Zometool Shape Approximation

Henrik Zimmer, Florent Lafarge, Pierre Alliez, Leif Kobbelt

Available online at www.sciencedirect.com
ScienceDirect
Graphical Models 00 (2014) 1–12
Graphical Models

Zometool Shape Approximation

Henrik Zimmer[a], Florent Lafarge[b], Pierre Alliez[b], Leif Kobbelt[a]

[a]*Computer Graphics Group, RWTH Aachen University, Germany*
[b]*Inria Sophia Antipolis - Méditerranée, France*

Abstract

We present an algorithm that approximates 2-manifold surfaces with Zometool models while preserving their topology. Zometool is a popular hands-on mathematical modeling system used in teaching, research and for recreational model assemblies at home. This construction system relies on a single node type with a small, fixed set of directions and only 9 different edge types in its basic form. While being naturally well suited for modeling symmetries, various polytopes or visualizing molecular structures, the inherent discreteness of the system poses difficult constraints on any algorithmic approach to support the modeling of freeform shapes. We contribute a set of local, topology preserving Zome mesh modification operators enabling the efficient exploration of the space of 2-manifold Zome models around a given input shape. Starting from a rough initial approximation, the operators are iteratively selected within a stochastic framework guided by an energy functional measuring the quality of the approximation. We demonstrate our approach on a number of designs and also describe parameters which are used to explore different complexities and enable coarse approximations.

Keywords: Zometool, Remeshing, Simulated Annealing, Topology Preservation

1. Introduction

Zometool is a tangible, mathematical modeling system not only used in various fields of science for research and teaching, but also recreationally for personal fabrication. The standard Zometool system is compact, structured and mathematically well-thought-out. It consists of 3 different edge types (or struts) each coming in 3 different lengths related by the golden ratio. The single node type has 62 different slots (or directions) based on 2, 3, and 5-fold symmetry axes derived from a icosahedron/dodecahedron. While being limited in the sense of having only a small, discrete set of available angles and edges, the Zometool system allows for a very rich set of structures.

A common Zometool application is as a teaching aid for hands-on visualization of geometric structures and symmetries. For the digital modeling and design of Zometool structures two software are available [1, 2], both allowing for simple point-and-click adding of new nodes and struts, as well as modeling of pre-defined polyhedra and exploration of, and autocompletion based on, the system symmetries. Recently, a first algorithmic approach dealing with Zometool for architectural and home-fabrication scenarios was contributed [3]. While panel planarity can be guaranteed by that approach, the therein utilized surface growing approach is applicable only to disk topologies and the run-times (several minutes to hours) hamper many practical applications. To the best of our knowledge, there exists no software solutions or algorithms for assisting in the Zometool realization of closed freeform surfaces of arbitrary genus. 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. Thus, although mathematically well-founded and beautiful, typ-

ical recreationally built Zometool structures are still often (i) geometrically and topologically simple or (ii) highly symmetric and regular. Some complex, real-life examples are shown in Figure 1(a,b). Hart and Picciotto [4] 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`). For geometry processing tasks, the inherent discreteness and combinatorics of the system rule out the use of efficient numerical solvers and make freeform surface approximation a challenging problem. In this paper we present an algorithm to support freeform Zometool structure creation by exploring the model-space to find a fitting approximation to a given input design. We explore the search space efficiently through a simulated annealing framework. Figure 2 outlines our pipeline. Figure 1(c) shows a real-life assembled result based on our algorithm: the first Stanford Zome-Bunny.

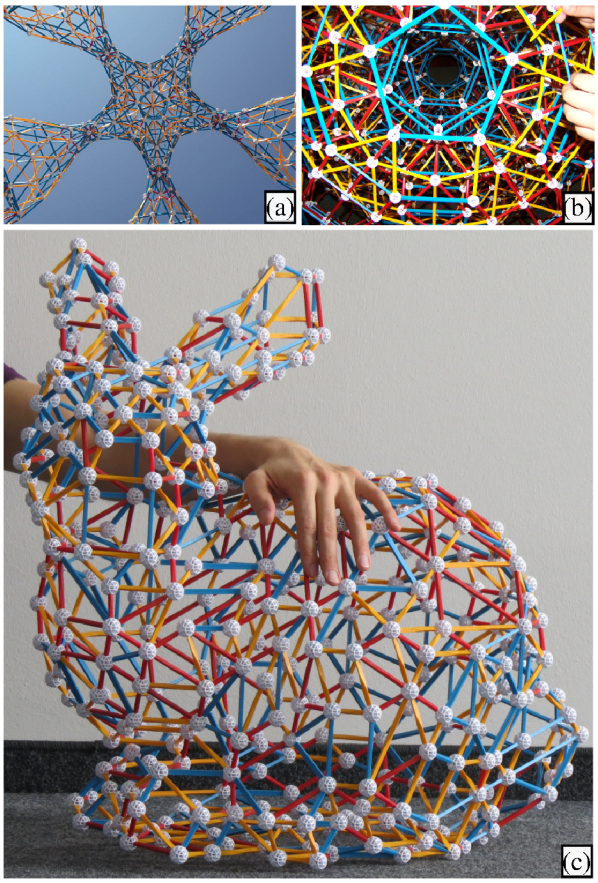


Figure 1. (a,b) Some complex, real-life Zometool structures. (b) Our result: Stanford Zome-Bunny (cf. Figure 10).

