dominates the end. This was confirmed by starting ten identical runs on the RockerArm. The initial energy was dominated by the distance term whereas the final energy was comprised mainly of the orientation:

	Initial		Final (avg)	
E_distance	12.5	82%	0.039	13%
$E_\mathrm{orientation}$	2.68	18%	0.226	73%
E_fairing	0.06	0.0%	0.042	14%
$E_\mathrm{complexity}$	0.0	0.0%	0.0	0.0%
E	15.24	100%	0.318($\pm$0.013)	100%

Each computation took only 3 min. The low standard deviation (0.013) further suggests that a common, nearby minimum has been found. However, although visually pleasing results, the meshes can still differ in the details as demonstrated in Figure 9.5(a,b).

Comparing the above energy to that of a "ground truth" computed with a slower cooling schedule over $24h$ time (cf. Figure 9.6), we can conclude that these are very good local optima.

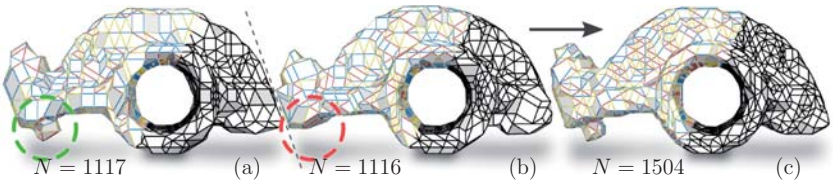


Figure 9.5.: (a) and (b) demonstrate how using SA the same weights can sometimes lead to slightly different results. It is possible to use further optimization passes to refine the result: The mesh in (c) is obtained starting from (b) by allowing additional nodes.

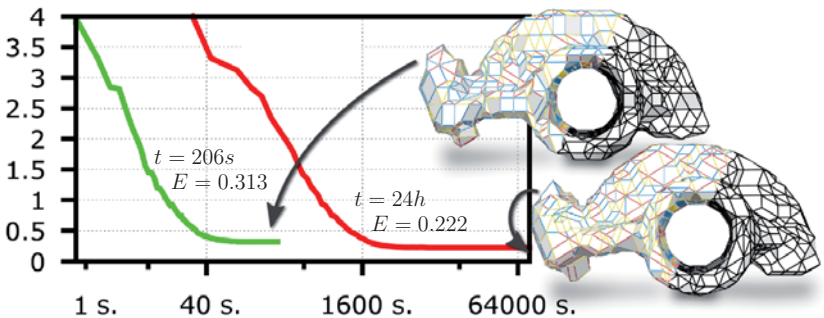


Figure 9.6.: The top RockerArm was obtained after about $3min$ with a final energy of 0.313, while the bottom result was obtained after $24h$ with a final energy of 0.222.

Exploring Model Complexity Once a good solution has been found, the parameters can be tailored to modify the solution further (by running the optimization again, with the previous solution as initialization). Figure 9.9 (along the green strip) shows 3 meshes out of 8 from of two different fixed-scale hierarchies respectively, one *refinement* hierarchy of the RubberDuck and one *decimation* hierarchy of the Fertility. These hierarchies were computed in 7 iterations by starting from the finest/coarsest mesh and in each step decreasing/increasing the complexity by around 30%/25%. The decimation hierarchy

demonstrates how genus-features are preserved through-out the hierarchy. Even at very coarse resolutions the topology preserving operators keep the arms 2-manifold and the *forbidden zone* keeps them in place. The refinement hierarchy shows how, with increasing resolution, surface details (such as the bill of the duck) become increasingly well developed. While the decimation has a natural "saturation-point" (basically when only a very coarse mesh with long edges remains), new nodes on the other hand can always be inserted when refining. Enforcing too many points eventually leads to fold-overs and self-intersections as the distance energy tries to keep $\mathcal{Z}$ close to $\mathcal{S}$. This is an effect which cannot be handled by local decisions alone but would require more global mechanisms. However, a guarantee against global self-intersections is not trivial to realize, especially not efficiently in a parallelized setting. Luckily, in "not too extreme" cases such effects are implicitly handled by the orientation and fairing energies, but there are no 100% guarantees. All shown examples, computed using the standard weights, are fold-over free.

The possibility to increase resolution within the given scale can also effectively be utilized as a post processing operator to reconstruct small missing surface details: by increasing N_{target} and tightening the *forbidden zone* the missing knob on the ROCKER-ARM in Figure 9.5(b) could be restored. Note that such modifications can also trivially be localized by permanently tagging all nodes except those around the problematic area. Although it is tempting to tighten the forbidden zone preemptively, this heavily constrains the exploration and leads to inferior solutions.

Preserving Thin Surface Parts If a very thin part of the surface is lost, it might not be reconstructable by post-optimization. The ELK model has two critical areas: (1) the thin separations between the "wheels" and the body, and (2) the antlers, which are even thinner (!) than the shortest strut of the system y_0 (cf. Figure 9.7). On the ELK result in Figure 9.9, using the standard weights, the wheels and the body are separated but the antlers are partly lost. The above mentioned post-optmization approach fails in this case, due to the antlers being thinner than any strut, causing no or only very few samples of E_{distance} to fall into the *forbidden zone*. Hence, care must be taken that such details are never lost in the first place. Here the properties of the orientation energy can be exploited further. By using an even higher w_{o} (e.g., 500) a rounder, thicker result is obtained with the antlers still intact, but with a poor distance approximation (cf. Figure 9.8). Now, by iteratively tightening the *forbidden zone* from the outside, i.e., by increasing $F_{\max}$, and decreasing the orientation weight, the final result (right) was

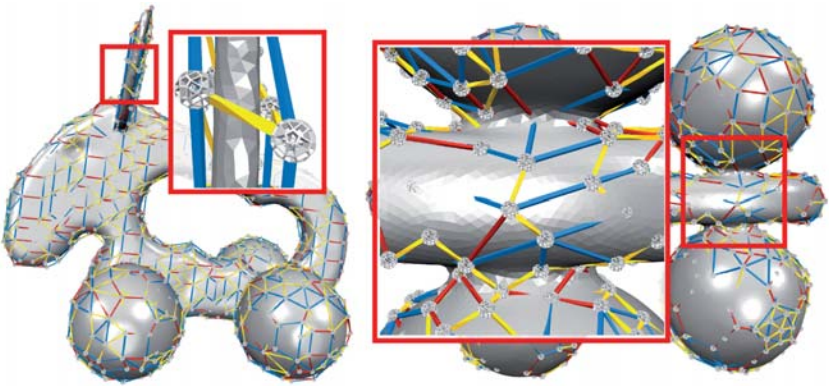
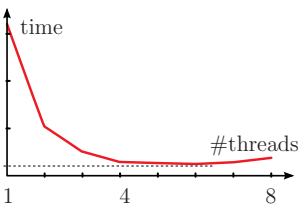


Figure 9.7.: At the chosen resolution the antlers are thinner than y_0 and it is a difficult task to separate the "wheels" from the body of the ELK.

obtained in about 15 minutes, with each iteration taking about 5 min. The Stanford BUNNY was also computed using this "3-step" strategy.

Scalability A performance speed-up of about $3 - 4$ times can be trivially achieved by parallelizing the optimization loop. However, after an initial (super-)linear scaling with the number of new threads/cores, the locking mechanisms, needed to keep the mesh consistent, take the upper hand. The initial, excellent performance gain can be explained by the validity check of an operation not requiring a lock, allowing for one thread to check an operation even if another thread has locked the mesh for modifications. The exemplary time vs. number of threads plot in the inset shows that no performance gain is expected from the current implementation when using more than 7 threads. This achievable gain also much depends on the type and number of locks used, and further optimizing these parameters could slightly improve the plot. An experiment to achieve consistent linear speed-up behavior was performed by using a patch-based approach that iteratively: (1) patches the surface into independent parts (thereby alleviating the need for locks) and (2) optimizes the patches separately. While the actual optimization now



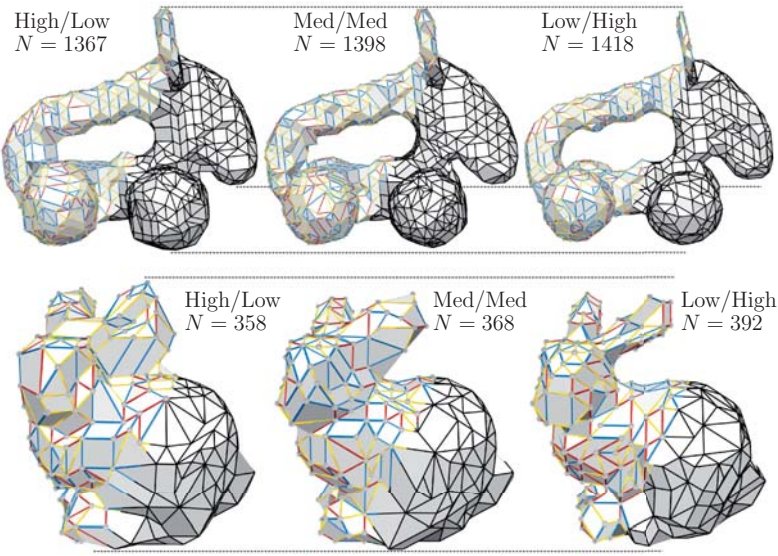


Figure 9.8.: A 3-step scale of decreasing orientation weights and increasingly tightened forbidden zone weights (High/Low to Low/High) demonstrating how thin surface parts (ELK antlers and BUNNY ears respectively) can be preserved while still properly reaching concavities (e.g., between BUNNY ears or ELK wheels, also cf. Figure 9.7). The dotted lines show how high orientation tends to "bloat" the surface. Using the resulting N of the previous step as the new N_{target} has a relaxing effect on the complexity.

scaled well, too much time was lost in stitching the patches back together and recomputing a new layout. The convergence properties of this approach are also questionable, since at no point the complete global energy is optimized.

