Interactive Surface Design with Interlocking Elements

Mélina Skouras[1,2,3] Stelian Coros[1,4] Eitan Grinspun[5] Bernhard Thomaszewski[1]

[1]Disney Research Zurich [2]ETH Zürich [3]MIT CSAIL [4]Carnegie Mellon University [5]Columbia University

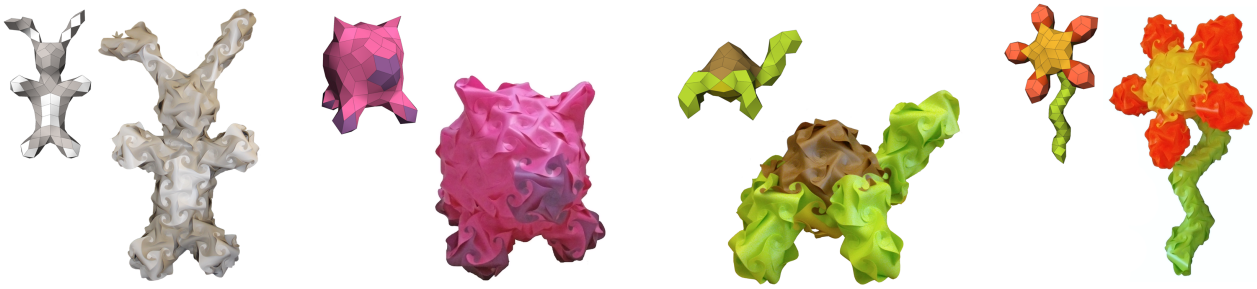


Figure 1: *We present a computational approach to designing assemblies made from interlocking quadrilaterals of the same size and shape.*

Abstract

We present an interactive tool for designing physical surfaces made from flexible interlocking quadrilateral elements of a single size and shape. With the element shape fixed, the design task becomes one of finding a discrete structure—i.e., element connectivity and binary orientations—that leads to a desired geometry. In order to address this challenging problem of combinatorial geometry, we propose a forward modeling tool that allows the user to interactively explore the space of feasible designs. Paralleling principles from conventional modeling software, our approach leverages a library of base shapes that can be instantiated, combined, and extended using two fundamental operations: merging and extrusion. In order to assist the user in building the designs, we furthermore propose a method to automatically generate assembly instructions. We demonstrate the versatility of our method by creating a diverse set of digital and physical examples that can serve as personalized lamps or decorative items.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

Keywords: interlocking, quadrilateral elements, physical surfaces.

1 Introduction

The assembly of complex structures from a small number of simple elements is a powerful design paradigm, one that harnesses economy of scale, freedom in form and function, and lightweight portable deployment. Drawing inspiration from existing *puzzle lamps*, we focus on the design of free-form surfaces built by interlocking multiple instances of a single element. The element that we consider in this work is quadrilateral in nature, having four *ports* that interlock with ports of neighboring elements (see Fig. 2). For such surfaces built from a single repeated element, the geometry of the surface is entirely governed by combinatorial considerations: the graph topology and the binary orientation of each element. See Fig. 2 for an example.

Designing in this restrictive space is challenging. In particular for novice users, deciding how to layout and connect individual elements in order to achieve a desired shape is very difficult and comes as a distraction from the creative intent—traits that quickly combine into a frustrating experience. We therefore propose a forward design tool that allows the user to think directly in terms of *shape*, rather than having to worry about connections between individual elements. The enabling technology for this design experience are high-level modeling tools that combine the widely followed paradigms of *modular design* and *geometry extrusion* in a simple-to-use front-end. Transparent to the user, however, we exploit geometric and combinatorial techniques to facilitate the design process: to fuse two design modules into one, we leverage a discrete differential geometric approach to quickly rule out incompatible interfaces among the exponentially growing graph combinatorics. To extrude a geometric form, we use an online graph search that finds self-similar interfaces in arbitrary assemblies, allowing them to be replicated periodically to achieve directable extensions of the structure. With these tools, users can quickly design larger models that would be very tedious to create by hand via trial and error.