1.1. Related Work

(Constrained) Remeshing. To the best of our knowledge there are no directly comparable remeshing techniques with similar constraints (restricted to a predefined, fixed set of elements). Typically, remeshing algorithms ([5, 6, 7] provide an overview) are often 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. The so-called rationalization approaches in architectural geometry dealing with panel optimization are related in that the shape diversity of panels is optimized. Methods such as [8, 9, 10, 11] typically consist of two main components: a discrete optimization for determining a minimal set of panels and a continuous optimization of the panel shapes. They differ to our setting in that (i) an initial solution (in particular a fixed tessellation) is given and (ii) they allow for a continuous relaxation to make it fit. Neither of these assumptions hold in the Zometool setting considered here.

Simulated Annealing. The huge search-space and discrete nature of the Zometool surface approximation problem maps well to the setting of simulated annealing (SA) [12, 13]. SA, often applied on discrete search spaces and complex problems not allowing for any straight forward analytical computation, has been successfully employed in various areas of computer vision and graphics, e.g., for generating good building layout [14], for approximation of scattered data [15], structural reconstruction from images [16] or triangle mesh repair [17]. The setting considered in this work is however even more constrained, having not only requirements of 2-manifoldness and topological consistency of the polygonal (Zome) meshes but also only a discrete set of possible connecting edges.

1.2. Our Approach

We pose the problem of finding the best 2-manifold Zometool mesh representation, a *Zome mesh*, for a given input surface as a non-linear optimization problem specified by an energy function, which measures the quality of the approximation. Our solution consists of two parts: (1) finding a valid, initial approximating Zome mesh and (2) incrementally modifying the approximation to minimize the energy. We find an initial Zome mesh based on a voxelization of the input shape and minimize the energy by iteratively applying a set of local mesh modification operators. The output is a mixed triangle/quad polygonal surface mesh with

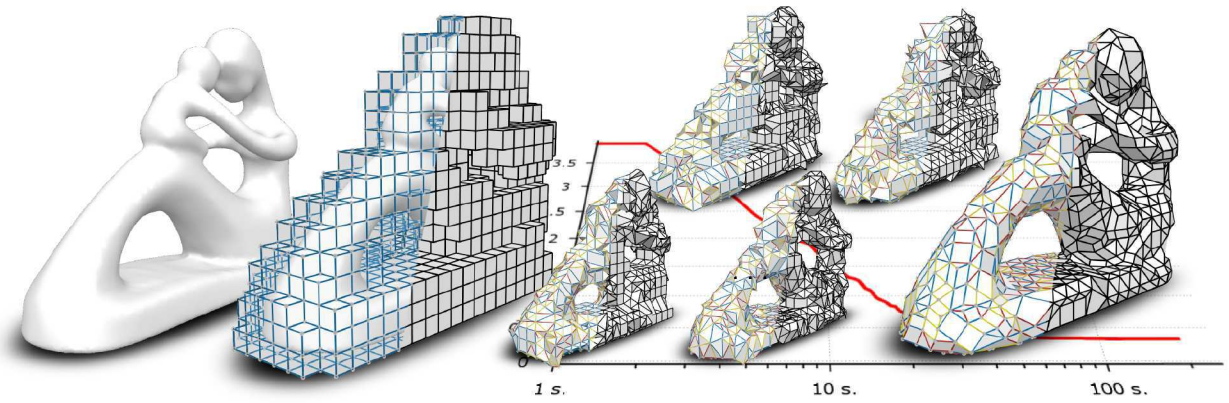


Figure 2. Zometool Surface Approximation Pipeline: First a rough Zometool approximation of the input surface is computed. Then the model-space around the input shape is explored by our algorithm to find an optimized (low energy) Zometool representation.

edges and nodes compatible with the Zometool system. While our approach is not limited to a certain polygonal type, we restrict the polygonal degree to 3 and 4 for structural robustness, and to implicitly avoid "too" concave or non-planar configurations.

The topology and genus of the Zome mesh are defined by basing it on an underlying 2-manifold polygon mesh, whose vertices and edges are Zometool nodes and struts respectively. The faces of this polygon mesh imply faces of the Zome mesh defining its surface topology and orientation. Throughout this paper $\mathcal{Z}$ refers to the Zome mesh, $\mathcal{S}$ to the input surface and E to the energy functional.

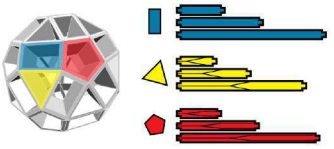
1.3. Contributions

Our main contributions lie in the definition of the mesh modification operators, which provide the basis for the global optimization process that finds a Zometool approximation for a given input mesh.

- *Topology Preservation*: The operators are topology (manifoldness and genus) preserving, meaning that given an initial 2-manifold Zome mesh of a certain genus the result is guaranteed to preserve these properties.
- *Parallel Exploration*: By design the operators are local as only influence a small region on the mesh, a fact that we exploit to implement the exploration of the Zometool space in an efficient and parallel manner.

Furthermore, by a trivial mapping of the irrational, 3-dimensional Zome coordinates to a 6-dimensional integer space, we enable *exact* and efficient equality comparisons in the implementation, avoiding the need to resort to ϵ tolerances.

2. Zometool Background



The three different strut types of the standard Zometool system are colored blue, yellow and red. Each strut is straight and comes in three lengths. Let b_0, b_1, b_2 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. Note that the strut lengths are measured from node center to node center. The Zome node is a slightly modified rhombicosidodecahedron. Each of the 62 slots is restricted to a single type of strut: there are 30 rectangular slots for blue, 20 triangular slots for yellow, and 12 pentagonal slots for red struts.