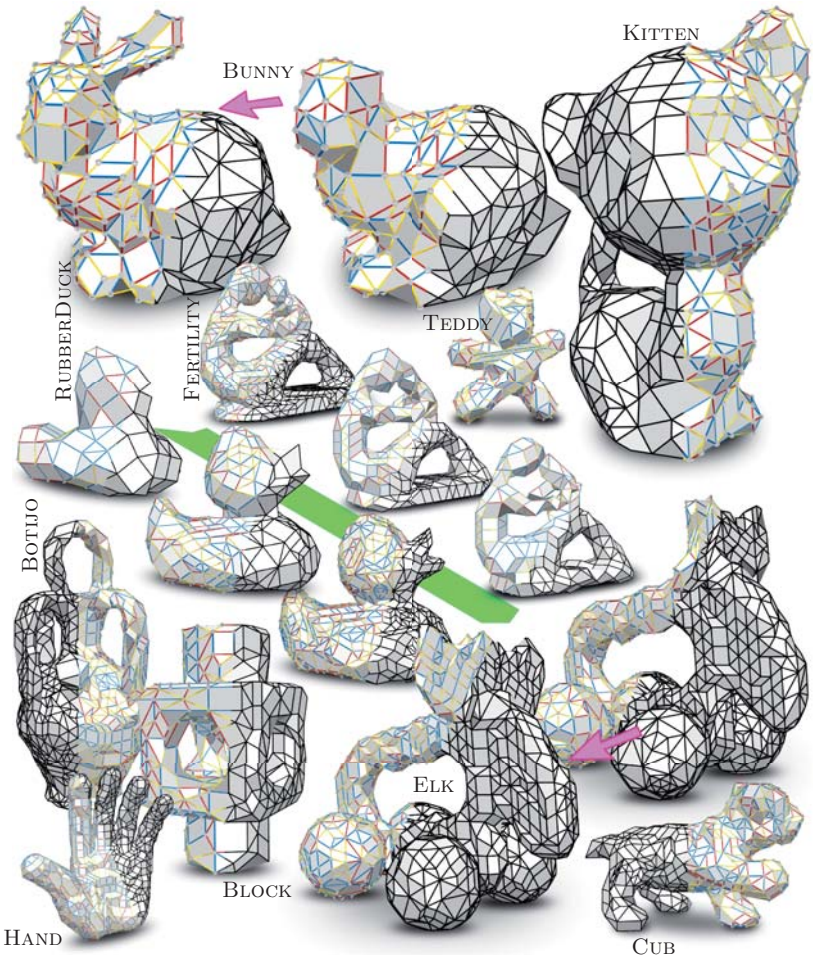


Figure 9.9.: Resulting Zometool meshes. Along the green strip excerpts from a refinement and a decimation hierarchy are shown. At the shown coarse resolution, the Bunny and Elk both had details missing using the standard weights. By using the 3-step strategy to preserve thin features (discussed in Section 9.6) these details were successfully captured (pink arrow).

10. Zometool Paneling of Freeform Patches

Zometool is a powerful system which, unrestricted, allows for a huge variety of different panel types. For the task of providing efficient paneling of freeform patches (disk topologies), in this chapter only a small subset consisting of 29 triangles and 118 convex (planar) quads is considered. Due to the fact that there are more such quads than triangles, the paneling method presented here 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. This adds yet another discrete constraint to the already restricted approximation problem presented in the previous chapter. There, all possible quads and triangles were allowed, which could be formed based on the available set of struts. The performance of the approximation in the previous chapter relied heavily on the fact that, during the exploration, different concave, non-planar and even self-intersecting configurations are allowed for the energy to efficiently explore also remote corners of the solution space. The fact that the results were generally well-natured, i.e., non-intersecting and largely planar, was mainly due to the choice of energy functionals and properties of the optimization procedure. However, there were no guarantees. As noted in Chapter 8, additionally constraining the modification operators would severely hamper the exploration freedom of such methods and their capabilities to escape local minima.

Here a different approach is required to guarantee well-shaped panels throughout the whole process. The complexity and inherent discreteness of this problem exclude the possibility of a direct solution or even relaxation of the problem and make finding a globally optimal solution infeasible in general. Faced with such problems one is in a sense restricted to making (the best of) local decisions. One 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 elements where needed. In general, two

different parts of a front meeting somewhere on a curved surface can always be joined by inserting a custom element. This is not possible in the Zometool system, where there may be no fitting (set of) elements available for appropriately joining the two sides. Front self-collisions are related to the topological structure of a surface [NGH04] and are inevitable on closed surfaces and surfaces with genus $\neq 0$. While the growing technique presented in this chapter is restricted to disk topology surfaces, this does not directly imply a restriction to simple geometries. As demonstrated by the results also doubly-curved surfaces with concavities and small boundaries can be paneled. Unfortunately, front collisions can occur also on open flat surfaces if the growing order is not handled properly. To this end our method relies on a special growing order to guide the process.

A vast number of advancing front remeshing strategies exist, with applications ranging from Delaunay mesh generation [FBG96], point cloud interpolation [BMR*99], height-field triangulation [SM98] to extracting iso-surfaces from implicit functions [SSS06] to name a few, but all without the fixed-struts and fixed-panels restrictions posed in a Zometool-setting.

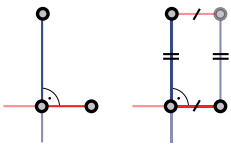
10.1. Algorithm Overview

Presented in this chapter is a novel advancing front technique to grow the Zome mesh $\mathcal{Z}$ over the input freeform surface patch $\mathcal{S}$, based on a pre-computed *harmonic* growing field designed to avoid front self-collisions. The front is generally not grown on a face-by-face basis but rather locally optimal solutions are computed to fill gaps on the front in a "best-fitting" manner. This growing performs well on general freeform patches, however, on objects with a reflectional symmetry, asymmetric results can be distracting. To further enable reflective symmetric results a simple, but effective symmetry plane constraint is introduced.

Section 10.2 presents the foundations of our advancing front approach. How reflectional symmetries can be respected by the growing is explained in Section 10.3. Different experiments were performed, e.g., to investigate the degrees of freedom of different symmetry plane types and finding good weighting factors for the energy functionals, these are described in Section 10.4. Section 10.5 concludes this chapter with results and a discussion.

Notation and Setup The input freeform surface is denoted $\mathcal{S}$ and the approximating Zome mesh $\mathcal{Z}$. As before, $\pi : \mathbb{R}^3 \mapsto \mathcal{S}$ is a projection operator for mapping 3D points

to their respective closest point on $\mathcal{S}$. $\pi(\mathbf{p})$ is the closest point on $\mathcal{S}$ to $\mathbf{p}$ and $\mathbf{n}_{\pi(\mathbf{p})}$ is the normal at $\pi(\mathbf{p})$ on $\mathcal{S}$. For practical reasons $\mathcal{S}$ is represented as a (high resolution) tri mesh. The vertices of the Zome mesh are referred to as nodes. Let $D = \{0, \cdots, 61\}$ denote the set of the 62 outgoing directions (or *slots*) of a Zometool node and Z the set of all possible polygons in the Zometool system. In the here presented approach $\mathcal{Z}$ is built from a restricted set of triangular and planar, convex quadrilateral panels denoted $P \subset Z$. While using planar panels can be motivated by their architectural and fabricational advantages, the use of mixed panels can be motivated by mutual practical considerations. E.g., while triangles generally allow for closer approximations, there are local configurations on the front which cannot be closed by any triangular panel in P but require the use of quads. The inset figure shows a configuration in a Zometool plane of type 4 (cf. Section 8.1) which only contains quadrilaterals, and there hence only exists one planar panel (quad) to close it. Depending on the context, s and (d, l) (a pair of direction and length) will be used synonymously to refer to a strut. Bold notation is used to refer to 3D coordinates and vectors: E.g., $\mathbf{v}$ represents the 3D position of a node v.



10.2. Zometool Front Growing

After placing an initial panel, $\mathcal{Z}$ is grown by incrementally adding panels $p \in P$ to the struts on the boundary $\partial\mathcal{Z}$ of the current Zome mesh. To avoid front self-collisions, the order of growing is controlled by a harmonic field of arrival times, while the actual grow-operation performed depends on the local shape of the front. Panel approximation energies are used to evaluate the quality of different grow-operations. 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 panels compatible with the current strut s. These tasks are detailed in the following subsections and the structure of the method is illustrated in the block diagram in Figure 10.1.

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

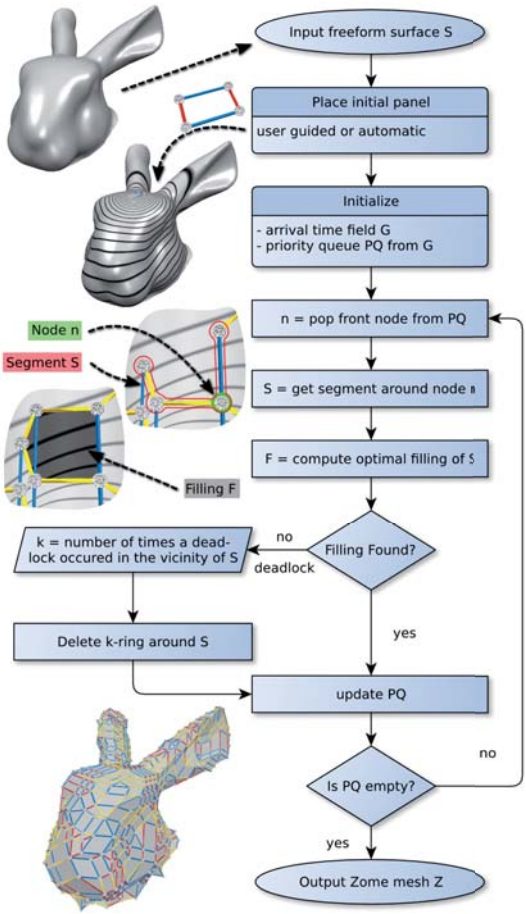


Figure 10.1.: Block diagram of the Zome mesh growing algorithm.

points. The field 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

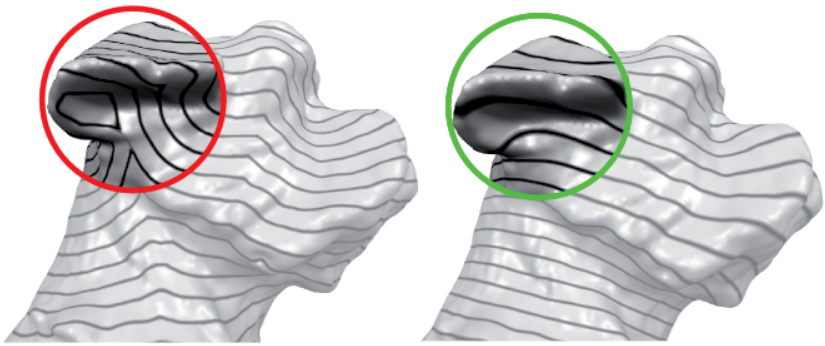


Figure 10.2.: Growing along a geodesic field (left) can lead to self-intersections of the front, whereas an harmonic field (right) is free of critical points.

the front, as this field is generally not free of critical points. Figure 10.2 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 lowest arrival time shall be popped from the queue and grown next. Note that the graph-Laplacian is guaranteed to always yield a field free of degeneracies. This field 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, regularly remeshed tri mesh, where both are acceptable choices and the graph-Laplacian can safely be chosen for simplicity.

