3D Polyomino Puzzle

Kui-Yip Lo
The Hong Kong University of Science and Technology

Chi-Wing Fu
Nanyang Technological University

Hongwei Li
The Hong Kong University of Science and Technology

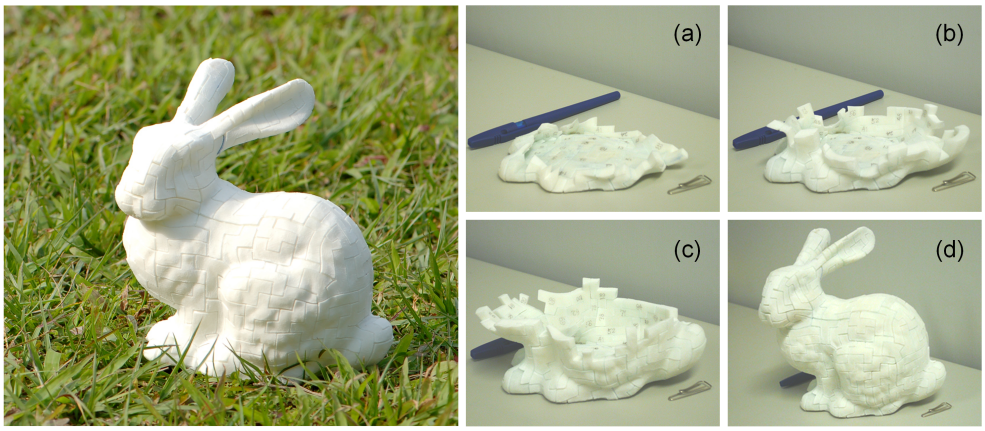


Figure 1: Left: a completed BUNNY puzzle; right: an image sequence (a-d) showing the building of BUNNY from bottom to top.

Abstract

This paper presents a computer-aided geometric design approach to realize a new genre of 3D puzzle, namely the *3D Polyomino puzzle*. We base our puzzle pieces on the family of 2D shapes known as *polyominoes* in recreational mathematics, and construct the 3D puzzle model by covering its geometry with polyomino-like shapes. We first apply quad-based surface parametrization to the input solid, and tile the parametrized surface with polyominoes. Then, we construct a nonintersecting offset surface inside the input solid and shape the puzzle pieces to fit inside a thick shell volume. Finally, we develop a family of associated techniques for precisely constructing the geometry of individual puzzle pieces, including the ring-based ordering scheme, the motion space analysis technique, and the tab and blank construction method. The final completed puzzle model is guaranteed to be not only buildable, but also interlocking and maintainable.

CR Categories: J.6 [Computer Applications]—Computer-aided design; K.8 [Personal Computing]: Games

Keywords: Computer-aided design, polyomino, puzzle

1 Introduction

Computer-aided geometric design methods have been frequently employed in games and recreational applications. One interesting example is the papercraft toy modeling application presented by Mitani and Suzuki [2004], who exploited an assortment of sophisticated computer graphics modeling techniques to turn virtual commodity 3D models into tangible papercraft toys that can be touched and manipulated. In this paper, we present an analogous idea that transforms commodity 3D models into a new genre of physically realizable 3D puzzle, the *3D Polyomino puzzle*, together with the essential family of computer modeling techniques required to construct and generate the appropriate puzzle pieces.

While picture-based 2D jigsaw puzzles are familiar and often challenging in their own right, 3D jigsaw puzzles based on 3D shapes can introduce interesting new challenges in the process of extending the puzzle geometry from 2D to 3D. However, if we survey currently available 3D puzzles, we note that the commonly employed component shapes are much the same as their 2D jigsaw puzzle counterparts. Our objective is to go beyond traditional jigsaw puzzle shapes and to exploit computer-aided geometric design techniques to generalize a particular family of 2D shapes, known as *polyominoes* in recreational mathematics, and employ them as the component shapes in 3D puzzles. A polyomino is a generalization of a domino constructed by connecting n squares edge-to-edge instead of the two squares of an ordinary domino. These shapes have been known for a long time, though the nomenclature was not coined until fairly recently by Golomb [1954; 1994] in a talk at the Harvard Mathematical Club in 1953. Golomb himself did pioneering research on this subject, followed by Gardner [1957], who wrote several interesting articles in *Scientific American*, thus popularizing the subject of polyominoes.

Starting with a designated set of polyominoes, we can tile 2D planar regions with polyominoes in the set; such a tiling is called *polyomino tiling*. One well-known example is the popular video game *Tetris* invented by the Russian Mathematician Pajitnov in 1985; it uses the five distinct shapes of tetromino (see Figure 2 (left)). Moreover, commercial products using polyominoes are also available, see the pentomino-based tiling game in Figure 2 (right). Recreational mathematicians have studied polyomino tiling for decades, and have investigated a variety of polyomino tiling problems [Golomb 1994; Putter 1998]. Recently, Ostromoukhov [2007] applied polyomino tiling to sampling and demonstrated various applications, including hierarchical importance sampling.