1. *Strut lengths*: The lengths of the struts are related by the golden ratio $\gamma = \frac{1+\sqrt{5}}{2}$ as follows $b_{i+1} = b_i \cdot \gamma$ (analogously for yellow and red). Also, the different colors are related by the relative edge lengths of the platonic solids, i.e., $y_i = \sqrt{3}/2 \cdot b_i$ and $r_i = \sqrt{2+\gamma}/2 \cdot b_i$.
2. *Node symmetry*: Due to the symmetry of the rhombicosidodecahedron there is for each slot an opposite slot of the same type and, incidentally, as $\gamma^2 = 1+\gamma$, the longest struts (r_2, y_2, b_2) can be built by combining the two shorter ones of the same type, e.g., $b_2 = b_0 + b_1$.
3. *Fixed orientation*: A consequence of the node symmetry and the nature of the struts is that the nodes at both ends of one strut are related by a pure translation.
4. *Zome vectors*: 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. We refer to these 186 vectors as the set of *Zome vectors* $\mathcal{V}$, where each $v \in \mathcal{V}$ corresponds to a unique slot/strut length combination. Taking any three pairwise orthogonal rectangular (blue) slots to define the axes of a right handed coordinate system, the coordinates of the Zome vectors are of the form[1]: $(a_0\gamma + a_1, a_2\gamma + a_3, a_4\gamma + a_5)/2 \in \mathbb{R}^3$ with $a_i \in \mathbb{Z}$.

2.1. 6D Integer Zome Coordinates

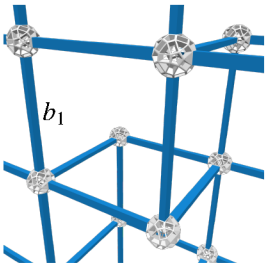
The irrational coordinate values of the vectors in $\mathcal{V}$ together with floating point arithmetic make exact comparisons of the form $v_i = v_j$ unreliable. Still, we need such comparisons between different vectors, e.g., for defining constraints. To avoid working with ϵ tolerances we transform the above 3D floating point coordinates to 6D integer coordinates which can then be compared exactly. This is done by expressing each of the above coordinates $(a\gamma + b)/2$ in the basis $(\frac{\gamma}{2}, \frac{1}{2})$ and taking the integer coefficients (a, b) to be a new 2D coordinate with one "golden" part and one "integer" part. These 6D vectors can then be added, multiplied exploiting the equivalence $\gamma^2 = \gamma + 1$ and compared exactly through their integer coefficients. In the following this representation is assumed whenever exact comparisons are needed.

3. Zometool Meshing

Generating good and valid Zome meshes from freeform shapes is a complex algorithmic problem. A naïve strategy would consist in constructing Zome meshes by a growing procedure: starting from a seed node further struts and nodes would be progressively added until recovering the whole input shape. However, in general there is no efficient way of merging the expansion on non-zero genus input shapes. We adopt a more global strategy based on a simulated annealing framework in which local operations are performed to iteratively modify an initial Zome mesh.

3.1. Initial Solution

The initial Zome mesh $\mathcal{Z}^0$ is obtained as the 2-manifold surface of a voxelization of input surface $\mathcal{S}$. This is possible due to the existence of pair-wise orthogonal (blue) node slots. We fit a bounding box to the input mesh $\mathcal{S}$ and fix the global node orientation by aligning these struts along the bounding box axes. For relating the fixed lengths of the struts to the metric of $\mathcal{S}$ we choose a global scale. 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 it is 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 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 output. 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 arrangement 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 $\mathcal{Z}^0$. We found that such configurations can often be removed by applying local, topology preserving morphological voxel operations.



We compute a so-called *conservative* voxelization which, at least at first, completely covers the input $\mathcal{S}$. To this end the (appropriately expanded and centered) bounding box is split into b_1-sized cells and all cells lying completely inside of $\mathcal{S}$ or intersecting $\mathcal{S}$ are tagged. The intersecting cells are found efficiently in a hierarchical fashion based on a three-color octree. To extract the 2-manifold boundary $\mathcal{Z}^0$ we assume that an appropriate scale b_1 has been chosen and that a voxelization of the desired genus has been computed. Then, as long as non-manifold configurations exist, we remove *simple* voxels incident to these configurations. A voxel is called *simple* if it does not change the genus (cf. Bischoff et al. [19]). A 2D illustration of this idea is shown in Figure 3. If not all non-manifold configurations can be removed by these local operations, which happens very rarely in degenerate configurations, we propose to slightly adjust the scale and retry.

[1] This form also holds for the curved green struts (cf. [18]), which can be additionally introduced to extend the system.

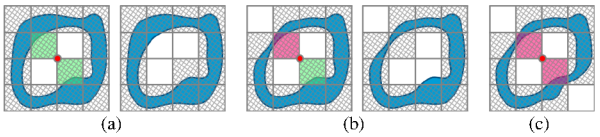


Figure 3. (a) depicts a non-manifold vertex (in 2D) and two adjacent *simple* cells in green, either can be removed to resolve the configuration while preserving the genus. In (b) the red cell is no longer simple, as its removal would change the genus, however, the green cell can still be removed. Finally, (c) depicts a configuration that cannot be resolved by this approach.

3.2. Simulated annealing

Simulated annealing (SA) optimization is adopted to let the initial configuration evolve. Simulated annealing can be seen as an optimization technique for non-convex energy functionals. Based on a randomized process, a local modification of the current configuration is proposed at each iteration. This proposal is then either 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 SA process is specified by three main 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 must be reversible, i.e., the inverse modification must be possible.

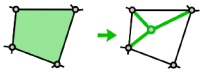
  In our setting three types of operators are required: operators for *increasing* model complexity, *changing* model geometry and *decreasing* model complexity (we measure model complexity by the number of nodes N). Simple node and face based operators are applied to locally modify the geometry and complexity of the mesh by inserting, moving or removing nodes. In order to explore other different connectivities 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 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.