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 10.3).

We define the *harmonic center* of $\mathcal{S}$ as the middle vertex of the harmonic disc parametrization of $\mathcal{S}$. Using this position as a starting point tends to produce harmonic fields G with more concentric iso-contours, cf. also experiment in Section 10.4. However, in coarse panelings 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., HOMER's nose was manually specified as the starting point in Figure 10.15.

Segment Filling Strategy

While the arrival time dictates *where* along the front 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 the method differentiates between *convex* and *concave* segments on the front (cf. Figure 10.3). 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/filled by the growing. 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 of 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. For this reason a [Lie03]-inspired filling strategy is adapted to *optimally* fill whole connected segments on the front in one step. The same strategy is applied to filling concave and convex segments, but, since the concave areas are the more problematic ones, these are prioritized and handled first. The strategy and the computation of the optimal filling is described in more detail below.

Convexity on the input surface is measured by first projecting the front $\partial\,\mathcal{Z}$ onto $\mathcal{S}$. For a node $v \in \partial\,\mathcal{Z}$ let $A(v) \in [0, 2\pi]$ be the *front angle* or *convexity* of v, defined as the angle between the two vectors spanned by the position of v and its boundary neighbors w and u when projected onto the tangent plane of $\mathcal{S}$ at $\pi(\mathbf{v})$. We call v *convex* if $A(v) > 180°$. 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 neighboring nodes are convex.

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

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

Computing the Optimal Segment Filling

First, a *valid filling* of a segment S is defined as a set of panels $\mathcal{F} \subset P$ that covers the inner nodes of the segment without intersecting other parts of $\mathcal{Z}$ or changing the topology. A covered node is defined to have a complete 1-ring (i.e., does no longer lie on the boundary). Figure 10.4 shows a valid filling (left) and an invalid filling (right) which neither covers the vertices nor leaves the topology intact. Assuming the existence 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 $\texttt{Cost}(\mathcal{F}) = \sum_{p \in \mathcal{F}} \texttt{Cost}(p, \mathcal{S})$.

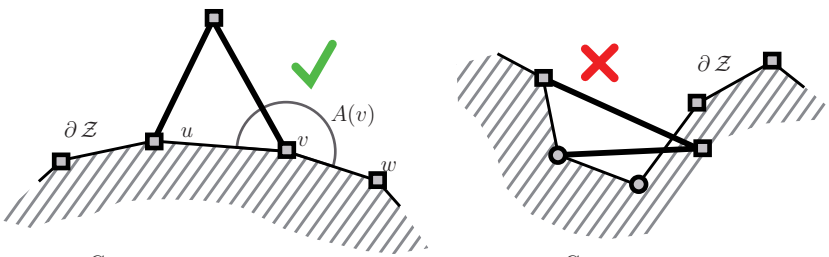


Figure 10.3.: Care needs to be taken when advancing the front in concave segments. Here, squares denote convex nodes and circles concave ones. The nodes u and v are *purely* convex. A concave segment is a connected set of struts with only concave inner nodes.

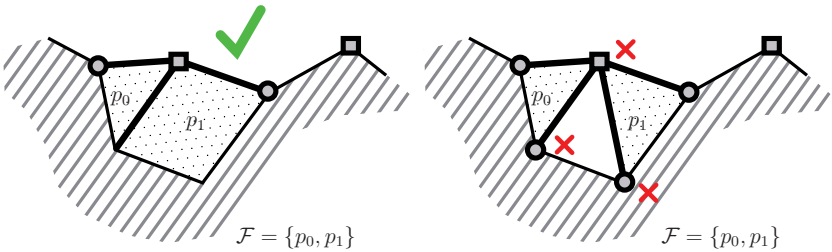


Figure 10.4.: Two fillings of the concave segment from Figure 10.3. Note that after the filling, some convex front nodes can turn concave.

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 [Lie03] 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 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 manner from left to right. During the filling, the segment S (whose inner nodes are to be covered) is modified as growing progresses and when S is empty and the filling is valid it is complete. 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 (the direction is inherently provided by the mesh orientation along the boundary). Growing is now restricted to only add a panel onto the active strut in each iteration. After a panel p has been added, the active strut (and other struts $s \in S$ now covered by p) is removed from S and the new strut that points to the next un-covered node is added to S (handled by the function `UpdateSegment`) and gets activated. Note that the next un-covered node is the same as the previous one if the added panel did not complete the corresponding 1-ring. Figure 10.5 demonstrates the filling process and how the segment and the active strut are updated. Pseudo-code for the `OptFill` function for computing an optimal filling of a segment is detailed in Algorithm 10.1. A useful heuristic in practice is the sorting of

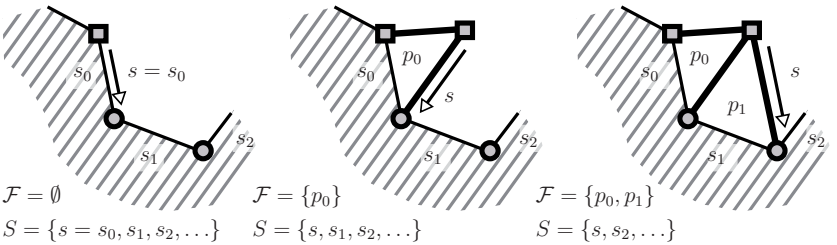
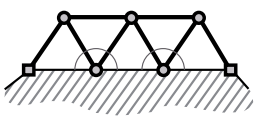


Figure 10.5.: A "gap" on the front to be filled is defined by a set of directed struts S, called segment. The optimal filling $\mathcal{F}$ is computed in a structured manner from left to right, in each iteration a panel is appended to the current *active strut* s (denoted by an arrow). The three subfigures demonstrate the filling process. Each added panel modifies the segment S (and the active strut), when S is empty the gap is filled.

the panels in the look-up table `Panels`(s) in descending order of approximation error (or cost). Based on the observation that panels with high costs (having 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, it enables reaching the optimal filling faster in general.

Continuing the iterative growing process only along the active strut localizes the filling, as it always remains directly connected to the segment to be filled. Still, to guarantee termination (2) the maximal recursion depth (*maxdepth*) must somehow be restricted. This is done by first limiting the concavity of the segment S to be filled and then derive an upper bound on *maxdepth* from that. The cardinality of S is restricted implicitly by limiting the convexity sum of its inner nodes $\sum_{v \in S} A(v) < 360°$. I.e., starting from a node popped from the front of the growing field, S is grown by adding struts in both directions until this bound is reached. Now, as a heuristic to correspondingly bound *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 an monotonically increasing cost function and the above structured growing strategy. Partial fillings $\mathcal{F}$ having a greater energy than the

Algorithm 10.1: 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 ← ∅, bestCost ← ∞, minGS ← min_{v∈S}(G(π(v)))

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

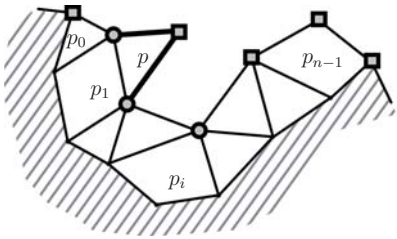
currently best valid filling $\texttt{Cost}(\mathcal{F}) > bestCost$ can be pruned, as can fillings containing panels that intersect $\mathcal{Z}$. Furthermore, unnecessary energy evaluations and intersection tests can be avoided for "inward" pointing panels by pruning these away based on their nodes' arrival time values. The panels in $\texttt{Panels}(s)$ of 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 grown part of $\mathcal{Z}$. We utilize the growing field G to quickly discard panels having nodes v with smaller arrival time than the current minimum of the segment, i.e., $G(\pi(\mathbf{v})) \leq \min_{u \in S} G(\pi(\mathbf{u}))$.

The intersection handling in explained next.

Handling Filling Intersections

The harmonic growing strategy was devised to avoid two distant parts of the front bumping into 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 strip around the segment to be filled. The thickness of the strip 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 exemplarily shows a 1-panel thick strip of panels $\{p_i\}$ around the nodes of a concave segment where a new panel p has been added.



Locally around the incident nodes of p the discrete set of 62 directions in D can be utilized to perform very efficient and numerically stable intersection tests between p and the 1-ring neighboring panels 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 coinciding with existing nodes on the front. Two neighboring panels sharing a node v are considered intersecting (cf. Figure 10.6) 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_0 and p_1 with a common node v (cf. Figure 10.6) 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_0 at v and d_2, d_3 corresponding to the struts of panel p_1 at v.