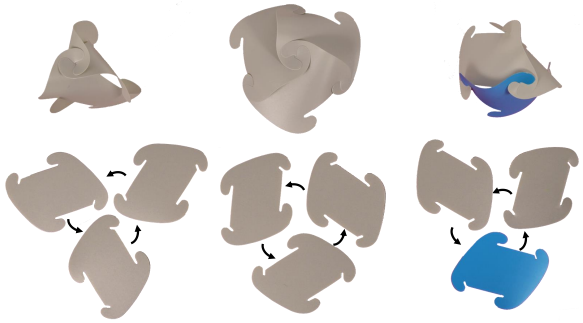


Figure 2: *Three identical interlocking elements are connected in different ways to create a variety of shapes. Blue color indicates* flipped *elements with opposite orientations.*

A design tool is only useful if it produces valid, realizable designs. Our system therefore accounts for two key fabrication constraints: (*a*) the flexible plastic or paper material from which the models are fabricated is *inextensible*, and (*b*) physical contact precludes packing more than a small number of elements incident to one graph node. In order to reflect these characteristics during the design process, we translate them into quantitative geometric and combinatorial constraints on the underlying search algorithms. Complementing shape design, we further propose a method to generate assembly instructions that guide the user through the fabrication process. Finally, we demonstrate the feasibility of the resulting designs by fabricating several physical surfaces.

2 Related Work

Our fascination for this subject has been sparked by recent work on creating physical artifacts made from interlocking three-dimensional [Xin et al. 2011; Song et al. 2012] or planar [McCrae et al. 2011; Hildebrand et al. 2012; Schwartzburg and Pauly 2013; Cignoni et al. 2014] pieces. While these works focus on generating the geometry of interlocking elements, we focus on the creation of networks from a given, specific interlocking element.

Building on previous works that develop interfaces for the crafting of tangible artifacts, we help the user to create networks of interlocking elements via an *assisstive user interface*. We are particularly inspired by the beadwork system of Igarashi et al. [2012], which guides the user in creating a polygonal mesh representation for the beadwork model. Fabrication constraints are considered by automatically adjusting edge lengths to comply with the minimum inter-bead distance. In a similar spirit, we also employ a polygonal mesh to abstract the design, but our problem brings forward new, intimate connections between topology and geometry.

In a broader scope, design tools for physical surfaces have received considerable attention from the graphics community in recent years. The range of surfaces is diverse, including paper-craft models [Mitani and Suzuki 2004; Kilian et al. 2008], paper popups [Li et al. 2010; Li et al. 2011], plush toys [Mori and Igarashi 2007], inflatable surfaces [Skouras et al. 2012; Skouras et al. 2014], wire-mesh sculptures [Garg et al. 2014], or models designed using construction kits with a discrete set of primitives [Zimmer and Kobbelt 2014].

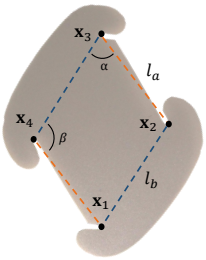
One particular line of work has explored the design and fabrication of self-supporting surfaces for masonry structures [Whiting et al. 2009; Vouga et al. 2012; de Goes et al. 2013; Deuss et al. 2014]. An alternative way of building self-supporting surfaces is by using reciprocal frames [2013], i.e., statically-stable structures made from interposed rods. We consider *interlocking* rather than self-supporting surfaces; both build surfaces out of elements without resorting to adhesives. However, rather than determining the parameters of a continuous design problem, the restriction to a single fabrication element renders our problem wholly discrete.

This turn toward the discrete is reminiscent of rationalization in architecture, i.e., the task of converting a given free-form surface into a design that is realizable with a restricted set of fabrication elements or techniques. As one example, Cutler and Whiting [2007] approximate free-form surface with planar panels using a constrained remeshing algorithm. Another example is the work by Eigensatz et al. [2010], who address the question of how to tile a free-form surface using as few custom-shaped panels as possible. The methods by Yang et al. [2011] and Deng et al. [2015] allow the user to explore spaces of meshes subject to nonlinear constraints such as planar quad meshes or circular meshes. While our approach has some similarities, it again differs in the fact that we are constrained to a single element shape, resulting in an even more restrictive design space. In order for the user to navigate this challenging design landscape, our method offers high-level modeling tools that encourage interactive shape exploration as a means of form finding and, ultimately, promote creativity.

3 Model

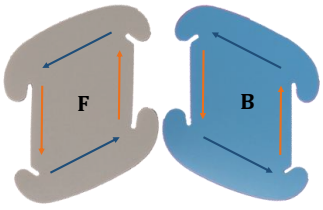
We consider the design of physical surfaces made from quadrilateral elements of the same size and shape. With a single type of element, the shape of the surface is determined entirely by its discrete structure, i.e., the connectivity of the elements and their orientations. Computing the *embedding* of a network with given structure, i.e., its deformed shape in 3D-space, is a critical part of our design tool that needs to be both fast and robust to warrant interactive feedback during design. As a prerequisite, we need a representation for networks of interlocking elements that (*a*) encapsulates all information required to determine the deformed geometry and (*b*) supports modeling operations for shape design (Sec. 3.1). Finally, in order to determine the global equilibrium shape, we need to understand the kinematic constraints governing the local deformations (Sec. 3.2).