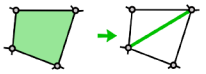
- *Cooling schedule.* It specifies the form of the relaxation parameter T_t, referred to as temperature, and its initial value T_0 that both control the degree of randomness of the simulated annealing. The temperature T_t is a decreasing series approaching zero as t tends to infinity. A logarithmic decrease of T is required to ensure convergence to the global minimum from any initial configuration. However, for efficiency we rely on the commonly used geometric cooling schedule $T_i = T_0 \cdot \alpha^i$ (cf. [20]), which decreases faster while still yielding good approximate solutions in practice.

Set of local Operators. Below, the implemented operators are listed together with a brief description and a figure of their respective support regions (their boundaries are referred to as *link*s).

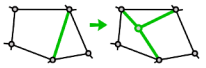
- InsNode(f) inserts a new node and adds all possible strut connections to the nodes of the link.



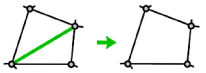
- AddDiag(f) splits a quad face by adding a diagonal strut.



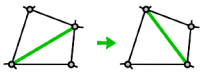
- SplitStrut(s) splits s by inserting a new node and forming all possible new connections to the link.



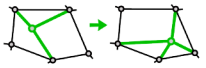
- RemDiag(s) removes a (diagonal) strut s and thus merges two triangles.



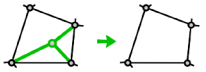
- FlipDiag(s) flips a (diagonal) strut s separating two triangles.



- MovNode(n) moves a node and forms the possible link connections.



- RemNode(n) removes a node 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 must at least 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. This means that the SA framework not only selects one of the 7 operators but also needs to select one of several combinatorial options. Details on how an operator can be efficiently generated without enumerating all such combinations of possible link connectivities are given in Section 5.

Operator Validity. The operators are constrained to never introduce vertices with valence < 2 and 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 forming all possible connections from the new center node to the nodes in the link and then testing for faces of valence > 4. 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 node slots. We note that local fold-overs of the Zome mesh may lead to strut collisions preventing physical realization. However, for efficiency these intersections are not explicitly checked for, but are instead handled indirectly by the orientation energy functional used to prevent fold-overs (cf. Section 4). The 2-manifoldness conditions are discussed in Section 5.

Parallelization. The conventional simulated annealing performs successive local modifications on the current configuration. Such a process 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 4 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 $v \in \mathcal{V}$ and the vector $w = -v$ pointing in opposite direction.

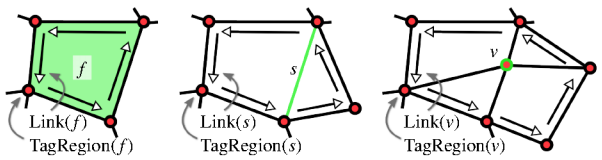


Figure 4. The TagRegion (red nodes) and oriented edges of the Link (arrows) of different mesh entities (green).

4. Energy Functional

E is a linear combination of separate energy terms, accounting for different aspects of the quality measure,

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

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 p to their nearest points $\pi(p)$ on $\mathcal{S}$, let $n(\pi(p))$ be the normal vector on $\mathcal{S}$ at this position.

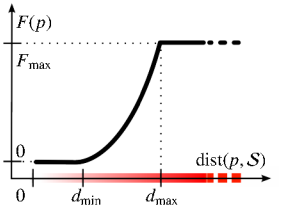
4.1. Distance

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

$$E_{\mathrm{distance}}(\mathcal{Z}) = \frac{1}{P \cdot d_{\mathrm{norm}}^2} \sum_{i=1}^{P} \|p_i - \pi(p_i)\|^2 \cdot (1 + F(p_i))$$

where P is the number of samples (#nodes + #struts + #faces) and the normalization factor d_{norm} is used to relate distances to the fixed lengths of the struts. We choose $d_{\mathrm{norm}} = b_0$, which keeps the energy within the range $[0, 1]$ for positions with distances less than b_0 from $\mathcal{S}$. We call the term $F(p_i)$ *forbidden zone*, which we introduce to further penalize points lying too far away from the surface.

Forbidden Zones. In combination with thin surface 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 disallow mesh primitives passing through the interior of $\mathcal{S}$ we make the interior more expensive for the optimization by introducing the *forbidden zone* term. $F(p)$ is a quadratically increasing function which depends on the distance of the position p to the surface $\mathcal{S}$: it is equal to zero for positions lying just below the surface of $\mathcal{S}$, then increases quadratically after a certain distance d_{min} until assuming the maximal value F_{max} at d_{max}. Our standard weights focus on penalizing interior points to avoid degeneracies, we use $d_{\mathrm{min}} = b_0/3$, $d_{\mathrm{max}} = b_0 \cdot 1.5$ and $F_{\mathrm{max}} = 35$, for outside points we use $d_{\mathrm{min}} = b_0/3$, $d_{\mathrm{max}} = b_0 \cdot 2.5$ and $F_{\mathrm{max}} = 15$. This choice works well in practice (cf. Figure 10), but can also further be adapted to object specific needs.



4.2. 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 a and b, direction $d = (b - a)$ and midpoint $m = (a + b)/2$. The tangential part measures the deviation of the strut direction d from the tangent plane at the point on $\mathcal{S}$ closest to m:

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

where S is the number of struts in $\mathcal{Z}$. Now, let a, b, 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:

$$E^{\text{normal}}_{\text{orientation}}(\mathcal{Z}) = \frac{1}{4C} \sum_{f \in \mathcal{Z}} \sum_{b \in f} \left(1 - n(\pi(b))^\top \frac{(c-a)\times(b-a)}{\|c-a\|\,\|b-a\|}\right)^2,$$

where C is the number of corners in $\mathcal{Z}$. This property is later (cf. Section 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 promotes face concavity. Note that the energy terms could be refined by further introducing length and area weighting. Empirically we found that combining the two parts 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, leads to good results.