Now, general polygon intersection tests only need to be performed between p and non-neighboring panels p_i. The efficient `tri_tri_intersect` test by Möller [Möl97] is

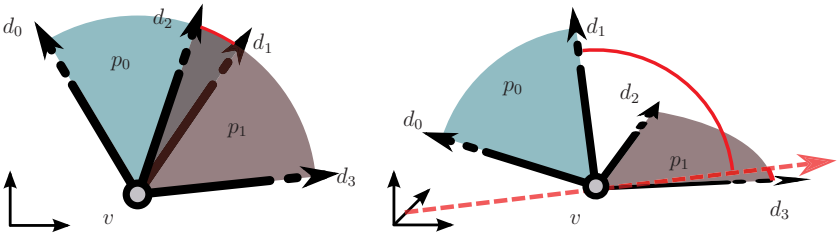


Figure 10.6.: The two *node*-intersection cases between two panels p_0 and p_1 incident to a node v. For co-planar panels it is enough to check if a strut of one panel is contained in the angle span of the other. If they are not co-planar, the intersection axis (red, dashed) is checked for containment in both angle spans. Here, the left configuration intersects while the right doesn't.

used for panels in general position, while for co-planar panels CGAL [CGAL] 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 and depending on the implementation the new struts of p also need to be tested against all other struts in the strip.

Panel Cost Function

The cost of a 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 N samples $\{\mathbf{p}_i\}$ on the panel p. The closeness energy measures the distance between the samples and their projections onto $\mathcal{S}$:

$$E_{\text{close}}(p, \mathcal{S}) = \tfrac{1}{N \cdot b_0^2} \sum_{i=1}^{N} (\pi(\mathbf{p}_i) - \mathbf{p}_i)^2,$$

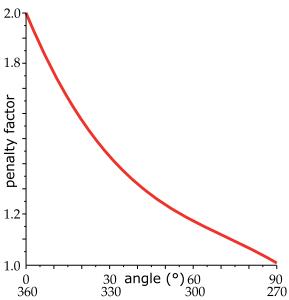
where the normalization by b_0 keeps the energy within the range $[0, 1]$ for distances less than b_0. The orientation energy measures the deviation between the normal $\mathbf{n}$ of the panel p and the normals at the projections of the samples:

$$E_{\text{orient}}(p, \mathcal{S}) = \tfrac{1}{4N} \sum_{i=1}^{N} (\mathbf{n}_{\pi(\mathbf{p}_i)} - \mathbf{n})^2,$$

where the normalization by 4 keeps the energy in the range $[0, 1]$. For combining the energies $\alpha = \frac{2}{3}$ is used in all experiments, this choice is motivated in Section 10.4.

Penalizing Pointy Fillings

Even though the growing field G prescribes a continuous, interference-free front, the discrete panels cannot perfectly adhere to this paradigm. The optimal filling of a segment can include pointy panels that unnecessarily intrude into neighboring areas or thin slits later only fillable by a corresponding pointy parallelogram. We consider angles below 90° and above 270° increasingly problematic. To keep the front of a filling $\mathcal{F}$ as compact as possible, we penalize $\texttt{Cost}(\mathcal{F})$ depending on the minimal and maximal front angle value according to the function visualized in the inset, i.e., if all angles on the front of the new filling are greater than 90° and less than 270° the cost is multiplied by 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 lesser panels are necessary. This is why we choose to just penalize them in favor of another solution where one is available.



Panel Look-Up Table

The set of all admissible triangular and (planar) quadrilateral panels is pre-computed and stored in a table: $\texttt{Panels}(s) \subset P$ parametrized over the struts s (or equivalently over the directions $d \in D$ and corresponding length types $l \in \{0, 1, 2\}$). Each pair (d, l) returns a list of all panels having an edge parallel to the direction of d and the corresponding 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 each direction $d \in D$ has about 40 triangles

and 370 quads, meaning there is a higher availability of quad panels when filling a segment. One notes that some of these panels must be geometrically identical, since P only consists of 29 different triangles and 118 different planar, convex quads in total.

Implementation Details

As already mentioned in Chapter 8, both $\mathcal{S}$ and $\mathcal{Z}$ are represented by halfedge-based mesh data structures. 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. Furthermore, as in the previous chapter, the projection operator $\pi(\mathbf{v})$ can be efficiently implemented using a BSP search structure supported by a spatial-hashing crust around the surface to cache already computed projections. In fact, for evaluating fillings in the present method, where a potentially huge number of projections is carried out from similar or identical sample positions, the spatial-hashing reduced projection times by as much as 2 orders of magnitude (depending on the model) compared to using BSP only.

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 now might be no valid, intersection-free filling for a neighboring segment – a deadlock, cf. Figure 10.7. 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 arises at a similar location again, 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 fix deadlocks. As indicated by Table 10.1 such fixes are rarely needed. To keep track of the number of times problems

have arisen, the corresponding areas in space can be tagged using the spatial hashing mentioned above.

In case growing can in fact continue without causing a deadlock, still unwanted "fold-over" configurations typically 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, this is one interesting problem to address in future work (cf. also Conclusion).

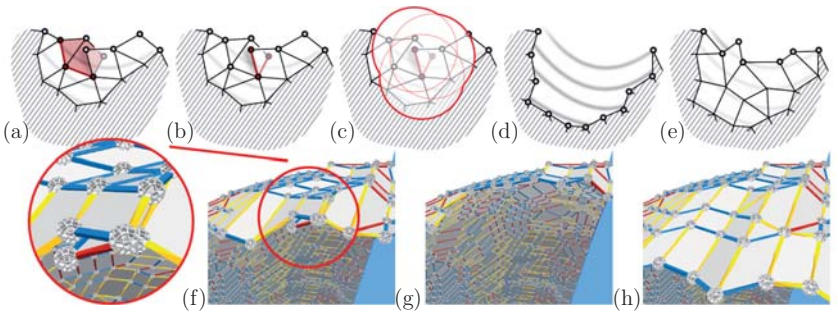


Figure 10.7.: (a) The filling of the segment (red) has grown underneath a neighboring part of $\mathcal{Z}$. The neighboring segment in (b,f) cannot be filled. This deadlock is removed by deleting a neighborhood of panels (c,d,g) and letting the standard growing re-fill the gap (e,h).

10.3. 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 $\Sigma = (\mathbf{x}, \mathbf{n})$, where $\mathbf{x}$ is a point on the plane and $\mathbf{n}$ 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.

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 8.2) is also properly aligned with Σ. Planes of the first three types are guaranteed symmetric, as each of them is orthogonal to a node slot or direction $d \in D$, for which there always exists a slot $\bar{d}$ pointing in the opposite direction. 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 10.8. Let $D|_\Sigma$ denote the set of directions lying in Σ, these are highlighted in green in the right-most subfigure. Motivated by the symmetry experiment in Section 10.4, 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), one can also rotate the node in the plane (around $\mathbf{n}$) to align its directions $D|_\Sigma$ as well as possible with these directions.

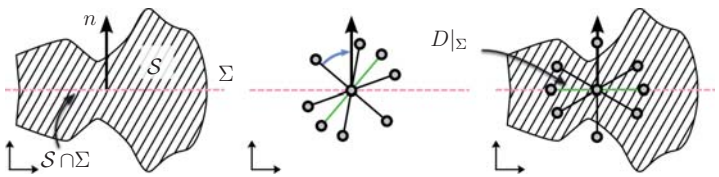


Figure 10.8.: For an exemplary 2D configuration (middle) the default global node orientation (denoted by a node with outgoing struts) is not reflective symmetric w.r.t. Σ (pink dashed), hence symmetric meshing is not possible. Rotating the global node orientation as indicated by the blue arrow aligns one of its symmetry planes with Σ and enables symmetric results by mirroring.

Initialization

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

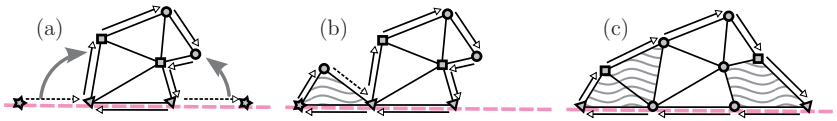


Figure 10.9.: Growing along a symmetry plane (pink dashed) requires special attention as all symmetry nodes lie on the boundary. The boundary orientation is denoted by arrows. (a) shows two possible active struts (dashed arrows) to/from the two "outer" symmetry nodes (triangular) from/to potential new nodes (star-shaped) respectively. The gray arrows denote the oriented growing direction used to close the respective gaps, this is basically the rotation the active strut undergoes as panels are added. In (b) a panel has been added to the left active strut, which is subsequently updated to still point to its triangular node. Similarly the active strut would "rotate" around the right triangular node when filling the right gap. A gap is considered filled when the ring growing around a triangular node connects to the corresponding square node. In (c) both the left and right gap from (a) have been filled.

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 $\mathcal{Z}$.

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 10.2 above relies on the halfedge data structure of $\mathcal{Z}$ and is based on covering boundary vertices to no longer lie on the boundary $\partial\,\mathcal{Z}$. Symmetry nodes always lie on $\partial\,\mathcal{Z}$ and call for a slightly modified approach. This is demonstrated in Figure 10.9. The first step, similar to adding the initial strut s above, is to add the best fitting strut to the popped symmetry node, this is the *active strut*. (This temporary configuration is not 2-manifold.) Now the only difference to the filling procedure from above is how the active strut is updated and when the gap is considered closed. The active strut is always connected to the popped symmetry node, and depending on which node was popped (the "leftmost" or "rightmost") it is rotated clockwise or counter clockwise by an added panel. The gap is closed when the popped node is no longer on the boundary in that halfspace, i.e., when

the half-ring of growing panels connects to its nearest non-symmetry neighbor node on $\partial\,\mathcal{Z}$.

10.4. Experiments

This section summarizes three experiments which were performed to arrive at appropriate choices for various method parameters.

Relative Weighting of Energies

An appropriate choice of coefficient for the linear combination of energy functionals was obtained experimentally by analyzing the effects of the individual energies (closeness and orientation) for different coefficients $\alpha \in [0..1]$ on a hemisphere object (cf. Figure 10.10). Naturally, the orientation error decreases with increasing α, while the closeness error increases. 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. The 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.