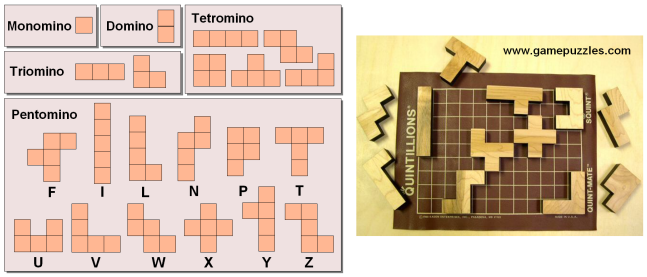


Figure 2: Left: Different types of polyomino; right: Commercial product Quintillion®, courtesy of Kadon Enterprises, Inc.

By employing a series of computer-aided geometric modeling methods, this paper utilizes polyominoes to build a new genre of 3D puzzle model, the *3D Polyomino puzzle*. We adapt Polyomino shapes to provide tilings on *parametrized surfaces*:

- A valid polyomino tiling on a non-planar parametrized surface domain is a segmentation of the parametrized space into disjoint regions, where each disjoint region is flattenable and continuously mappable to a *certain polyomino shape* in a chosen polyomino tile set.

Polyomino tiling could be well-extended to parametrized domains, such as the polycube surface, but could typically be distorted after being mapped onto the actual geometry surface. Hence, after the surface mapping, the distorted polyomino shapes may no longer be perfect polyominoes. Though this phenomenon is unavoidable given the nature of surface mapping, this, however, helps to introduce variation to the puzzle piece geometry. Such a scenario parallels that in conventional 2D jigsaw puzzles, where the shape of 2D jigsaw puzzle pieces is usually different in the puzzle set so as to avoid mis-placement of puzzle pieces.

Objectives in constructing the puzzle models. To put this new genre of 3D puzzle into practice, we compile the following objectives on the construction of the puzzle model geometry:

- *Shape* – The exposed surface of each puzzle piece has to take the shape of a certain polyomino in a given polyomino tile set (deformed in the parametrized surface).
- *Thickness* – The constructed puzzle pieces have to be thick enough to support neighboring puzzle pieces.
- *Non-intersecting* – Neighboring (thickened) puzzle pieces should precisely touch each other without intersection.
- *Buildable* – The puzzle is buildable and can be assembled with a bottom-up building order. We define a puzzle piece as *buildable* if it can be inserted into the partially-completed puzzle model from the outside without being blocked by other pieces that have been inserted previously.

Contributions. The proposed puzzle modeling framework introduces several novel elements that generalize beyond the explicit recreational context:

- The 3D polyomino puzzle is a new idea introduced in this paper, with various new concepts such as non-planar tiling on parametrized surface, mapping ambiguity, etc.;
- The work cohesively exploits computer-aided geometric design methods to create a 3D puzzle model, introducing a number of original geometric construction ideas, including buildability of puzzle pieces, motion space analysis, and shape optimization.
- A physical realization of a typical puzzle model has been built. This demonstrates and verifies the feasibility of the various proposed techniques.

Related work: Tangible toy models and artwork. After Mitani and Suzuki [2004] created papercraft toy models using a strip-based method to guide the cutting and unfolding of triangular meshes, Shatz [2006] designed papercraft models by segmenting 3D meshes into developable approximations, and Massarwi [2007] made papercraft models using generalized cylinders as the developable surfaces. Other than papercraft models, Mori and Igarashi [2007] developed an interactive user interface system that allows users to efficiently design and create plush toys, while Weyrich et al. [2007] developed a modeling system to create bas-reliefs from 3D models.

2 Polyomino Tiling on Puzzle Models

To build a 3D Polyomino puzzle, the first two steps are 1) parameterize the surface of an input solid, and 2) tile the parametrized surface with polyominoes. This section details these two steps.

Surface parametrization and dual graph. Among the surface parametrization methods [Eck et al. 1995; Tarini et al. 2004; Floater and Hormann 2004; Clarenz et al. 2004; Boier-Martin et al. 2004; Dong et al. 2006; Ray et al. 2006; Tong et al. 2006] available to construct quad-based manifolds, we select those appropriate to our puzzle modeling objectives: To the extent that it is feasible, we require the quad-based models to retain the square shape of the generated quads with minimal stretching and shape distortion. In practice, we typically employ surface parametrization results from [Tarini et al. 2004; Ray et al. 2006]. Next, we create a dual graph, say G, over the surface, where each node in G corresponds to a quad region on the parametrized surface (see Figure 3).

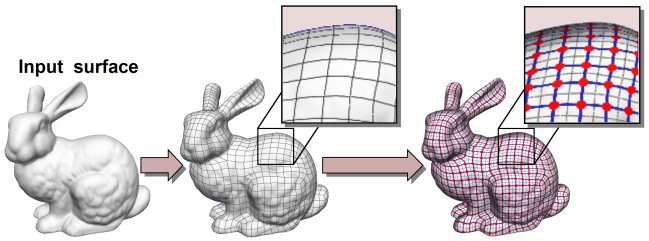


Figure 3: From left to right: the input model BUNNY, a surface parametrization, and an example dual graph.