4.3. Fairing

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

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

where N is the number of nodes in $\mathcal{Z}$ and $N_1(p)$ the 1-ring neighborhood of nodes around p. By uniformly distributing the nodes this energy also contributes to fold-over reduction.

4.4. 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:

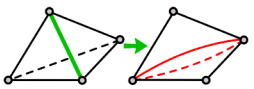
$$E_{\text{complexity}}(\mathcal{Z}) = \frac{1}{N_{\text{target}}}(N - N_{\text{target}})^2.$$

5. Implementation Details

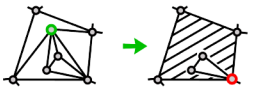
Our implementation is based on a standard halfedge-based mesh data structure (`www.openmesh.org`) to represent both $\mathcal{S}$ and $\mathcal{Z}$, and relies on OpenMP (`www.openmp.org`) for parallelization. The projection operator $\pi(p)$ is implemented as a BSP on $\mathcal{S}$ supported by a spatial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

5.1. The Operators

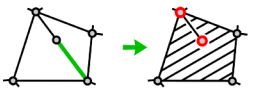
2-Manifoldness of Operators. To keep implementation simple all operators are built upon combinations of the four basic functions supplied by most halfedge-based mesh data structures: *add/remove vertex* and *add/remove face*. While these functions themselves are safe, there are some pathological configurations which need to be handled. E.g., recall the MovNode(n) operator, which generally yields a new connectivity inside the link of the node n. This operator can be implemented in three simple steps: (1) clear the old link, 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 shown in the second inset below, already the first step (removing old faces) can lead to a non-manifold configuration, from which the following steps and the data structure may no be able to recover. Three prominent, problematic cases are:



- Flipping certain edges (e.g., in a topological tetrahedron) can lead to pairs of nodes "doubly" connected by two edges.



- Removing links in which a single node is pointed to by more than one of the oriented boundary edges yields a non-half-disc neighborhood around this node.



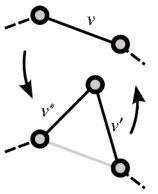
- Removing an edge of a valence 2 node leads to a dangling valence 1 node and a non-disc neighborhood.

In short, we disallow operations that lead to doubly connected vertices and operations where a node in the link is pointed to twice by the oriented edges.

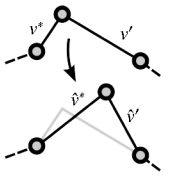
Look-Up Tables of Basic Two-Strut-Operations. Enumerating all possible (and valid) Zometool fillings for an arbitrary link can not be done efficiently. Even the much smaller setting where the filling is restricted to add only a single new node is already complex. We reduce this complexity by not considering the whole link

at once but by building the operators on combinations of simple sub-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 SplitVector(v) = $\{(v^*, v')\}$ table maps a Zome vector v to a set of *pairs* of vectors (v^*, v') with $v = v^* + v'$, meaning a Zome strut with vector v can be replaced by two struts (connected by a new node) with vectors v^* and v'.



- MergeVectors(v^*, v') is the table of inverse operations, mapping a pair of vectors (v^*, v') to *at most* one direction v. Note that not all strut pairs can be replaced by a single strut.
- The ChangeDirections(v^*, v') = $\{(\hat{v}^*, \hat{v}')\}$ table, maps a pair of vectors (v^*, v') to a set of new pairs $(\hat{v}^*, \hat{v}')$, thereby effectively moving the 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 a positional degree of freedom. The combinatorial computation of the tables is straight forward 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) look-ups in the aforementioned tables. The InsNode, MovNode and SplEdge operators are similar: (1) the link 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 MergeVectors operation on two random (adjacent) edges of the link of a quad, and FlipDiag, RemDiag, and RemNode are straight-forward.

5.2. Parallelized Simulated annealing

The simulated annealing process used to explore the Zometool model-space is detailed in Algorithm 1, where the "Parallel Region" is executed by each thread. 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 validity of the operator is evaluated and if the operator is valid then the energy update, which would result from performing the operation, is computed. Now, if the proposition of modification is accepted (energetically) then the operation is carried out and the region is subsequently un-tagged and is again free to be used by other threads/operators.

Algorithmically all the local operators (cf. Section 3.2) have a common interface of functions regardless of their input entity (node, strut or face) and all have a reference to the global energy and Zome mesh $\mathcal{Z}$:

- `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

Some functions either modify the mesh data structure (by setting tags or removing/adding entities) or rely on a consistent global state of the mesh (to correctly evaluate the global energy) and must be protected by appropriate locks/semaphores in a parallelized setting. In Algorithm 1 such critical rows are marked by an asterisk *. In particular, as the energy evaluation (`simulate`) depends on the global state of $\mathcal{Z}$ and performing an operation (`commit`) changes this global state, the 4 rows marked by ! must all be performed together in one critical section to keep the energy consistent. Further 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.