Starting Position

To determine an appropriate starting position for the growing procedure we experimented with starting positions distributed at different distances from the boundary of an input shape. Figure 10.11 shows the CHILD object partitioned in colored strips, each with a different distance from the boundary. In each of these strips growing was initialized at a number of random positions and the *time* and number of *fixes* was measured until growing was completed. The graph on the right in Figure 10.11 shows the average time and the total number of fixes per run. While also low timings and a low number of fixes are possible when starting close to the boundary the probability and stability of good solutions increases closer to the top (or "middle") of the object. Based on these observations, the use of the above mentioned *harmonic center* as starting position is proposed. The harmonic center is a natural "middle position", which enables low stretch of the growing field and a more geodesic growing.

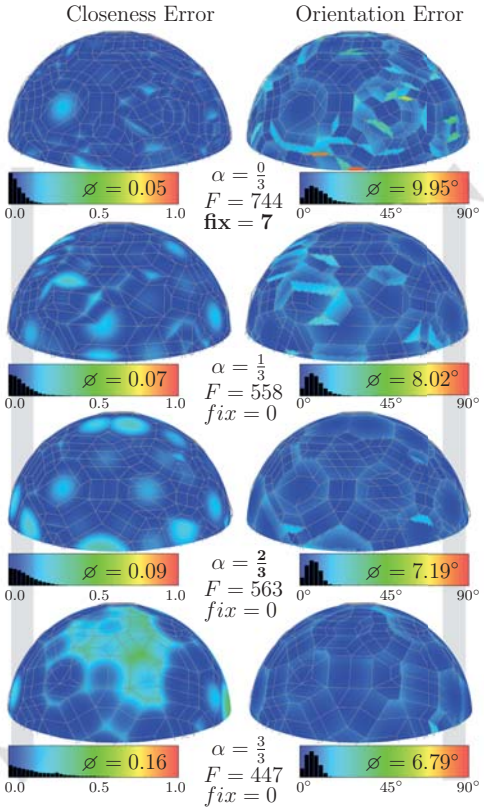


Figure 10.10.: Experiment: linear combination of energies on a hemisphere mesh. The errors are color-coded on the input freeform surface on a scale from dark blue (lowest) over green to red (highest). Here, the orientation error is the angle deviation. The Zometool wireframe is shown in gray. F is the number of faces in the resulting $\mathcal{Z}$ and fix is the number of deadlock repairs necessary.

Symmetry Plane

When meshing with symmetry constraints a Zometool plane of type 1 is used 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

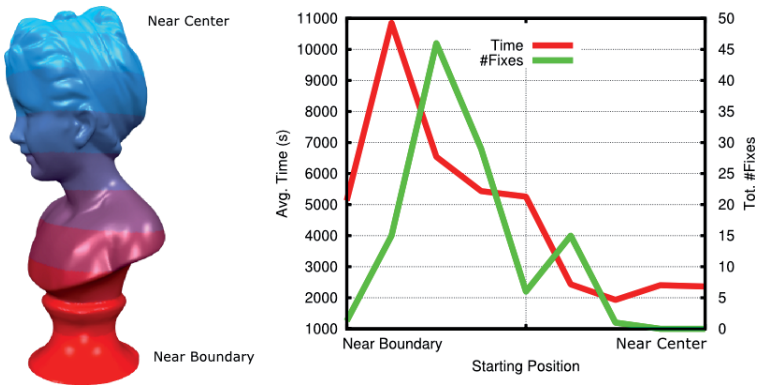


Figure 10.11.: Experiment: Starting position of the growing procedure. The lowest and most stable distribution of computation times and required number of fixes is achieved close to the center (in this case "top") of an object.

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 at the plane and be propagated further. Furthermore, a high resolution plane also leads to lower computation times as pruning is more effective when low-energy solutions can be found. Figure 10.12 shows meshing results using planes of type 1, 2 and 3 on the MOAI mask (cf. Table 10.1 for run-times).

10.5. Evaluation

In the following, the proposed Zometool paneling method is evaluated on the basis of results computed from different freeform surfaces. Computations were performed on a standard i7 PC using OpenMP parallelization. The results are collected in Table 10.1 and the therein bold-marked objects are shown in Figure 10.13. The "Ground Truth" row in the table specifies the quad-to-triangle ratio of panels in P and the relative frequencies of *r*ed, *b*lue and *y*ellow slots in the Zometool node. A mock-up architectural-style rendering of the TRAINSTATION is shown in Figure 10.14.

Input to the paneling method are disk topology meshes, hence, if closed, the input surface needs to be cut open correspondingly. On standing objects (e.g. CHILD) it is natural to cut open the bottom, while for "masks" to be hung against a wall the back

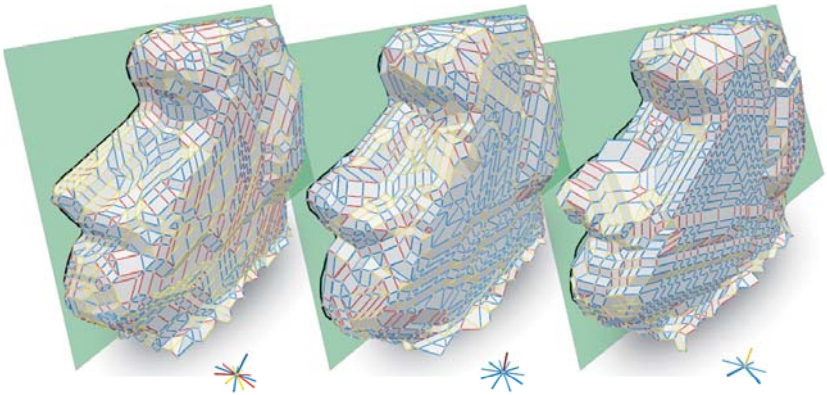


Figure 10.12.: Experiment: MOAI mask meshed using symmetry planes of type 1, 2 and 3. A low availability of directional resolution (node slots) in the plane leads to bad a approximation of the intersection curve $\mathcal{S} \cap \Sigma$ (black) and a propagation of errors into the interior of $\mathcal{Z}$. High energy values also cause longer computation times due to pruning being less effective (cf. Table 10.1).

could be opened (e.g., SUZANNE). Note that while an infinitesimal hole suffices in theory, a heavily distorted growing field is more prone to cause problems on the front. In general, "sufficiently smooth" input surfaces and a target panel-size corresponding reasonably well to the feature granularity of the input are assumed. However, HOMER and WUFFI show that also very coarse solutions as well as feature-rich inputs are possible. The output meshes are in general quad-dominant mesh 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.

While the single steps of the paneling method are 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.

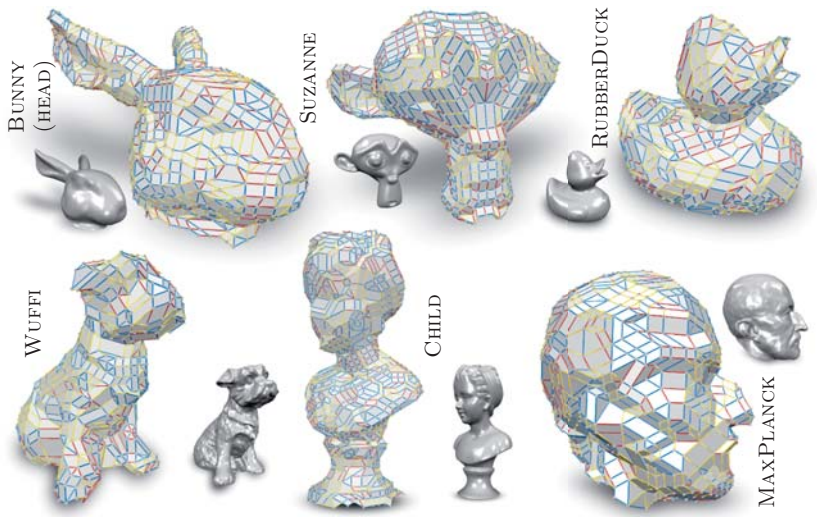


Figure 10.13.: Planar Zometool panelizations of different input surfaces. Cf. Section 10.5 and Table 10.1 for details. The meshes are courtesy of the AIM@SHAPE shape repository (`http://shapes.aim-at-shape.net`), Blender (`http://www.blender.org`) and the Chair for Computer Graphics and Multimedia (`http://www.rwth-graphics.de`).

When filling a segment from "left" to "right" along the active strut(s), 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 for occasional optimal solutions requiring panels to the left with high energy values.

Node Orientation Although the global orientation of the Zometool node potentially has an influence on the resulting $\mathcal{Z}$, the relative frequencies $\%\{r, b, y\}$ in Table 10.1 are generally (except for the symmetry plane experiments on Moai) closely distributed around the "Ground Truth" values. 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.

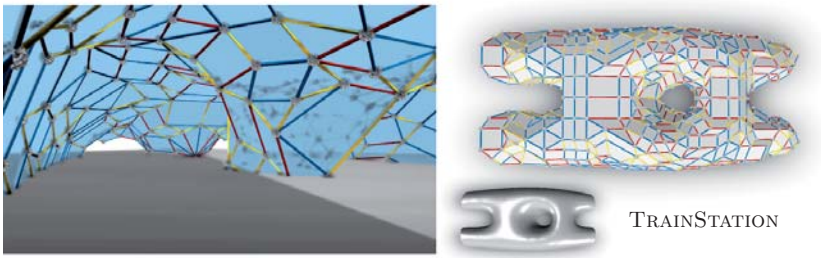


Figure 10.14.: Mock-up glass-panel rendering of an architectural design. The TrainStation is courtesy of Evolute GmbH (`http://www.evolute.at`).

Re-Usability and Scaling Zometool enables a huge re-usability potential as only 9 different edges are involved. Considering the general applicability, even the 147 geometrically different panels used by the paneling method presented here are justifiable since they can be re-used for all computed panelizations. However, the issue of fixed scaling (sizes of elements) might call for sets of differently sized panels depending on the scale of the application at hand, e.g., building-sized architecture vs. toy-sized RubberDucks.

Features Given the finite set of angles present in the Zometool system, the sharp features common on 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 10.15 on a series of Homers at different scales. Naturally, the preservation of small features cannot be guaranteed in practice, even if they are theoretically representable, as capturing them would require the growing process to place nodes precisely on the particular feature, which is hard to achieve in general due to the arbitrary distribution of features, fixed sizes of elements, and the nature of the growing and starting position.