Avoiding mapping ambiguity. Now, we can perform polyomino tiling on 3D models by decomposing G into N-node sub-graphs, where N denotes the number of squares in the employed polyomino shape set. Since our tiling space is no longer planar, we must avoid certain mapping ambiguities when mapping shapes on the parametrized surface to the polyomino shapes in a given tile set. Figure 4 demonstrates some example ambiguous cases — *Case 1*: The three interconnected nodes labeled as A, B, and C on the green tile prevent us from mapping the tile to any pentomino shape; *Case 2*: Similarly, the blue tile with a ring of five nodes also cannot be

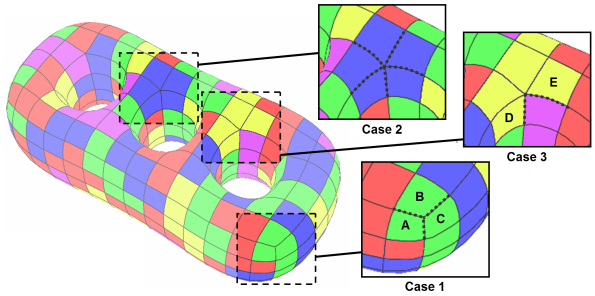


Figure 4: Examples of mapping ambiguities.

flattened and mapped to any pentomino shape; *Case 3*: Flattening the yellow tile will undesirably connect nodes D and E; though it appears to be a P pentomino, the surface region is not continuously mappable to any polyomino shape.

Node reachability. To facilitate a more efficient polyomino tiling so that we can avoid partitioning G when generating each tile on the parametrized surface; here we define the following *node reachability* term, $R_k(x)$, where x is a node in G:

$$\begin{cases} R_1(x) = \text{the number of } y_i \\ R_{k+1}(x) = R_k(x) + \alpha^k \sum_i R_k(y_i) \text{ where } y_i\text{'s are neighbors of } x,\ y_i \in G, \text{and } k \geq 1 \ . \end{cases}$$

Note that α is a weight factor, typically chosen as $\frac{1}{5}$, and we recursively compute R_5 for nodes in G for our standard case that uses pentominoes. Since polyomino tiling is done by sequentially removing N-node subgraphs from G (see Procedure 1 for the detail), during the running of the procedure, nodes with small R values in G are mostly bordered by previously-selected polyominoes. Guided by $R_k(x)$, we can quickly identify nodes along narrow branches in G in the selection of N-node sub-graphs. Therefore, we can reduce the chance of partitioning G and hence accelerate the tiling process.

Generating a polyomino tiling pattern. Procedure 1 shows the pseudo-code for tiling polyominoes on a parametrized surface. Note that conventional polyomino tilers [Golomb 1994; Putter 1998] available in the public domain are mainly designed for tiling bounded 2D planar regions, but not for manifolds of non-planar topology, such as parametrized surfaces. In addition, it is also worthwhile to note that the goal of this step is to generate a polyomino tiling pattern for making up the 3D puzzle, rather than solving a tiling problem as conventional polyomino tilers do.

Procedure 1 POLYOMINO_GENERATION (G)

```
polyomino_list = ∅
reachability_info = initialize_node_reachability ( G )
fail_count = 0

/* N=5 for pentomino tiling */
while num_nodes in G ≥ N do
  /* randomly pick a node among the nodes with very low reachability */
  n = get_node ( reachability_info )

  /* schematically generate a candidate polyomino starting from node n */
  p = generate_polyomino ( G , reachability_info , N , n )

  /* make sure p is a valid polyomino */
  if p ≠ ∅ AND NOT ambiguity ( p ) AND NOT disconnected ( G , p ) then
    update_reachability_info ( reachability_info , G , p )
    G = G - p
    polyomino_list = polyomino_list ∪ p
    fail_count = 0
  else
    /* do backtracking */
    if fail_count > MAX_FAIL_COUNT then
      undo_polyomino( G , polyomino_list )
      reachability_info = initialize_node_reachability ( G )
      fail_count = 0
    else
      fail_count = fail_count + 1
    end if
  end if
end while
return polyomino_list
```

Basically, this procedure takes G as input and sequentially generates polyominoes. The subroutine *ambiguity()* avoids ambiguous polyominoes by checking whether the candidate polyomino p can be flattened into a unique polyomino shape, whereas the subroutine *disconnected()* prevents G from being partitioned by the candidate polyomino p. Furthermore, in case we cannot find a valid polyomino after a number of runs in the *while* loop, backtracking will be carried out to undo previously-selected polyominoes that are neighbors of nodes in the current G. If the number of nodes is not a multiple of N, we construct a single polyomino with the remaining nodes. Note that since the problem of polyomino tiling is undecidable by nature, we employ backtracking in our tiling procedure (see Chapter 4 of [Golomb 1994]).