3.1 Rhomboid Elements and Mesh Representation



The quadrilateral elements that we consider are *rhomboids*, i.e., parallelograms with opposite sides of equal lengths, adjacent sides of unequal lengths, and oblique interior angles. The physical elements are typically made from thin sheets of plastic material and will readily bend, but strongly oppose stretching. A flat rhomboid element is defined by its two side lengths $l_a < l_b$ and interior angles $\alpha < \beta = \pi - \alpha$ as shown in the inset figure. Each of the four nodes is equipped with an oriented hook, serving as a fixture to connect edges from neighboring elements. Two elements are connected by inserting a long edge of one element into a short slot on the other element. Physically, this connection is achieved by bending the long edge such that the distance between its nodes becomes equal to the short edge length l_a. This leads to the characteristic curved appearance of the elements, but it also produces forces on the nodes of the receiving element. These forces secure the connections, leading to a stable assembly of interlocking elements.

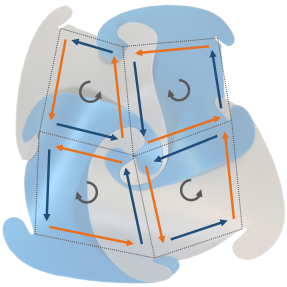
Mesh Representation We represent an interlocking element assembly by an *oriented manifold quad mesh*, using the terminology of a *tagged half-edge data structure*. Let a mesh be formed from n vertices at positions $\{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$ organized into quadrilateral faces $\{f^1, \ldots, f^m\}$. The interior mesh edges consists of pairs of half-edges, each associated to an incident face.



Each face f^i corresponds to one design element, with corners occupying positions $\mathbf{x}^i_j$, $j \in \{1, \ldots, 4\}$ drawn from the set of vertex positions. Furthermore, each design element has two opposing long sides and two opposing short sides and we therefore tag each half-edge as either *long* or *short*. Finally, each element may be oriented as either front- or back-facing relative to the mesh orientation. We

therefore tag each face with this *F*/*B* orientation, thereby indicating which diagonal of the quadrilateral face is long/short.

Each mesh edge represents the interlocking of two adjacent design elements. By nature of the interlock mechanism, each edge joins one short side to one long side. We therefore require that half-edge pairs have opposite *long*/*short* labels (see incident figure). This constraint fixes the *long*/*short* labeling of all mesh faces once the orientation is selected at any one face. As a result, once the connectivity among the elements is known and any one edge in a face has been tagged as *long* or *short*, the only remaining choice for each element is whether it is front- or back-facing. If all the faces sharing a common vertex have the same orientation, the angles incident to this vertex will be either all small (Fig. 2, left) or all large (Fig. 2, middle). Changing the orientation of selected elements allows for combining small and large angles while ensuring the short-to-long condition (Fig. 2, right). It is worth emphasizing that, while the element orientations have to be set once per base shape, they are automatically adapted during interactive design such that the user remains untroubled by technical issues. With a proper representation for our surfaces in hand, we now turn to the conditions that govern their deformed shapes.



3.2 From Combinatorics to Embedding

Given the discrete structure of the network, we wish to determine the embedding of the resulting form in 3D-space. Computing this embedding is a critical step that has to be performed whenever the user changes the design. We therefore seek a solution that is fast yet accurate enough to faithfully represent the deformed surface.

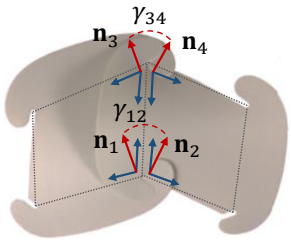
The shape of the deformed surface results from a complex interplay of large bending deformations and frictional contact between the curved elements. While nonlinear thin shell solvers can predict elemental and global deformations with high accuracy, they are computationally too demanding for interactive applications. However, if the local shape of individual elements is not important, the global shape can be captured in a simple and efficient way: since the medium is quasi-inextensible, the distance between any two corner vertices of a deformed element can never exceed their distance in the initial (flat) configuration. These observations translate into six kinematic constraints per element (four edges, two diagonals) that can be modeled with a minimal representation, i.e., the corner positions of the elements and their orientation—which is precisely the information furnished by our tagged half-edge data structure.