Outlook

The triangles and convex (planar) quads used in this work corresponded well to the harmonic growing strategy as well as enabled intuitive intersection tests. However, with a limited set of directions (and panels) automatically comes a limited fairness of the resulting Zome meshes. More detailed evaluations of the obtainable fairness would

Model	$\lvert\mathcal{Z}\rvert$	▱/△	%r	%b	%y	time	mode	fix
"Ground Truth"	–	4.1	20	48	32	–	–	–
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
RubberDuck	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/S1**	**0**
Wuffi	**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 10.1.: 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 (*U*ser-defined or *A*utomatic) and growing mode (*F*ree or *S*ymmetric). The x in Sx denotes the used plane type. "fix" is the number of deadlock repairs that were necessary.

be required for architectural applications. Extending the set of panels by introducing further planar, convex n-gons is theoretically straightforward, but would imply a heavy increase of computational costs due to the exponentially growing branching possibilities.

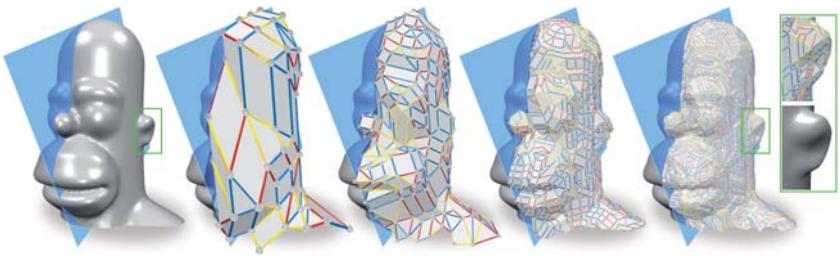


Figure 10.15.: Different scale quad-dominant Zometool meshings of a HOMER mask (back of the head open), having $0.1k$, $0.5k$, $1.5k$ and $6k$ faces.

A generalization to concave panels could be possible but would require extra care in the priorization. It is unclear how to efficiently handle the intersection tests for non-planar panels.

Extending the presented (surface) growing approach to non-disk topologies is problematic due to the problem of qualitatively joining fronts. Growing a polyhedral (volume) mesh, from which a Zome (surface) mesh could then be extracted, in a similar fashion would be theoretically possible, but again the exponential growth of the number of panels and branching possibilities pose hard constraints in practice.

In areas with features, concavities, peaks or high distortion of the growing field (e.g., back of SUZANNE's chin) panels on the front of the growing can intrude too much into neighboring regions and eventually cause ugly fold-over configurations. Presently, these configurations are only "fixed" when they lead to deadlocks. The fix then consists of deleting and regrowing a conservative area surrounding the deadlock. By further investigating the topological region of influence of nodes located in the interior of a mesh, it could be possible to develop a remeshing operator based modifying only a localized neighborhood of minimal size. 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.

11. Conclusion

This thesis presented a set of novel methods for dealing with geometric problems relating to different tasks of the overall goal of enabling efficient realizations of freeform structures. The following sections conclude this work by summarizing the main contributions and presenting an outlook over interesting directions for future research on these topics.

Summary

Architectural Geometry is both a young and active field of research, constantly extended by new geometric challenges as related sciences such as materials research, production, and architecture advance. The rationalization of freeform point-folded structures, with elements produced by incremental sheet forming (ISF), which was a topic of Part II, is one such example. Interesting problems were also shown to arise when exploring generalizations of the scope of possible applications of existing construction systems, originally designed for different purposes, as was done in Part III for the Zometool system. However, also simply taking a different view on a classical problem, such as polygon mesh planarization, as done in Part I, can result in novel optimization formulation useful for a range of meshing scenarios.

By a variational re-formulation, in Part I we equipped the concept of tangent plane intersections (TPI) with additional degrees of freedom and presented a robust, unified approach for creating polygonal structures with planar faces that is readily able to integrate various objectives and constraints needed in different application scenarios. Besides stabilizing the computation of hex-dominant meshes with planar faces we demonstrated the abilities of the approach on two further important classes of problems: First, for the task of general polygon mesh planarization, the inherent planarity given by the TPI formulation avoids the (unknown number of) necessary re-weightings and re-optimizations involved when working with penalty based techniques, and for comparable levels of planarity the run-times could be considered on par with compared methods. Second, by

using an appropriate set of constraints we enabled efficient computation of a special class of intersection-free, double-layer support structures.

Part II further dealt with the anti-diversification of dies for the production of point-folded elements. We analyzed the rationalization possibilities inherent in the design of point-folded structures and formalized the rationalization problem accordingly. By a geometric interpretation of given hard-constraints, controlling the degrees of freedom of the process reduced to defining appropriate so-called validity volumes. Then, for solving the rationalization task of efficiently covering tri mesh designs with triangle-based point-folded elements, these volumes were mapped into an angle space, where similar dies map to nearby positions, and a memory efficient, high accuracy, greedy set cover algorithm was presented to extract a minimal set of such different base shapes. Rationalization gains of around 90% proved realistic and achievable without altering the geometry of the given triangle mesh and significant reductions of fabrication costs might thus be attained in practice.

In the last part, Part III, a different approach was taken to anti-diversification of paneled freeform structures: Given a pre-defined set of construction elements, here based on the Zometool system, for two different use-cases algorithms were developed for tessellating freeform designs using only these elements. For the task of approximating closed freeform surfaces of arbitrary genus, an efficient, stochastic algorithm was developed that yielded high-quality results quickly and was further able to respect even very delicate surface features. The efficiency of the method was enabled by a carefully designed set of modification operators, which could be applied in a parallel fashion to update a computed initial approximation. For paneling the architecturally important class of freeform surface patches, a second method was developed, which was further restricted to only use a small set of planar, convex panels. This method proceeds by growing the new tessellation from scratch over the surface.

Outlook

The methods developed in this thesis largely deal with new topics for which they present the very first approaches in their respective fields, e.g., the anti-diversification of triangle-based point-folded elements and the Zometool panelization of freeform shapes, while the variational formulation of tangent plane intersections offers a new view on the classical problem of mesh planarization. The respective fields are still far from exhausted and

the presented approaches have even give rise to a new set of interesting questions and problems to be explored in future work. In the following potential possibilities and remaining problems of the presented methods are discussed and summarized.

While using an optimized, blackbox solver for the optimization problem eases implementation in Part I and possibly increases numerical stability, it also disables direct control over certain parts of the optimization process. Such control could be important to obtain high quality results in some applications, e.g., where the preferred positions of the closeness energy are not fixed but rather allowed to evolve during optimization, i.e., move along the input surface. The optimization of dual-layer support structures dealt with panel planarity and avoiding inter-layer intersection on an infinitesimal level. However, for real-life realizations of such structures also detailed modeling and optimization of real-sized nodes and beams is necessary.

The anti-diversification of point-folded structures presented in Part II is restricted to triangle-based elements and thus only presents a first step towards enabling a complete and efficient building system based on point-folded structures. We believe two important parts of future work to be (a) generalizing the technique to elements based on other types of polygons and (b) introducing more control over the dual, second layer. Already mentioned in Chapter 7 is that higher dimensional search spaces would rapidly reduce the possible accuracy due to increased memory consumption. One idea to alleviate this problem could be to trade space complexity for time by using higher order elements, i.e., instead of piecewise linear prisms using, e.g., piecewise quadratic ones. Initial experiments have shown that this efficiently increases accuracy of the representation, necessitating much fewer refinement levels, while at the same time increasing the complexity of the intersection tests. Additionally, problems relating to the representation of higher-dimensional parachutes and the mapping from angle space need to be solved. The anti-diversification method presented was entirely discrete. Further integrating continuous optimization goals in the process, such as planarity of the faces of the second layer, calls for a simultaneous optimization for continuous and discrete goals, possibly necessitating completely new approaches. In this regard, also allowing modification of not only the geometry, but also the connectivity of both layers might be necessary to enable enough degrees of freedom for successful optimization.

The Zome meshes resulting from the paneling methods presented in Part III had different qualities. Especially for practical applications where aesthetics play a central role, fairness can be an important quality. The fairness is naturally limited by the 62 directions of the system, but can be influenced by the additional restrictions on the

types of used panels, e.g., planar panels or panels with a limited polygonal degree. In general, the method in Chapter 9 enables higher (node) fairness as it (a) specifically utilizes a fairness energy functional and (b) is optimized in a global fashion. While the front growing method in Chapter 10 was restricted to using energy functionals relying only on single mesh entities, due to it being based on a monotonically increasing energy. Integrating a fairness energy functional is not directly possible in this method, since evaluating fairness of, e.g., a node, requires a complete 1-ring, which is only available after a whole filling has been completed. Although a simple heuristic could be employed to penalize fairness after completing a filling, similarly to how pointy fillings are handled, a better founded solution should include global optimization with fairness persistent throughout the process. As discussed, in most cases it would be too restrictive to simply restrict the method in Chapter 9 to planar panels. However, additionally allowing a set of non-planar, bilinear panels could provide sufficient degrees of freedom for the optimization and increase both smoothness and approximation power, while still offering well-defined panel-panel intersection tests. Another interesting area of research is to enable direct form-finding based on the Zometool system instead of the current two step procedure where Zome meshes are fitted onto existing freeform designs. We envision a setting similar to Laplacian-based mesh editing, where freeform modeling is performed by transforming a handle, possibly re-tessellating the mesh, and deforming it to best integrate the new geometry of the handle region.

Finally, this work dealt with geometric aspects only, further integrating physics and structural constraints into the optimization can be relevant for real-life applications concerning all discussed techniques.

Bibliography

[AM02] Akenine-Möller T.: Fast 3d triangle-box overlap testing. *J. Graph. Tools 6*, 1, Jan. 2002, 29–33.

[Ame08] Ames J.: *Systematische Untersuchung der Beeinflussung des Werkstoffflusses bei der Inkrementellen Blechumformung mit CNC-Werkzeugmaschinen.* PhD thesis, RWTH Aachen University, Aachen, NRW, Germany, 2008.