3 Constructing the Puzzle Pieces

After the polyomino tiling, we can start constructing polyomino puzzle pieces. To accomplish this task, we propose the following five steps by exploiting various computer modeling techniques: 1) constructing an offset surface beneath the model surface; 2) building the dependency graph; 3) constructing the ring-based bottom-up building order for puzzle pieces; 4) modeling the geometry of the puzzle pieces; and 5) generating tabs and blanks on puzzle pieces.

3.1 Constructing the offset surface

In constructing the offset surface for a shell volume to hold the puzzle pieces, certain considerations have to be taken into account regarding the puzzle modeling objectives. First, the shell volume between the offset surface and model surface has to be *sufficiently thick* to hold neighboring puzzle pieces against each other. Second, the offset surface should not have *self-intersections*, so that the puzzle pieces carved from it are glitch-free. Finally, the offset surface has to be *smoothly parametrized* to facilitate efficient computation in subsequent puzzle-piece construction.

To address these issues, we base our procedure on the shell construction algorithm by Peng et al. [2004], and employ the localized L_p-averaged distance function to compute the distance from any point $\mathbf{x}$ in a given 3D solid to the surface of the solid:

$$d_p(\mathbf{x}, M) = \left[\frac{1}{N(\mathbf{x}, M, R)} \sum_{\mathbf{y} \in M, |\mathbf{y}-\mathbf{x}| \leq R} |\mathbf{y} - \mathbf{x}|^{-p} \right]^{-1/p},$$

where M is the set of grid points on the parametrized surface, R is the radius of a local bounding sphere at $\mathbf{x}$, $N(\mathbf{x}, M, R)$ counts the number of grid points in the local bounding sphere (centered at $\mathbf{x}$), and p is chosen to be 10. By iteratively descending the parametrized surface along the gradient of this distance field, we can properly create a smooth and non-intersecting offset surface inside a given 3D model (see Figure 5). Furthermore, to speed up the performance in locating points in a given local sphere, we pre-compute a K-d tree structure to store points in M.

3.2 Building the dependency graph

Rather than having an arbitrary puzzle building order, we assume a bottom-up building order in the game play because it is a natural and reasonable way for building up a physical 3D puzzle. To construct a bottom-up order, we first have to construct an auxiliary data structure, the dependency graph.

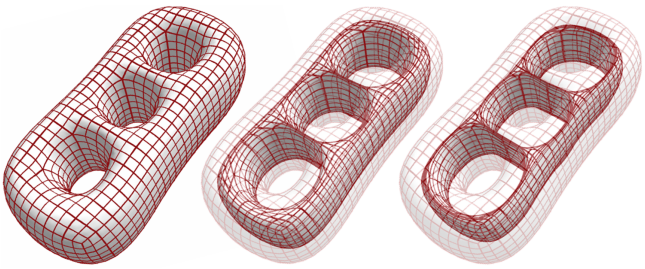


Figure 5: Creating an offset surface: HOLES3 (left to right).

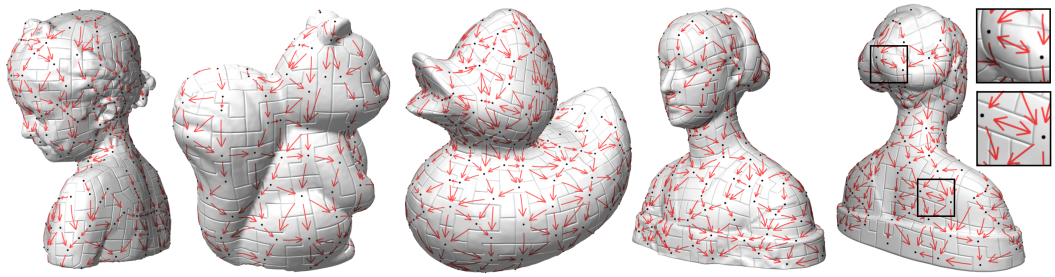


Figure 6: Dependency graphs constructed on four different 3D puzzle models.

The dependency graph is a directed graph describing the building-order dependency (positional relationship) among the puzzle pieces. Each puzzle piece is a node in the dependency graph and a directed edge, say $p \rightarrow q$, is added to the graph if puzzle piece p is on top of puzzle piece q, i.e., q is assumed to be placed into a partially-completed puzzle model before p. In practice, we compute the averaged normal along each polyomino edge shared between neighboring polyominoes p and q on the actual geometry surface, and check to see if this normal points horizontally within a user-defined threshold, typically set to be 45 degrees upward or downward in our implementation. Figure 6 shows example dependency graphs constructed on four different puzzle models: BIMBA, SQUIRREL, DUCK, and LAURANA (from left to right). Red arrows indicate the constructed directed edges. In some rare cases, such as a U-shaped polyomino on a vertical face (see the rightmost zoom-in windows in Figure 6), we could have cycles in the dependency graph. To get rid of cycles (which could break the subsequent puzzle building steps), we compute centroids of puzzle pieces in the related cycle and remove any directed edge that goes from a lower puzzle piece to a upper puzzle piece.