Energetic formulation In order to model the kinematic constraints described above, we follow a penalty approach and formulate energy terms that strongly penalize stretching of edges and diagonals. Compressions of edges and diagonals correspond to bending of the element and should be penalized as well, but at much lower intensity. We model this behavior with a biphasic energy

$$E_l = \frac{k^l(l_{ij})}{L_{ij}}(l_{ij} - L_{ij})^2 \,, \qquad k^l = \begin{cases} 1 & l_{ij} \le L_{ij}, \\ 100 & l_{ij} > L_{ij}, \end{cases}$$

where $l_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$ and L_{ij} are the distances in the deformed and undeformed (flat) configurations, respectively. The deformation-dependent stiffness coefficient k^l models the difference in the material's resistance to stretching and compression.

While these per-edge energies capture all in-plane deformation, they do not penalize out-of-plane motion. More concretely, for any one-ring of elements incident to a vertex with nonzero angular defect, both the convex and concave configurations store equal energy. In practice, however, the connecting tabs and hooks of the elements lead to a strong preference for locally convex regions. We therefore enrich our energy with a term that penalizes concavity,



$$E_b = k^b_{12}\gamma^2_{12} + k^b_{34}\gamma^2_{34} \,, \qquad k^b_{ij} = \begin{cases} 0 & \text{edge is convex,} \\ 1 & \text{edge is concave.} \end{cases}$$

In order to compute the embedding for a network with given discrete structure, we minimize the penalty energies summed over all edges and diagonals. Since both energies admit closed-form expressions for first and second derivatives, we use Newton's method to quickly converge to a minimum. It is worth mentioning that, although both energies have discontinuous second derivatives at the transition between their two regimes, their gradients are continuous and we observe good convergence in practice.

Discrete Gaussian Curvature While the energetic formulation allows us to compute the embedding of a given tagged mesh, our interactive design tools also require a faster, local predictor of shape and feasibility. Indeed, the mesh combinatorics (including tags) can be used to rapidly determine certain intrinsic shape characteristics of local mesh neighborhoods (see Fig. 2). Consider the elements incident to vertex i. The discrete surface around node i is characterized by the number of incident elements, and by their orientation, which determines whether their incident angle is large or small. The angular defect at vertex i is

$$\delta = 2\pi - \sum_j^{v(i)} \theta_j \,, \tag{1}$$

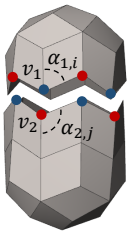
where $v(i)$ denotes the valence of vertex i and $\theta_j \in \{\alpha, \beta\}$ is the incident angle of element j. As established by the Gauss-Bonet theorem, the angular defect at a given node is equal to its integrated Gaussian curvature at that point, thus defining the *intrinsic* shape of the surface. More concretely, the angular defect determines whether the surface is locally flat ($\delta = 0$), convex or concave ($\delta > 0$), or hyperbolic ($\delta < 0$). Since the incident angles θ_j can only assume two values, α and β, the angular defect is quantized to a discrete set of values. The largest value is determined as $\delta_{max} = 2\pi - 2\alpha$, since the minimum valence is two. The smallest value δ_{min} is determined by the physical limit on how many elements can be inserted at a given vertex. For our elements and their particular interior angles, the minimum angular defect we could achieve was $\delta_{min} = 2\pi - 4\beta - \alpha$. Since smaller (more negative) angular deficits correspond to packing more elements around one vertex, we have found in practice that designs with angular defects approaching δ_{min} are increasingly difficult to fabricate; therefore, one of the objectives in our design tool is to avoid such cases.

4 Design

One of the most intriguing aspects of interlocking elements is that geometry is entirely determined by topology. However, having to think in terms of element connectivity and orientation in order to achieve a desired shape is very unintuitive for novices and arguably human users in general. We therefore take modularity to a higher level of abstraction and leverage a library of base shapes that can be instantiated, combined, and extended. In particular, we found that a wide range of designs could be easily and intuitively produced using two fundamental operations: *merging* of base shapes enables rapid drafting of a rough form; *extrusion* then plays the role of a more refined sculpting operation. Both operations can be framed as the problem of finding *interfaces* on two input shapes that can be *matched* and *merged* into a discrete structure with desired, realizable embedding.

4.1 Interface Matching and Merging

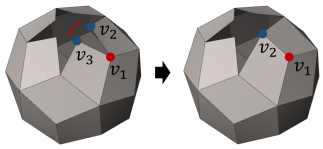
When searching for candidate *interfaces* on two shapes with given position and orientation, a key requirement is that they be *compatible*, i.e., can be merged into a discrete structure with realizable embedding. An interface $\mathcal{I} = \{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$ is a set of $|\mathcal{I}| = n$ vertices defining an edge loop that divides a given shape into two separate pieces. As shown in the inset figure, interfaces typically have a sawtooth profile with alternating valley and mountain vertices, indicated in red and blue color, respectively.