[AUGA08] Alliez P., Ucelli G., Gotsman C., Attene M.: Recent advances in remeshing of surfaces. In *Shape Analysis and Structuring, Mathematics and Visualization*, 2008, Springer.

[AW11] Alexa M., Wardetzky M.: Discrete Laplacians on general polygonal meshes. *ACM Trans. Graph. 30*, 4, 2011, 102:1–102:10.

[BDS*12] Bouaziz S., Deuss M., Schwartzburg Y., Weise T., Pauly M.: Shape-up: Shaping discrete geometry with projections. *Comp. Graph. Forum 31*, 5, Aug. 2012, 1657–1667.

[BK03] Bischoff S., Kobbelt L.: Sub-voxel topology control for level-set surfaces. *Computer Graphics Forum 22*, 2003.

[BK04] Botsch M., Kobbelt L.: A remeshing approach to multiresolution modeling. In *Proc. SGP '04*, 2004, pp. 185–192.

[BKP*10] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B.: *Polygon Mesh Processing.* AK Peters, 2010.

[Blo09] Block P.: *Thrust Network Analysis: Exploring Three-dimensional Equilibrium.* PhD thesis, Cambridge, MA, USA, May 2009.

[BLP*13] Bommes D., Lévy B., Pietroni N., Puppo E., Silva C., Tarini M., Zorin D.: Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 2013.

[BMR*99] Bernardini F., Mittleman J., Rushmeier H., Silva C., Taubin G., Member S.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5*, 1999, 349–359.

[BO07] Block P., Ochsendorf J.: Thrust network analysis: a new methodology for understanding three-dimensional equilibrium. In *Proceedings of the IASS Symposium 2007, Architectural engineering - Toward the future looking to the past*, Venice, Italy, 2007.

[Bom12] Bommes D.: *Quadrilateral Surface Mesh Generation for Animation and Simulation*. Selected topics in computer graphics. Shaker, 2012.

[BPK05] Bischoff S., Pavic D., Kobbelt L.: Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4, Oct. 2005, 1332–1352.

[BPK*11] Bo P., Pottmann H., Kilian M., Wang W., Wallner J.: Circular arc structures. *ACM Trans. Graph. 30*, 4, July 2011, 101:1–101:12.

[BS08] Bobenko A., Suris Y.: *Discrete Differential Geometry: Integrable Structure*. Graduate studies in mathematics. American Mathematical Society, 2008.

[BV04] Boyd S., Vandenberghe L.: *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[BW10] Buri H. U., Weinand Y.: Origami – geometry of folded plate structures. In *Structures & Architecture*, 2010.

[BYMW13] Bao F., Yan D.-M., Mitra N. J., Wonka P.: Generating and exploring good building layouts. *ACM Trans. Graph. 32*, 4, 2013.

[BZK09] Bommes D., Zimmer H., Kobbelt L.: Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3, July 2009, 77:1–77:10.

[BZK12] Bommes D., Zimmer H., Kobbelt L.: Practical Mixed-Integer Optimization for Geometry Processing. In *Curves and Surfaces*, Berlin, Heidelberg, 2012, Boissonnat J.-D., Chenin P., Cohen A., Gout C., Lyche T., Mazure M.-L., Schumaker L. L., (Eds.), vol. 6920 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 193–206.

[CGAL] Computational Geometry Algorithms Library:. `http://www.cgal.org`. [Online].

[Cox73] Coxeter H. S. M.: *Regular Polytopes*. Dover Publications, 1973.

[CSAD04] Cohen-Steiner D., Alliez P., Desbrun M.: Variational shape approximation. *ACM Trans. Graph. 23*, 3, 2004, 905–914.

[CW07] Cutler B., Whiting E.: Constrained planar remeshing for architecture. In *Proceedings of Graphics Interface 2007*, 2007, GI '07, ACM, pp. 11–18.

[D'A00] D'Azevedo E. F.: Are bilinear quadrilaterals better than linear triangles? *SIAM J. Sci. Comput. 22*, 1, Jan. 2000, 198–217.

[Dav07] Davis T.: The mathematics of zome. `http://geometer.org/mathcircles/zome.pdf`, 2007. [Online].

[DDSD03] Décoret X., Durand F., Sillion F. X., Dorsey J.: Billboard clouds for extreme model simplification. *ACM Trans. Graph. 22*, 3, July 2003, 689–696.

[DGAOD13] De Goes F., Alliez P., Owhadi H., Desbrun M.: On the Equilibrium of Simplicial Masonry Structures. *ACM Transactions on Graphics 32*, 4, July 2013.

[DPW11] Deng B., Pottmann H., Wallner J.: Functional webs for freeform architecture. *Comput. Graph. Forum 30*, 5, 2011, 1369–1378.

[Ede01] Edelsbrunner H.: *Geometry and Topology for Mesh Generation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2001.

[EKS*10] Eigensatz M., Kilian M., Schiftner A., Mitra N. J., Pottmann H., Pauly M.: Paneling architectural freeform surfaces. *ACM Trans. Graph. 29*, 4, July 2010, 45:1–45:10.

[FB81] Fischler M. A., Bolles R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6, 1981, 381395.

[FBG96] Frey P. J., Borouchaki H., George P.-L.: Delaunay tetrahedralization using an advancing-front approach. In *5th Int. Meshing Roundtable, Sandia Nat. Lab.*, 1996, pp. 31–46.

[Fei98] Feige U.: A threshold of ln n for approximating set cover. *J. ACM 45*, July 1998, 634–652.

[FLHCO10] Fu C.-W., Lai C.-F., He Y., Cohen-Or D.: K-set tilable surfaces. *ACM Trans. Graph. 29*, 4, July 2010, 44:1–44:6.

[GLM96] Gottschalk S., Lin M. C., Manocha D.: Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1996, SIGGRAPH '96, ACM, pp. 171–180.

[GSC*04] Glymph J., Shelden D., Ceccato C., Mussel J., Schober H.: A parametric strategy for free-form glass structures using quadrilateral planar facets. In *Automation in Construction*, 2004, no. 13, pp. 187–202.

[HJJ03] Henderson D., Jacobson S., Johnson A.: The theory and practice of simulated annealing. In *Handbook of Metaheuristics*, Glover F., Kochenberger G., (Eds.), vol. 57 of *International Series in Operations Research & Management Science*. Springer US, 2003, pp. 287–319.

[HJW02] Hirt G., Junk S., Wituski N.: Incremental Sheet Forming: Quality Evaluation and Process Simulation. In *7th International Conference of Technology of Plasticity*, 2002, vol. 1, pp. 925–930.

[HP00] Hart G. W., Picciotto H.: *Zome Geometry: Hands-on Learning with Zome Models*. Key Curriculum, Dec. 2000.

[HSL11] HSL: A collection of fortran codes for large scale scientific computation. `http://www.hsl.rl.ac.uk`, 2011. [Online].

[HT11] Herkrath R., Trautz M.: Starre Faltungen als Leichtbauprinzip im Bauwesen. *Bautechnik 88*, 2, 2011, 80–85.

[IGG01] Isenburg M., Gumhold S., Gotsman C.: Connectivity shapes. In *Proceedings of the Conference on Visualization '01*, Washington, DC, USA, 2001, VIS '01, IEEE Computer Society, pp. 135–142.

[JMH*05] Jeswiet J., Micari F., Hirt G., Bramley A., Duflou J., Allwood J.: Asymmetric single point incremental forming of sheet metal. *CIRP Annals - Manufacturing Technology 54*, 2, 2005, 88 – 114.

[KCY93] Kaufman A., Cohen D., Yagel R.: Volume graphics. *Computer 26*, 7, 1993, 51–64.

[KFC*08] Kilian M., Flöry S., Chen Z., Mitra N. J., Sheffer A., Pottmann" H.: Curved folding. *ACM Transactions on Graphics 27*, 3, 2008, #75, 1–9.

[KGV83] Kirkpatrick S., Gelatt C. D., Vecchi M. P.: Optimization by simulated annealing. *Science 220*, 4598, 1983, 671–680.

[KH01] Kreylos O., Hamann B.: On simulated annealing and the construction of linear spline approximations for scattered data. *IEEE TVCG 7*, 2001, 189–198.

[KK95] Karypis G., Kumar V.: *METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*. Tech. rep., 1995.

[KSAZ01] Knebel K., Sanchez-Alvarez J., Zimmermann S.: *The Structural Making of the Eden Domes*. Tech. rep., Mero GmbH & Co. KG, Würzburg, 2001.

[LDZPD10] Lafarge F., Descombes X., Zerubia J., Pierrot Deseilligny M.: Structural approach for building reconstruction from a single DSM. *IEEE Trans. on Pattern Analysis and Machine Intelligence 32*, 1, 2010, 135–147.

[LeM08] LeMasters R.: Advantages and versatility of steel truss bridges. In *Ohio Transportation Engineering Conference*, 2008.

[Lie03] Liepa P.: Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Aire-la-Ville, Switzerland, Switzerland, 2003, SGP '03, Eurographics Association, pp. 200–205.

[LPS*13] Liu Y., Pan H., Snyder J., Wang W., Guo B.: Computing self-supporting surfaces by regular triangulation. *ACM Trans. Graph. 32*, 4, July 2013, 92:1–92:10.

[LPW*06] Liu Y., Pottmann H., Wallner J., Yang Y.-L., Wang W.: Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graphics 25*, 3, 2006, 681–689. Proc. SIGGRAPH.

[LXW*11] Liu Y., Xu W., Wang J., Zhu L., Guo B., Chen F., Wang G.: General planar quadrilateral mesh design using conjugate direction field. *ACM Trans. Graph. 30*, 6, Dec. 2011, 140:1–140:10.

[LZKW10] Li Y., Zhang E., Kobayashi Y., Wonka P.: Editing operations for irregular vertices in triangle meshes. *ACM TOG 29*, December 2010, 153:1–153:12.

[Möl97] Möller T.: A fast triangle-triangle intersection test. *Journal of Graphics Tools 2*, 1997, 25–30.