6. Results

Zometool shapes computed by our method are shown in Figure 10. Please also see the accompanying video and the project website (`www.rwth-graphics.de/zometool`). For comparability all results were computed using 7 threads on a standard i7-PC using the *same*

Algorithm 1: SA for Zometool Approximation

Input: input surface $\mathcal{S}$, initial mesh $\mathcal{Z}$, set of operations O, initial temperature T_0, min. temperature $T_{\min}$ and decrease factor α
Shared vars.: T_i, α, $\mathcal{Z}$, time
--- Begin Parallel Region ---;
while *done = false* **do**
  o = random_operation(O);
* tagok = o.tag_region();
  if *tagok* **then**
    if *o.valid()* **then**
      p = uniform_prob[0,1];
! ΔE_o = o.simulate();
! **if** $p > exp(-\frac{\Delta E_o}{T_i})$ **then**
! o.commit();
! cnt++;
      end
    end
* o.untag_region();
  end
* $T_i = T_0 \cdot \alpha^{\text{cnt}}$;
  if *time up or* $T_i < T_{min}$ **then**
    done = true;
  end
end
--- End Parallel Region ---;

parameter settings (unless stated otherwise). The input surfaces to our method are generally assumed to be sufficiently smooth surfaces and we have intentionally computed the examples with coarse resolutions to challenge our method. Naturally, as a trade-off for larger physical realizations, the voxelization resolution can be increased to trivially reduce the relative error. But even within the same scale the number of nodes can be influenced by modifying N_{target}. Note that due to the limited availability of angles and edge lengths in the system general feature preservation is not possible. To approximate extreme details such as needle-like spikes, the voxelization resolution needs to be correspondingly adapted to the feature size (cf. Figure 5).

Weights and Parameters. In all our experiments the weights and parameters are fixed to: $w_d = 10$, $w_f = 5$, $w_o = 100$, $w_c = 1$, $T_0 = 0.1 \cdot S\,tdev(\Delta E(\mathcal{Z}^0))$, $T_{\min} = 0.00001$, $\alpha = 0.999995$. The target complexity is $N_{\text{target}} = N^0$, with N^0 being the number of nodes of the initial solution. A high orientation and fairing energy helps to 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

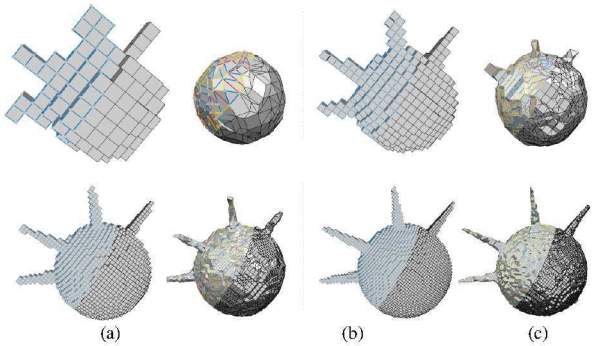
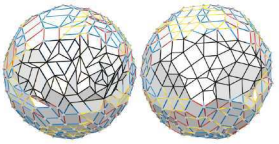


Figure 5. Different resolution conservative voxelizations and resulting Zome meshes for a sphere with needle-like spikes.

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 larger than 1 for distances outside the range $[0, b_0]$ (cf. Section 4), this means that distance minimization is prioritized in the beginning of the optimization, while orientation dominates toward the end. This is confirmed by starting ten identical runs on the RockerArm. The initial energy is dominated by the distance term whereas the final energy is comprised mainly of the orientation:

	Initial		Final (avg)	
E_{distance}	12.5	82%	0.039	13%
$E_{\text{orientation}}$	2.68	18%	0.226	73%
E_{fairing}	0.06	0.0%	0.042	14%
$E_{\text{complexity}}$	0.0	0.0%	0.0	0.0%
E	15.24	100%	0.318(±0.013)	100%

Each computation takes only 3 min. The low standard deviation (0.013) further suggests that a common minimum has been found. However, although visually pleasing, the meshes can still differ in the details as demon-

strated in Figure 7(a,b). Comparing the above energy to that of a "ground truth" computed with a slower cooling schedule over $24h$ time (cf. Figure 6), we can conclude that these are very good local optima. The evolution of the optimization is also visualized in the accompanying video.

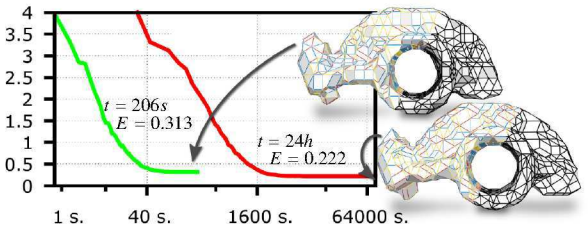


Figure 6. The top RockerArm is obtained after about $3min$ with a final energy of 0.313, while the bottom result is obtained after $24h$ with a final energy of 0.222.

Exploring Model Complexity. Once a good solution is found, the parameters can be tailored to modify the solution further (by running the optimization again, with the previous solution as initialization). Figure 10 (along the green strip) shows 3 meshes out of 8 from of two different fixed-resolution hierarchies respectively, one *refinement* hierarchy of the RubberDuck and one *decimation* hierarchy of the Fertility. These hierarchies are 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. 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 processes. 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.

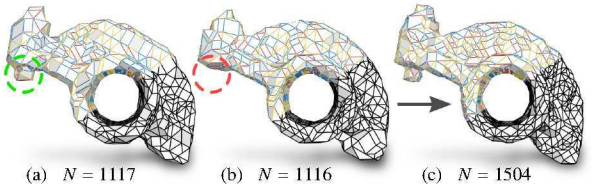


Figure 7. (a) and (b) demonstrate how using SA the same weights can sometimes lead to slightly different results. Once a good solution exists, further passes can be used to refine the result (b)→(c).