3.3 Ring-based bottom-up order

Given N puzzle pieces, we in general have $N!$ different orders for building a puzzle model. This, however, is not true when considering a bottom-up building order: if we apply topological sort to the dependency graph, we may end up having one building order only. To devise a *bottom-up and yet flexible* building order, we devise the following ring-based ordering scheme that partitions the N puzzle pieces into K ordered and disjoint subsets, say S_0, S_1, ..., S_{K-1}, with the following properties:

- First, we should be able to insert puzzle pieces from the same ring into a partially-completed puzzle model *in any order*. In other words, puzzle pieces should not block the insertion of any other puzzle piece in the same ring.
- Second, when we insert a puzzle piece in S_i ($i > 0$), we assume that all puzzle pieces in L_{i-1} *may* have already been inserted into the partially-completed puzzle model, where $L_i = \cup_{j=0}^{i} S_j$, and at the same time, none of the puzzle pieces in U_{i+1} have been inserted into the partially-completed puzzle model, where $U_i = \cup_{j=i}^{K-1} S_j$. In other words, it means that we should always be able to insert a puzzle piece in S_i into the partially-finished puzzle model in the presence of any puzzle piece in L_{i-1}, but not U_{i+1}.

In our implementation, we generate rings as follows:

1. First, we need to select puzzle piece(s) in S_0. It can be a single puzzle piece on the bottom of the puzzle model, or it can be a set of bottom-most puzzle pieces as in LAURANA (with an opening on the bottom, see the rightmost model in Figure 6). However, puzzle pieces in S_0 should not depend on any other puzzle piece (including S_0) so that we can guarantee an arbitrary building order for puzzle pieces within S_0.
2. Then, we can move on to the next ring and generate a candidate set for S_i ($i > 0$), say $\hat{S}_i$:
   $\{\, p : p \notin L_{i-1}$ and $\exists q \in L_{i-1}\ s.t.\ p$ and q are neighbors $\}$.
3. Next, to comply with the two ordering properties of rings, we may have to take out some puzzle pieces from the candidate set $\hat{S}_i$ by checking the dependency graph we previously constructed. In detail, given a puzzle piece, say $p \in \hat{S}_i$,
   $$\text{If } p \rightarrow q \text{ (in dependency graph) and } q \notin L_{i-1},$$
   then we have to remove p from $\hat{S}_i$. After this removal step, all puzzle pieces left in S_i are independent of each other (property 1) and they may only depend on puzzle pieces in L_{i-1} (property 2). Furthermore, it is worth noting that since the dependency graph we created previously is cycle-free, we can always obtain a non-empty S_i.
4. After that, we can then apply the motion space analysis technique (see the next subsections) to further construct the 3D geometry of puzzle pieces in S_i.
5. Finally, we go back to step 2 to construct S_{i+1} if L_i does not yet contain all puzzle pieces in the model.

Figure 7 shows an example ring construction sequence on LAURANA; puzzle pieces in S_0 to S_5 are colored in red, green, blue, cyan, magenta, and yellow, correspondingly, and we recycle the colors for successive rings.

3.4 Modeling the puzzle pieces

After creating the offset surface, one straightforward way to construct 3D models of puzzle pieces is by directly applying the one-to-one mapping between the model surface and offset surface (resulting from the shell construction) to carve puzzle pieces out of the shell volume. However, we found in practice that puzzle pieces created in this manner may not be *buildable*; here, we define the *buildability* of puzzle pieces based on the ring property:

> A puzzle piece, say $p \in S_i$, is said to be *buildable* if we can insert it into a partially-completed puzzle model in the presence of all puzzle pieces in $L_i - \{p\}$.

Note that we had to define buildability in this way so that we could properly include the case of S_0. Hence, if all puzzle pieces are buildable, we can guarantee that all puzzle pieces of the same ring do not block the insertion of one another and the puzzle building order can be bottom-up. Furthermore, to make each puzzle piece buildable, we have to take further steps to optimize the puzzle pieces' geometry. To support such an optimization, we develop the following modeling scheme to shape individual puzzle pieces:

> Let M_j be the set of grid points on the parametrized surface corresponding to the jth polyomino puzzle piece, $\tilde{M}_j$ be the set of points in M_j along the boundary of the polyomino shape, and E_j be the set of edge segments that connect neighboring points in $\tilde{M}_j$ (see below).