To connect two shapes along their interfaces $\mathcal{I}_1$ and $\mathcal{I}_2$, we require a *compatible* correspondence (bijection) between the vertices of $\mathcal{I}_1$ and $\mathcal{I}_2$. Assuming $|\mathcal{I}_1| = |\mathcal{I}_2|$, we seed the correspondence with the most proximate inter-interface vertex pair. Subsequent correspondences follow since the matching must be monotonic and one-on-one.

The correspondence is *compatible* if each vertex pair, when merged, would have an angular defect in the permissible range $[\delta_{\min}, \delta_{\max}]$. If the correspondence is incompatible, we reseed the correspondence process, shifting the seed pair by one step and trying again. If these new attempts fail, we continue the shifting process until a correspondence is found. In the case no correspondence can be found, we return control to the user. It is worth noting that this process is very fast such that the user can interactively drag and/or rotate one of the shapes until a solution is found.

When $|\mathcal{I}_1| \neq |\mathcal{I}_2|$, we precede the correspondence process by merging vertices. Two mountain vertices can be merged into one, thus consuming the sandwiched valley vertex and reducing the number of vertices by two. To determine which of the vertices to merge, we choose the triple that results in the largest angular defect, thus maximizing the likelihood of finding a compatible correspondence.



With the matching and merging of two given interfaces established, we can now turn to the question of how to find interfaces in the context of the two editing operations.

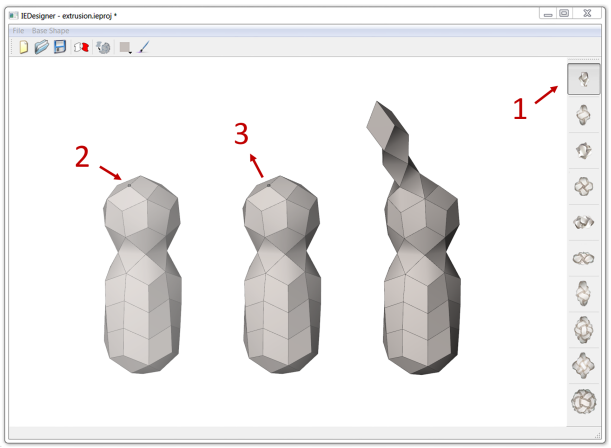
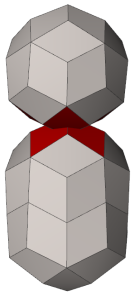


Figure 3: *After selecting a brush (1), the designer picks (2) and drags (3) a target vertex to extrude a shape.*

4.2 Merge Tool

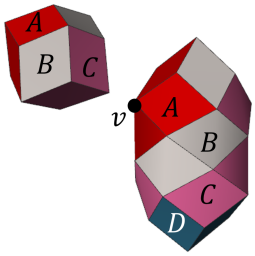
Merging is a simple way of creating new geometry by combining existing base shapes. To this end, the designer first positions and orients two input shapes in such a way that they intersect. We then seek a pair of interfaces—one on each shape—that, once merged, lead to a realizable embedding that is geometrically as close as possible to the individual shapes. In order to maintain their relative position and orientation, we would like to merge the shapes along interfaces that are close to the actual curve of intersection. As a first step, we therefore compute the set of intersecting and/or overlapping elements on both shapes (see inset figure), retaining only those adjacent to non-intersecting elements. We then restrict considerations to interfaces formed from elements that are intersecting or directly adjacent to intersecting elements. However, even in this reduced setting, the number of potential interfaces is still too large to explore all of them at interactive rates. We therefore focus on four candidate interface pairs, corresponding to two binary decisions: to retain or discard the intersecting faces on the first shape; likewise, for the second shape. The (up to four) valid results are presented to the designer for consideration. Although this selection is not exhaustive, the process is very fast such that the user can adjust the relative position and orientation of the shapes interactively until an acceptable solution is found.



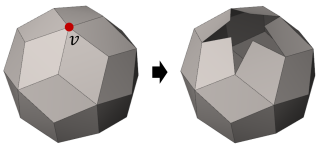
4.3 Extrusion Tool

The extrusion tool allows for a shape to be swept along a user-specified path, thus enabling the designer to quickly and easily go beyond the scope of a single merge operation. In our interface, the user first selects an *extrusion brush* from the library of base shapes, then drags on a target mesh vertex (the *extrusion apex*) to extrude a form in the drag direction with the profile of the selected brush (see Fig. 3). Similar to the merging operation, we again seek pairs of interfaces to merge the target shape and the selected brush shape. However, since we do not know the relative position and orientation of the shapes in advance, we have to determine candidate interfaces in a different way.