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 RockerArm in Figure 7(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. E.g., 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 8). On the Elk result in Figure 10 the wheels and the body

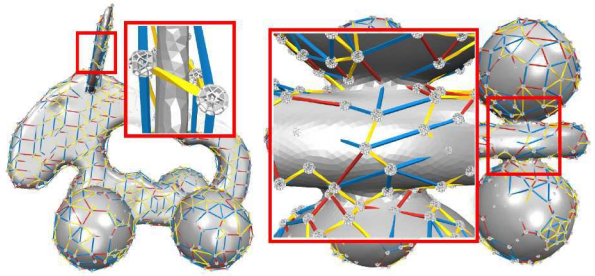


Figure 8. At the chosen resolution the antlers are thinner than y_0 and it is a difficult task to separate the "wheels" from the body of the Elk.

are separated but the antlers are partly lost. The above mentioned post-optimization 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 they 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). Now, by iteratively

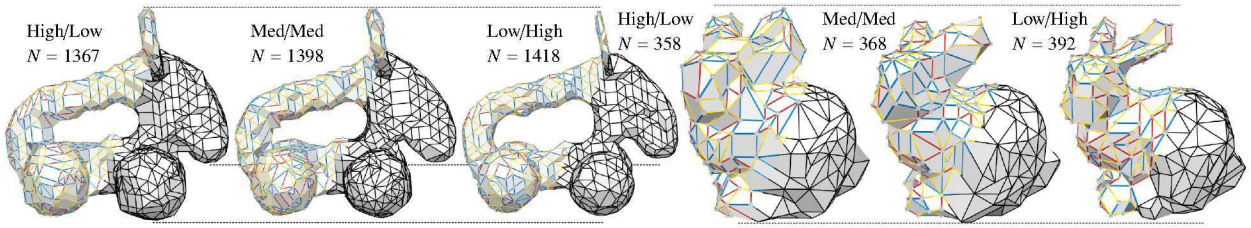
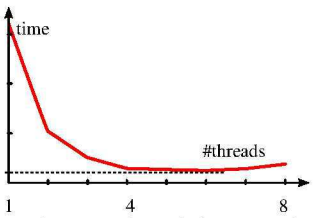


Figure 9. A 3-step scale of decreasing orientation weights and increasingly tightened forbidden zone weights (High/Low to Low/High) demonstrating how thin surface parts (ELK antlers and BUNNY ears respectively) can be preserved while still properly reaching concavities (e.g., between ELK wheels). The dotted lines show how high orientation tends to "bloat" the surface. Using the resulting N of the previous step as the new N_{target} also has a relaxing effect on the mesh complexity.

tightening the *forbidden zone* from the outside and decreasing the orientation weight, the final result (right) is obtained in about 15 minutes, with each iteration taking about 5 min. The Stanford BUNNY is also computed using this strategy.

Scalability. A performance speed-up of about 3 – 4 times is 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 initially 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 exemplary time vs. number of threads plot in the inset shows that no performance gain can be expected from our implementation when using more than 7 threads. This gain also much depends on the type and number of locks used, and further optimizing these parameters could slightly improve the plot. To achieve true linear behavior we also experimented with a patch-based approach, that iteratively: (1) segmented the surface into independent parts (thereby alleviating the need for locks) and (2) optimized the patches separately. While the actual optimization now scales well, too much time is lost in stitching the patches back together and recomputing a new layout. Convergence is also questionable, since at no point was the complete global energy optimized.



7. Limitations and Future Work

While our solutions are already of quality, additionally mastering and optimizing the global orientation and scale of the Zometool system could further improve the results. One possibility would be to re-run the optimization for different global orientations and scaling factors. Also, to increase angular resolution the set of Zome vectors could be extended by additional strut types (and also non-standard strut lengths). E.g., green struts (although curved) yield coordinates of the same form and can be added at the price of increasing size of the lookup tables. However, owing to the restricted set of available angles and edge lengths general feature preservation will never be possible. As hinted by the 24h test a more clever cooling schedule could enable lower energies.

Our optimization is topology preserving, but requires a valid input Zome mesh. A theoretical limitation in the current setting is the lack of guarantee of the existence of such a valid initial approximation of preferred resolution. As also discussed in Section 3.1, in a general setting, given an arbitrary target edge length, the problem of guaranteeing a voxelization with correct topology from which a 2-manifold surface can be extracted is hard. By choosing geometrically meaningful edge lengths for the shown meshes this is, however, not an issue in practice.

As future extension we imagine to enable symmetric outputs by introducing symmetry planes (similar to [3]). Furthermore, to handle surfaces with boundaries a different initialization would be required along with a type of freeform curve constraints to pin the Zome mesh down and thus avoid drifting and collapsing.

By the used energy functionals we are able to compute a wide range of fold-over free, physically realizable Zome meshes. However, as already mentioned, there is no guarantee of physical realizability. For certain applications, enabling such guarantees is of paramount importance. In addition, both for practical realizations and high-quality remeshing tasks the problem of keeping the Zome mesh within an ε-envelope of the input shape is a stimulating research direction, where the for-

bidden zone penalty term used in this work is just a first step. Finally, in the future we wish to explore the design of effective construction sequences that obey physical constraints, such as being self-supporting, and exploit the flexible properties of the strut material.

8. Conclusion

We devised a global approach to compute a 2-manifold Zometool approximation of a given input shape. Given an initial solution a set of localized mesh modification operators are iteratively applied to improve the approximation. All results shown are efficiently computed in the range of minutes, demonstrating the capabilities of the method. To our knowledge this paper is the first to address the kind of discrete and restricted remeshing setting implied by the Zometool system on closed surfaces of arbitrary genus. We hope that our approach will inspire more research and applications dealing with constrained construction systems such as the Zometool system.

Acknowledgments. We are grateful to George Hart (`georgehart.com`) for Figure 1(a) (shape design by Chris Kling), Tanya Khovanova (`blog.tanyakhovanova.com`) for Figure 1(b), Dominik Sibbing for the spatial hashing code, and to Tom Davis for technical discussions relating to the Zome coordinates. This work was funded by the DFG Cluster of Excellence UMIC (DFG EXC 89). Pierre Alliez is supported by an ERC Starting Grant "Robust Geometry Processing" (257474).