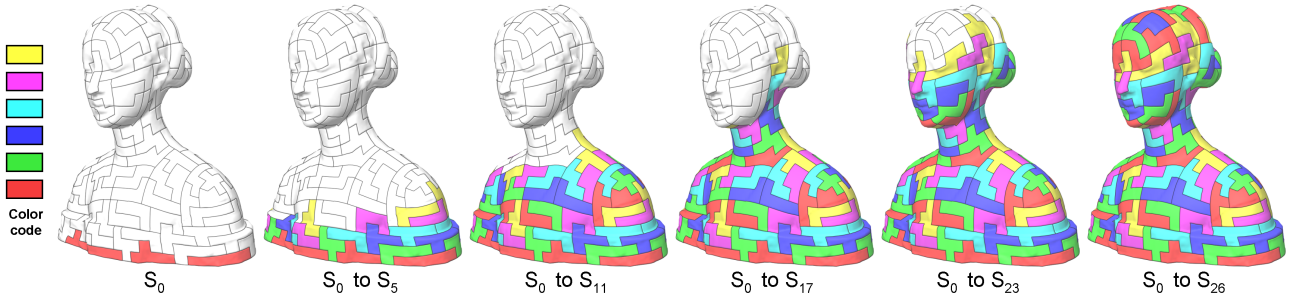
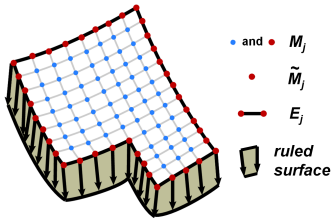
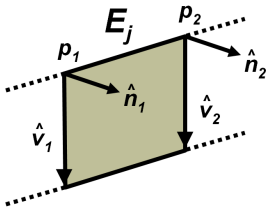


Figure 7: Bottom-up building order: the twenty-seven rings constructed on the LAURANA puzzle.

Now, at each boundary point, say $p_k \in \tilde{M}_j$, we can define a unit vector, say $\hat{v}_k$, to ray-trace from p_k to obtain an intersection point on the offset surface. Hence, we can interpolate $\hat{v}_k$ between point pairs in E_j, and create a *ruled surface* corresponding to edge segment to carve puzzle pieces out of the shell volume. In addition, to ensure that the carved 3D model to be glitch-free, we perform a ray-surface intersection test in the optimization process, over sampled line segments on the ruled surface for each edge segment in E_j.



Buildability of puzzle pieces. The ruled surface modeling method not only helps to define the way we carve puzzle pieces out of the shell volume, it in fact also helps to formulate the mechanism for computing the buildability of puzzle pieces. By definition, if a given puzzle piece, say $q \in S_i$, is buildable, then q can be the last puzzle piece to be inserted into its place after all the other puzzle pieces in L_i. At the same time, it also means that we can take it out of the puzzle model without being obstructed by puzzle pieces in L_i. Thus, if a puzzle piece is buildable, we should be able to determine at least one *motion vector* (a 3D translation vector), to translate it out of its place in the presence of other puzzle pieces in L_i. Noticing the fact that a given puzzle piece could have numerous possible motion vectors (or none), we can form a trajectory space, namely the *motion space*, to include all these vectors for a given puzzle piece; note that the motion space can be interpreted as a spherical region on the unit sphere. Hence, a puzzle piece is buildable if and only if it has a non-empty motion space.



With the ruled surface modeling scheme, we can analytically compute the motion space, and hence determine the buildability of each puzzle piece. Supposed that an edge segment, say p_1-p_2 in the edge segment set E_j along the polyomino boundary of a given puzzle piece in S_i (p_1-p_2) is shared with some other puzzle pieces in L_i), with associated end-point vectors, $\hat{v}_1$ and $\hat{v}_2$, respectively. Since puzzle piece carving at this edge segment is done by the ruled surface swept between $\hat{v}_1$ and $\hat{v}_2$, we can define the end-point normals on the ruled surface to be $\vec{n}_1 = (p_1 - p_2) \times \hat{v}_1$ and $\vec{n}_2 = (p_1 - p_2) \times \hat{v}_2$, along the two straight lines from p_1 and p_2, respectively; also, we define $\hat{n}_1$ and $\hat{n}_2$ to be the normalized vectors of $\vec{n}_1$ and $\vec{n}_2$, respectively. Note that these normals are defined to point outward from the related ruled surface on the puzzle piece.

Case 1: If $\hat{n}_1 = \hat{n}_2$, the ruled surface extended from p_1-p_2 is simply a 3D plane. Hence, when taking the related puzzle piece out of its place (in the presence of the corresponding puzzle piece on the other side of p_1-p_2), the carving plane acts like a flat wall that keeps us from moving the puzzle piece towards the $\hat{n}_1$ side. Geometrically, if Ω is the motion space of the puzzle piece,

$$\forall\, \vec{v} \in \Omega,\ \hat{n}_1 \cdot \vec{v} \leq 0\,.$$

Case 2: If $\hat{n}_1 \neq \hat{n}_2$, the normals along the sweeping lines on the ruled surface can be computed by spherical linear interpolation between $\hat{n}_1$ and $\hat{n}_2$: $\hat{n}(t) = slerp(\hat{n}_1, \hat{n}_2, t)$ where $t \in [0, 1]$. Hence, rather than having one single normal as in case 1, we now have a family of normals on the ruled surface to bound the motion space:

$$\forall\, \vec{v} \in \Omega,\ \hat{n}(t) \cdot \vec{v} \leq 0\,.$$

Furthermore, by expanding the slerp formulae (since $\hat{n}(t)$ is a great circle arc), we can prove that

$$\hat{n}_1 \cdot \vec{v} \leq 0 \ \wedge\ \hat{n}_2 \cdot \vec{v} \leq 0 \ \Rightarrow\ \hat{n}(t) \cdot \vec{v} \leq 0\,.$$