Layers In order to implement the extrusion metaphor, we decompose both the brush and target shape into element layers, each of which define candidate interfaces for matching and merging. We define layers inductively per vertex: for a given vertex, the first layer (base case) consists of its one-ring of incident elements; layer $n+1$ (inductive case) consists of all elements edge-adjacent to layer n. The inset figure shows a schematic view of the three layers (A,B,C) of the brush shape and the four layers (A,B,C,D) around a selected vertex on the target shape. A computational preprocess decomposes all library shapes into layers, starting the peeling process from a designer-prescribed seed vertex. The layer structure surrounding the extrusion apex is computed online.



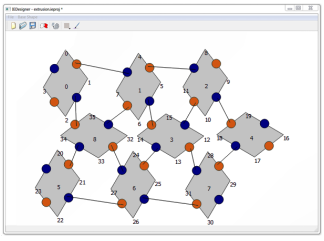
The edges between two layers on a given shape define candidate interfaces that segment the corresponding mesh into two disconnected parts. Furthermore, to provide additional possibilities for connections, we also consider *opening* the extrusion apex by removing the vertex and its incident edges to reveal another candidate interface.



With the brush and target shapes decomposed into layers, we have a number of candidate interfaces to connect to each other, and we seek to find the best pair. Since the goal of the extrusion operation is to add geometry, we first discard combinations that would decrease the number of layers of the target mesh. We then sort candidate interface pairs into groups according to the difference between the number of interface vertices. We start by testing the interfaces from the first group, i.e., with the same number of vertices. For each candidate pair, we store the minimum angular defect and return the result with the highest value, if it is larger than $\delta_{\min}$. Otherwise, we proceed to the next group.

4.4 Bootstrapping: Designing Base Shapes

Our method builds on a library of base shapes that the designer can instantiate, rigidly transform, and combine through merge and extrusion operations as described above. In order to facilitate the creation of base shapes, we provide a simple click-and-drag interface that allows designers to quickly create planar mesh layouts. The designer clicks to instantiate elements, drags them to the desired position, and rotates or flips them as desired. Each element has four ports for connections to other elements. Connections are established by dragging a free port of a long/short edge of one element onto a free port for a short/long edge of another element. Once a feasible design is created, the user can preview the geometry. This interface makes it easy to recreate existing layouts that are available online.



5 Assembly Instructions

Our forward design tool allows users to quickly create complex assemblies of interlocking elements. Without further assistance, however, constructing these models is challenging as the order in which elements are inserted has a significant impact on the difficulty of the assembly process. In order to simplify this task, we propose an automatic method that generates assembly instructions based on a set of key observations and experiences.

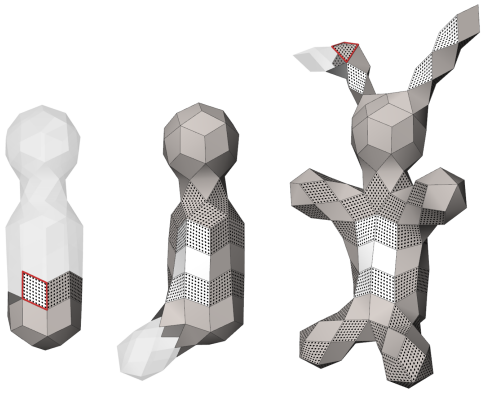


Figure 4: *Assembly instructions for the bunny at different stages of the construction process. The instructions provide information regarding back-facing elements (dotted elements), element to be inserted next (red) and shape being assembled (transparent mesh).*

Construction History We generally follow a *first-modeled-first-built* approach and assemble the components of a model in the order in which they were created by the user. In order to determine the assembly order within individual components, we leverage the layer structure of the base shapes and number the elements by spiraling upwards through the layers. It is worth noting that this approach directly applies to merged shapes and also extends to extruded geometry, which exhibit a layer structure by construction.

Sequential vs. Parallel Assembly Most shapes designed with our method exhibit a modular structure, typically with convex components and hyperbolic junctions between them. It is tempting to first build the individual components, then combine them into a single shape. However, vertices in junction regions often have negative angular defects and are thus more challenging to assemble—and fitting these junctions in between two assembled components can be virtually impossible. We therefore adopt a sequential rather than parallel element order: we start by building a given component, followed by its junctions, and only afterwards proceed to the next components.