[MRR*53] Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., Teller E.: Equation of state calculations by fast computing machines. *The Journal of Chemical Physics 21*, 6, 1953, 1087–1092.

[Nak13] Nakajima M.: Sumiaki ono, alexis andré, youngha chang, masayuki nakajima (member).

[NGH04] Ni X., Garland M., Hart J. C.: Fair morse functions for extracting the topological structure of a surface mesh. *ACM Trans. Graph. 23*, 3, Aug. 2004, 613–622.

[PBCW07] Pottmann H., Brell-Cokcan S., Wallner J.: Discrete surfaces for architectural design. In *Curves and Surface Design: Avignon 2006*, Chenin P., Lyche T., Schumaker L. L., (Eds.). Nashboro Press, 2007, pp. 213–234.

[PBSH13] Panozzo D., Block P., Sorkine-Hornung O.: Designing unreinforced masonry models. *ACM Trans. Graph. 32*, 4, July 2013, 91:1–91:12.

[PLW*07] Pottmann H., Liu Y., Wallner J., Bobenko A., Wang W.: Geometry of multi-layer freeform structures for architecture. *ACM Trans. Graphics 26*, 3, 2007. Proc. SIGGRAPH.

[PN89] Piano R., Nakamura T.: *Renzo Piano: building workshop, 1964-1988*. Architecture and Urbanism Extra Edition Series. A + U Pub. Co., 1989.

[PSB*08] Pottmann H., Schiftner A., Bo P., Schmiedhofer H., Wang W., Baldassini N., Wallner J.: Freeform surfaces from single curved panels. *ACM Trans. Graph. 27*, 3, Aug. 2008, 76:1–76:10.

[PW08] Pottmann H., Wallner J.: The focal geometry of circular and conical meshes. *Adv. Comp. Math 29*, 2008, 249–268.

[Sau70] Sauer R.: *Differenzengeometrie*. Springer, 1970.

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

[Sch03] Schober H.: Freeform Glass Structures. In *Glass Processing Days*, Tampere, Finland, 2003, pp. 46–50.

[SDG10] Schenk M., D. Guest S.: Origami folding: A structural engineering approach. In *Proc. 5OSME*, 2010, pp. 293–305.

[SHWP09] Schiftner A., Höbinger M., Wallner J., Pottmann H.: Packing circles and spheres on surfaces. *ACM Trans. Graph. 28*, 5, Dec. 2009, 139:1–139:8.

[SM98] Silva C. T., Mitchell J. S.: Greedy cuts: An advancing front terrain triangulation algorithm, 1998.

[SPC09] Silva L. F., Pamplona V. F., Comba J. L. D.: Legolizer: A real-time system for modeling and rendering lego representations of boundary models. In *Computer Graphics and Image Processing (SIBGRAPI), 2009 XXII Brazilian Symposium on*, 2009, IEEE, pp. 17–23.

[SS10] Singh M., Schaefer S.: Triangle surfaces with discrete equivalence classes. *ACM Trans. Graph. 29*, 4, July 2010, 46:1–46:7.

[SSAK] Stephan S., Sánchez-Alvarez J., Knebel K.: *Reticulated Structures on Free-Form Surfaces*. Tech. rep., Mero GmbH & Co. KG, Würzburg.

[SSS06] Schreiner J., Scheiclegger C., Silva C.: High-quality extraction of isosurfaces from regular and irregular grids. *IEEE TVCG 12*, 5, 2006, 1205–1212.

[TA11] Trautz M., Ayoubi M.: Das prinzip des faltens in architektur und ingenieurbau. *Bautechnik 88*, 2, 2011, 76–79.

[Tac06] Tachi T.: Simulation of rigid origami. In *Proc. 4OSME*, 2006.

[Tac09] Tachi T.: One-DOF cylindrical deployable structures with rigid quadrilateral panels. In *Proc. IASS*, 2009, pp. 2295–2305.

[Tac10a] Tachi T.: Geometric considerations for the design of rigid origami structures. In *Proc. IASS*, 2010, pp. 458–460.

[Tac10b] Tachi T.: Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 2, 2010, 298–311.

[TH09] Trautz M., Herkrath R.: The application of folded plate principles on spatial structures with regular, irregular and free-form geometries. In *Proc. IASS*, 2009, pp. 1019–1031.

[Tro08] Troche C.: Planar hexagonal meshes by tangent plane intersection. In *Advances in Architectural Geometry*, 2008.

[TSP13] Testuz R., Schwartzburg Y., Pauly M.: Automatic generation of constructable brick sculptures. In *Eurographics 2013-Short Papers*, 2013, The Eurographics Association, pp. 81–84.

[VHWP12] Vouga E., Höbinger M., Wallner J., Pottmann H.: Design of self-supporting surfaces. *ACM Trans. Graphics*, 2012. Proc. SIGGRAPH.

[Vor] Vorthmann S.: vZome `http://vzome.com`. [Online].

[WB06] Wächter A., Biegler L. T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming 106*, 2006, 25–57.

[WL10] Wang W., Liu Y.: A note on planar hexagonal meshes. In *Nonlinear Computational Geometry*, Emiris I. Z., Sottile F., Theobald T., (Eds.), vol. 151 of *The IMA Volumes in Mathematics and its Applications*. Springer New York, 2010, pp. 221–233.

[WLG03] Wagner M., Labsik U., Greiner G.: Repairing non-manifold triangle meshes using simulated annealing. In *4th Israel-Korea Bi-National Conference on Geometric Modeling and Computer Grahpics*, 2003, pp. 88–93.

[WLY*08] Wang W., Liu Y., Yan D., Chan B., Ling R., Sun F.: *Hexagonal Meshes with Planar Faces*. Tech. Rep. TR-2008-13, Department of Computer Science, The University of Hong Kong, 2008.

[WOD09] Whiting E., Ochsendorf J., Durand F.: Procedural modeling of structurally-sound masonry buildings. *ACM Trans. Graph. 28*, 5, Dec. 2009, 112:1–112:9.

[ZCBK12] Zimmer H., Campen M., Bommes D., Kobbelt L.: Rationalization of Triangle-Based Point-Folding Structures. *Computer Graphics Forum 31*, 2, 2012, 611–620.

[ZCHK13] Zimmer H., Campen M., Herkrath R., Kobbelt L.: Variational tangent plane intersection for planar polygonal meshing. In *Advances in Architectural Geometry 2012*, Hesselgren L., Sharma S., Wallner J., Baldassini N., Bompas P., Raynaud J., (Eds.). Springer Vienna, 2013, pp. 319–332.

[ZK14] Zimmer H., Kobbelt L.: Zometool rationalization of freeform surfaces. *IEEE Transactions on Visualization and Computer Graphics 99*, PrePrints, 2014, 1.

[ZLAK14] Zimmer H., Lafarge F., Alliez P., Kobbelt L.: Zometool shape approximation. *Graphical Models*, Preprint, 2014, –.

[Zom13] Zometool Inc.: Manual 2.3. `http://zometool.com/images/resources/Manual2.3web.pdf`, 2013. [Online].

[ZSW10] Zadravec M., Schiftner A., Wallner J.: Designing quad-dominant meshes with planar faces. *Computer Graphics Forum 29*, 5, 2010, 1671–1679.

Curriculum Vitae

Henrik Zimmer

Date of Birth	19.12.1980
Place of Birth	Väsby, Sweden
Citizenship	Swedish
E-Mail	zimmer@cs.rwth-aachen.de

Academic Education

Sep. 2008 – Dec. 2013	Doctoral Student at RWTH Aachen University, Computer Graphics Group Degree: Dr.rer.nat. Supervisor: Prof. Dr. Leif Kobbelt
Oct. 2003 – Aug. 2008	Computer Science Studies at RWTH Aachen University Degree: Dipl.-Inform.
Sep. 2001 – Jun. 2003	Computer Engineering Studies at LTH, Lund Institute of Technology, Lund, Sweden (Contd. RWTH Aachen)

Publications

Henrik Zimmer, Florent Lafarge, Pierre Alliez and Leif Kobbelt: *Zometool Shape Approximation*, Graphical Models (GMOD) 2014

Henrik Zimmer and Leif Kobbelt: *Zometool Rationalization of Freeform Surfaces*, IEEE Transactions on Visualization and Computer Graphics (TVCG) 2014

Henrik Zimmer, Marcel Campen and Leif Kobbelt: *Efficient Computation of Shortest Path-Concavity for 3D*, IEEE Computer Vision and Pattern Recognition (CVPR) 2013

Henrik Zimmer, Marcel Campen, Ralf Herkrath and Leif Kobbelt: *Variational Tangent Plane Intersection for Planar Polygonal Meshing*, Advances in Architectural Geometry (AAG) 2012

Henrik Zimmer, Marcel Campen, David Bommes and Leif Kobbelt: *Rationalization of Triangle-Based Point-Folding Structures*, Comp. Graph. Forum 31/Eurographics 2012

David Bommes, Henrik Zimmer and Leif Kobbelt: *Practical Mixed-Integer Optimization for Geometry Processing*, LNCS: Proceedings of Curves and Surfaces 2010

David Bommes, Henrik Zimmer and Leif Kobbelt: *Mixed-Integer Quadrangulation*, ACM Trans. Graph. 28/Siggraph 2009

N. Stache, H. Zimmer, J. Gedicke, B. Regaard, A. Olowinsky, A.Knepper and T. Aach: *Approaches for High-Speed Melt Pool Detection in Laser Welding Applications*, Vision, Modeling, and Visualization (VMV) 2006

Nicolaj C. Stache and Henrik Zimmer: *Robust Circle Fitting in Industrial Vision for Process Control of Laser Welding*, Proc. of the 11th International Student Conf. on Electrical Engineering POSTER 2007

Nicolaj Stache, Henrik Zimmer, Jens Gedicke, Alexander Olowinsky and Til Aach: *Robust High-Speed Melt Pool Measurements for Laser Welding with Sputter Detection Capability*, DAGM07: 29th Annual Symposium of the German Association for Pattern Recognition