Hence, we can simplify the motion space constraint in case 2 as (again in the presence of the corresponding puzzle piece on the other side of p_1-p_2):

$$\forall\, \vec{v} \in \Omega,\ \hat{n}_1 \cdot \vec{v} \leq 0 \text{ and } \hat{n}_2 \cdot \vec{v} \leq 0\,.$$

Summarizing the above cases, we can see that each edge segment is actually associated with a certain spherical region on the unit sphere that bounds the puzzle piece's motion space. Geometrically, case 1 yields a hemisphere that is opposite to $\hat{n}_1$, while case 2 yields a spherical wedge opposite to both $\hat{n}_1$ and $\hat{n}_2$. Hence, given Ω_k as the spherical region for the kth edge segment in E_j, we can determine the motion space Ω of a given puzzle piece, say $q \in S_i$, by computing the intersection of all Ω_k's that belong to edge segments shared between q and all other puzzle pieces in L_i. In other words, if a puzzle piece is buildable, its Ω should be a non-empty region (typically, a convex spherical polygon) on a unit sphere (domain of the motion space).

Optimizing the shape of puzzle pieces. While optimizing puzzle pieces to be buildable, we have to enforce the following three constraints on the shape of puzzle pieces:

- *Intersection constraint* – Ensuring that the extended ruled surface from each edge segment can properly intersect the offset surface.
- *Integrity constraint* – Ensuring that the extended ruled surfaces from two different edge segments do not intersect.
- *Smoothness constraint* – Enforcing the smoothness of neighboring ruled surfaces along the boundary of puzzle pieces.

In the current implementation, we apply simulated annealing as the optimization kernel and employ the mapping resulting from the shell construction algorithm as our initial condition. We have the following operators to iteratively alter the direction of the $\hat{v}_k$'s so as to adjust the extended ruled surfaces:

- *Push operator* – Given a non-buildable puzzle piece, say $q \in L_i$, this operator first predicts a potential motion vector, $\hat{m}$, by taking the spherical average [Buss and Fillmore 2001] over all

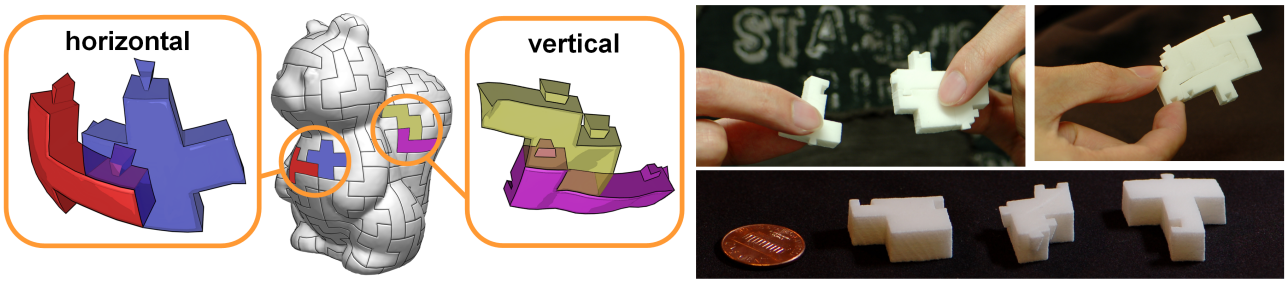


Figure 8: Tabs and blanks on puzzle pieces: horizontal and vertical tabs and blanks (left) and close-up views of some physical puzzle pieces.

end-point vectors, $\hat{v}_k$'s, along a puzzle piece's boundary. For each edge segment p_k-p_{k+1} in E_j, and their related motion space Ω_k (obtained by normal $\hat{n}_k$ at p_k) and Ω_{k+1} (obtained by normal $\hat{n}_{k+1}$ at p_{k+1}), if $\Omega_k \cap \Omega_{k+1}$ does not contain $\hat{m}$, this operator jitters v_{k+1} so as to push $\Omega_k \cap \Omega_{k+1}$ towards $\hat{m}$ over the spherical domain of the motion space. Note that we only need to consider edge segments that are shared between puzzle piece q and other puzzle pieces in L_i.

- *Smooth operator* – This operator applies Gaussian filtering to smooth the $\hat{v}_k$'s in between corners of puzzle pieces.