Closing and Reopening Closed shapes are generally more stable than partly assembled structures. The reason for this is that elements tend to detach from open boundaries while the shape is being manipulated. In practice, we found it advantageous to first build a completely closed shape and reopen at the corresponding locations only when adding the next component (created through merge or extrusion operations). We therefore generate corresponding instructions to implement these closing and reopening operations. The information which of the elements to remove in what order is directly available from the construction history.

Following these principles, we generate a global insertion order for each element. We use this order to incrementally construct the model, allowing the user to follow the operations in a step-by-step manner. See Fig. 4 for an illustration as well as the accompanying video for a comprehensive example.

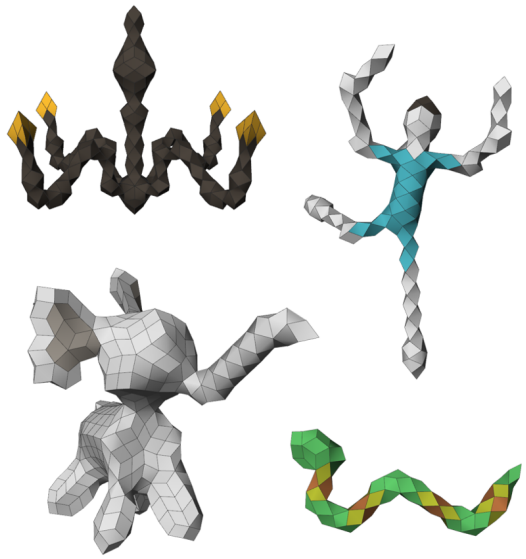


Figure 5: *An overview of additional results: Chandelier (440 elements), Dancer (215), Elephant (588), and Snake (117).*

6 Results

In order to investigate the expressive range of our design tool, we created a number of example shapes that are shown in Figs. 1 and 5. We were able to easily and quickly design very different and complex shapes, demonstrating that two simple operations—merging and extrusion—form a powerful and expressive pair of tools, able to reach diverse points in this difficult design space.

We furthermore fabricated four of these designs and found them to be in a good agreement with the shapes predicted by our interactive tool, indicating that the underlying physical model and feasibility conditions closely agree with the physics of the actual interlocking structures. In all cases, the design time was between 10 and 25 minutes and thus an order of magnitude faster than the actual fabrication time (see Table 1). This informs the claim that a trial-and-error approach to design with interlocking elements is indeed an exhausting alternative. By contrast, our interactive modeling tool allows for quick exploration of various design options, allowing the user to convergence on a virtual design before devoting time to the assembly process.

We found that the choice of which tool to use when is typically answered naturally from the context. As an example, for the bunny model (Fig. 1, *left*) we started from a spherical base shape of 30 elements. In order to fit the body's proportions, we extended the sphere along the vertical dimension using the extrusion tool with the same base shape as brush. The head was instantiated using a another spherical base shape and merged to the main body in order to create a succinct delineation between the two components. The legs, arms, and ears are again conveniently created with extrusion operations.

	Turtle	Pig	Flower	Bunny
# elements	123	187	288	188
assembly time	155	145	240	160

Table 1: *Statistics for our fabricated models. Assembly times (in minutes) are approximate.*

7 Limitations & Future Work

We have presented an interactive method for designing assemblies made from interlocking elements of a single shape. Using two basic operations, merging and extrusion, we were able to create a diverse set of shapes, each designed in a matter of minutes. The most immediate benefit of our system is that it enables an interactive, virtual exploration of the design space; to be contrasted with a trial-and-error approach in physical reality. The assembly instructions generated by our method further simplify the task of creating physical surfaces, reducing the total turnaround time to a few hours at most. Nevertheless, there are various possible improvements to our system and a number of interesting directions for future work.

In order to compute equilibrium shapes for our surfaces, we introduce simplifying kinematic assumptions that allow for computations at interactive rates. While this approach captures their main characteristics, the physical surfaces are subject to more complex conditions. A treatment in terms of nonlinear thin-shell and contact mechanics would likely yield better accuracy, albeit at the cost of a significant increase in computation times.

As an alternative to the forward design methodology that we pursued in this work, it would be interesting to explore an inverse approach in which the user specifies a target geometry to be approximated with interlocking elements. One possibility would be to build on quad remeshing algorithms such as [Tarini et al. 2010], adapted and extended to handle the unique constraints introduced by interlocking quadrilateral elements. However, since the design space is very restrictive, a major challenge would be to find adequate ways of approximating, or rather abstracting, input geometry.

Acknowledgments

We thank Bernd Bickel for initial discussions as well as Alessia Marra and Maurizio Nitti for artistic feedback.