References

[1] E. Schlapp, ZomeCAD, 2014. URL: `http://www.softpedia.com/get/Science-CAD/ZomeCAD.shtml`, [Online].

[2] S. Vorthmann, vZome, 2014. URL: `http://vzome.com`, [Online].

[3] H. Zimmer, L. Kobbelt, Zometool rationalization of freeform surfaces, TVCG Preprint (2014).

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

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

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

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

[8] C.-W. Fu, C.-F. Lai, Y. He, D. Cohen-Or, K-set tilable surfaces, ACM Trans. Graph. 29 ('10) 44:1–44:6.

[9] M. Singh, S. Schaefer, Triangle surfaces with discrete equivalence classes, ACM Trans. Graph. 29 (2010) 46:1–46:7

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

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

[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics 21 (1953) 1087–1092.

[13] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[14] F. Bao, D.-M. Yan, N. J. Mitra, P. Wonka, Generating and exploring good building layouts, ACM Trans. Graph. 32 (2013) 122:1–122:10.

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

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

[17] M. Wagner, U. Labsik, G. Greiner, 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.

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

[19] S. Bischoff, L. Kobbelt, Sub-voxel topology control for level-set surfaces, Computer Graphics Forum 22 (2003) 273–280.

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

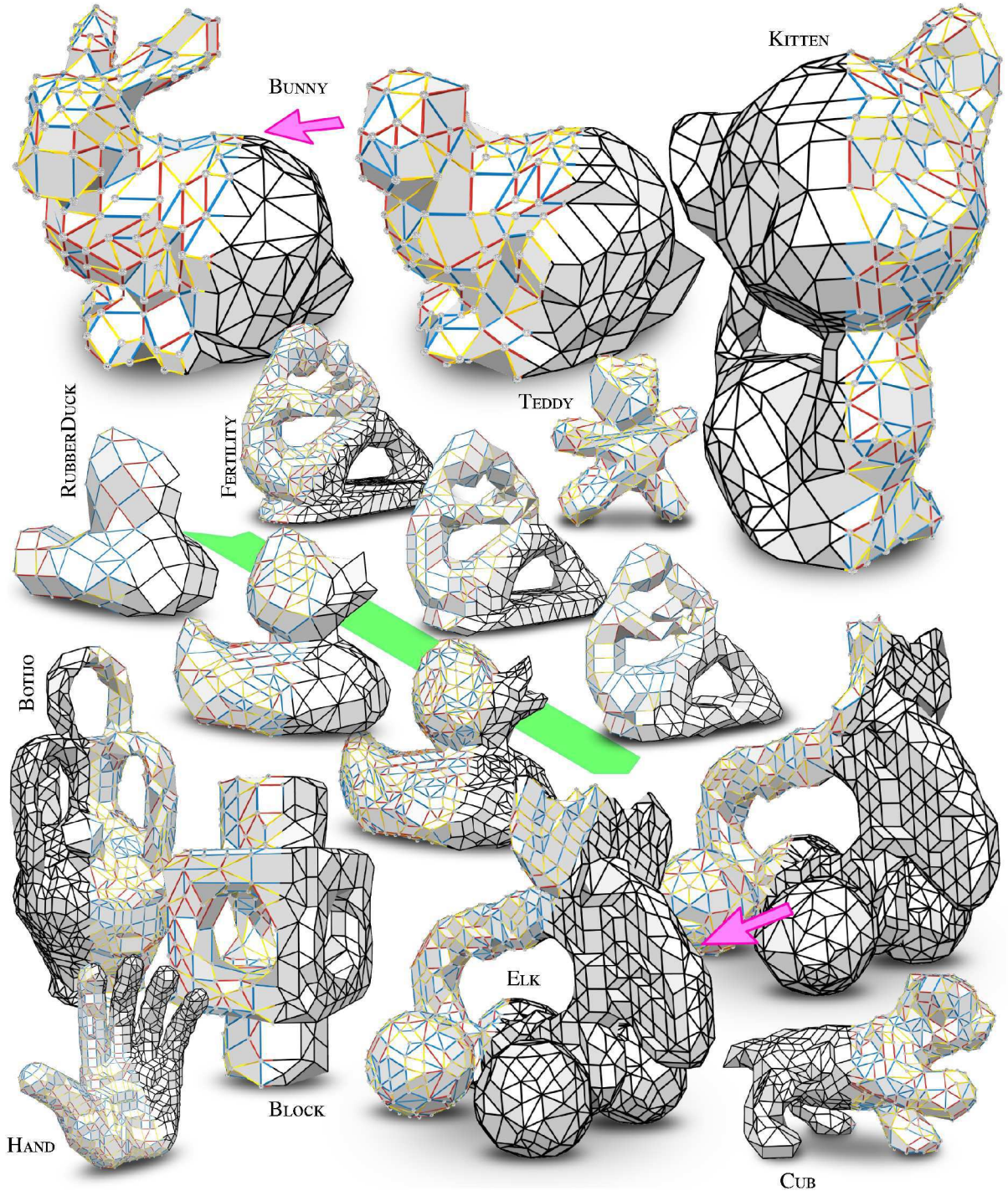


Figure 10. Resulting Zometool meshes. Along the green strip excerpts from a refinement and a decimation hierarchy are shown. At a coarse resolution the Bunny and Elk both had details missing using the standard weights (pink arrow). By using the 3-step strategy to preserve thin features (discussed in Section 6) these details were successfully captured.