References

CIGNONI, P., PIETRONI, N., MALOMO, L., AND SCOPIGNO, R. 2014. Field-aligned mesh joinery. *ACM Trans. Graph. 33*, 1.

CUTLER, B., AND WHITING, E. 2007. Constrained planar remeshing for architecture. In *Proceedings of Graphics Interface 2007*, 11–18.

DE GOES, F., ALLIEZ, P., OWHADI, H., AND DESBRUN, M. 2013. On the equilibrium of simplicial masonry structures. *ACM Trans. Graph. 32*, 4.

DENG, B., BOUAZIZ, S., DEUSS, M., KASPAR, A., SCHWARTZBURG, Y., AND PAULY, M. 2015. Interactive design exploration for constrained meshes. *Computer-Aided Design 61*.

DEUSS, M., PANOZZO, D., WHITING, E., LIU, Y., BLOCK, P., SORKINE-HORNUNG, O., AND PAULY, M. 2014. Assembling self-supporting structures. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 33*, 6.

EIGENSATZ, M., KILIAN, M., SCHIFTNER, A., MITRA, N. J., POTTMANN, H., AND PAULY, M. 2010. Paneling architectural freeform surfaces. *ACM Trans. Graph. 29*, 4.

GARG, A., SAGEMAN-FURNAS, A. O., DENG, B., YUE, Y., GRINSPUN, E., PAULY, M., AND WARDETZKY, M. 2014. Wire mesh design. *ACM Trans. Graph. 33*, 4.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2012. Crdbrd: Shape fabrication by sliding planar slices. *Comp. Graph. Forum 31*, 2pt3 (May), 583–592.

IGARASHI, Y., IGARASHI, T., AND MITANI, J. 2012. Beady: Interactive beadwork design and construction. *ACM Trans. Graph. 31*, 4.

KILIAN, M., FLÖRY, S., CHEN, Z., MITRA, N. J., SHEFFER, A., AND POTTMANN, H. 2008. Curved folding. *ACM Trans. Graph. 27*, 3.

LI, X.-Y., SHEN, C.-H., HUANG, S.-S., JU, T., AND HU, S.-M. 2010. Popup: Automatic paper architectures from 3d models. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4.

LI, X.-Y., JU, T., GU, Y., AND HU, S.-M. 2011. A geometric study of v-style pop-ups: Theories and algorithms. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4.

MCCRAE, J., SINGH, K., AND MITRA, N. J. 2011. Slices: A shape-proxy based on planar sections. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 30*, 6.

MITANI, J., AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graph. 23*, 3.

MORI, Y., AND IGARASHI, T. 2007. Plushie: An interactive design system for plush toys. *ACM Trans. Graph.*.

SCHWARTZBURG, Y., AND PAULY, M. 2013. Fabrication-aware design with intersecting planar pieces. *Comput. Graphics Forum (Proc. Eurographics) 32*, 2.

SKOURAS, M., THOMASZEWSKI, B., BICKEL, B., AND GROSS, M. 2012. Computational design of rubber balloons. *Comput. Graphics Forum (Proc. Eurographics) 31*, 2.

SKOURAS, M., THOMASZEWSKI, B., KAUFMANN, P., GARG, A., BICKEL, B., GRINSPUN, E., AND GROSS, M. 2014. Designing inflatable structures. *ACM Trans. Graph. 33*, 4.

SONG, P., FU, C.-W., AND COHEN-OR, D. 2012. Recursive interlocking puzzles. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 31*, 6.

SONG, P., FU, C.-W., GOSWAMI, P., ZHENG, J., MITRA, N. J., AND COHEN-OR, D. 2013. Reciprocal frame structures made easy. *ACM Trans. Graph. 32*, 4.

TARINI, M., PIETRONI, N., CIGNONI, P., PANOZZO, D., AND PUPPO, E. 2010. Practical quad mesh simplification. In *Proc. of Eurographics '10*.

VOUGA, E., HÖBINGER, M., WALLNER, J., AND POTTMANN, H. 2012. Design of self-supporting surfaces. *ACM Trans. Graph. 31*, 4.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5.

XIN, S., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3d models. *ACM Trans. Graph. 30*, 4.

YANG, Y.-L., YANG, Y.-J., POTTMANN, H., AND MITRA, N. J. 2011. Shape space exploration of constrained meshes. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 30*, 6.

ZIMMER, H., AND KOBBELT, L. 2014. Zometool rationalization of freeform surfaces. *IEEE Trans. on Visualization and Computer Graphics 20*, 10.