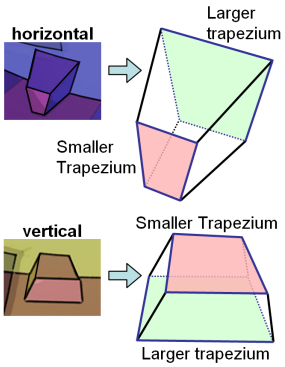
We define the guidance function (per puzzle piece) by considering each edge segment around the puzzle piece boundary:

$$W = \sum_k f\big((\hat{m} \cdot \hat{n}_k) \cdot ||p_k - p_{k+1}|| \big) ,$$

where $f(x) = (\min(x, 0))^2$. Hence, if $\hat{m} \in \Omega_k$, the contribution of the corresponding edge segment will be zero. In the optimization process, our goal is to iteratively minimize W (to zero) for each puzzle piece in the same ring. Note also that if a candidate operation results in $\hat{v}_k$ that violates the intersection or integrity constraint, the candidate operation has to be ignored.

3.5 Constructing tabs and blanks

After modeling the geometry of individual puzzle pieces, our last step is to construct tabs and blanks on puzzle pieces so that we can connect neighboring puzzle pieces. Consider two neighboring puzzle pieces, say $p \in S_i$ and $q \in S_j$, where without loss of generality, we assume $i < j$. Hence, p could be in its place while q is being inserted into the puzzle model. Therefore, the tab and blank to be constructed between p and q should not block the insertion of q. As a result, we can look solely at the motion space of q (but not that of p) when constructing the related tabs and blanks between p and q. In practice, we devise two kinds of tabs and blanks: *a horizontal pair of tab and blank* and *a vertical pair of tab and blank*, corresponding to an insertion direction that is perpendicular or tangential to the puzzle-piece surface, respectively, see Figure 8.



In detail, we model tabs and blanks by applying extrusion between two trapezium shapes with slightly different sizes, see the inset on the right. In addition, we also apply motion space analysis to compute the motion space of tabs and blanks by considering the contact planes (on the tabs and blanks) between the connecting puzzle pieces, so that we can guarantee a proper insertion of q by ensuring that the motion space intersection between that of q and that of the tab/blank is non-empty. Furthermore, it is worthwhile to note that for puzzle pieces facing upward (such as the last puzzle piece on the top), we need not arrange tabs and blanks on them.

When we model the geometry of tabs and blanks, we also apply the motion space analysis to compute the motion space of tabs and blanks (and model their geometry), so that we can guarantee a proper insertion of q by ensuring that the motion space intersection between that of q and that of the tab/blank is non-empty. Furthermore, it is also worthwhile to note that for puzzle pieces facing upward (such as the last puzzle piece on the top), we need not arrange tabs and blanks on them.

4 Implementation, Results and Discussion

Polyomino tiling. When using Procedure 1 to generate a polyomino tiling, we observe that the procedure could generate an unbalanced population of pentomino shapes, with some shapes appearing more often than the others. To address this issue and make the overall tiling appearance more natural, we use a histogram approach in practice to balance the population. Our method maintains a histogram of tiled polyomino shapes in runtime and discards shapes of high frequency at the end of the *generate_polyomino()* subroutine.

Table 1 shows the performance of Procedure 1 in tiling pentominoes on three different 3D models. The experiment was carried out on a Pentium-4 PC with 3GHz CPU and 512MB memory.

Table 1: Timing result on polyomino generation.

	LAURANA	HOLES3	SQUIRREL	BUNNY	DUCK	BIMBA
Number of tiles	48	80	159	258	263	325
Time (sec.)	0.012	0.031	0.146	0.224	0.228	0.290

Puzzle model construction. We employed our puzzle modeling methods to create six different 3D puzzle models: BUNNY, LAURANA, HOLES3, SQUIRREL, DUCK, and BIMBA. The first three are polycube-mapped models from [Tarini et al. 2004], whereas BIMBA is derived from the periodic global parametrization method [Ray et al. 2006]. In addition, we also construct surface parametrizations for two models, SQUIRREL and DUCK, using a simple polycube editing tool we developed. Figure 9 depicts the process of building 3D polyomino puzzles for these models, with one model per column. The 1st to 3rd rows in the figure correspond to polyomino tiling, offset surface, and ring-based building order, respectively. Note that polyomino tiling (1st row) is shown using five coloring while ring ordering (3rd row) is shown using the color coding scheme described in Figure 7. Furthermore, using the ring-based bottom-up scheme to formulate the buildability of puzzle pieces, we find that all the constructed puzzle pieces in the six puzzle models shown can be iteratively optimized to be buildable.

Rapid prototyping. The last stage in puzzle modeling is rapid prototyping. We employed the rapid prototyping method known as selective laser sintering (SLS) (with 0.15mm granularity), to construct individual puzzle pieces of the BUNNY puzzle model. Figure 1 shows the completed BUNNY puzzle model, together with an image sequence showing the building of the puzzle from bottom to

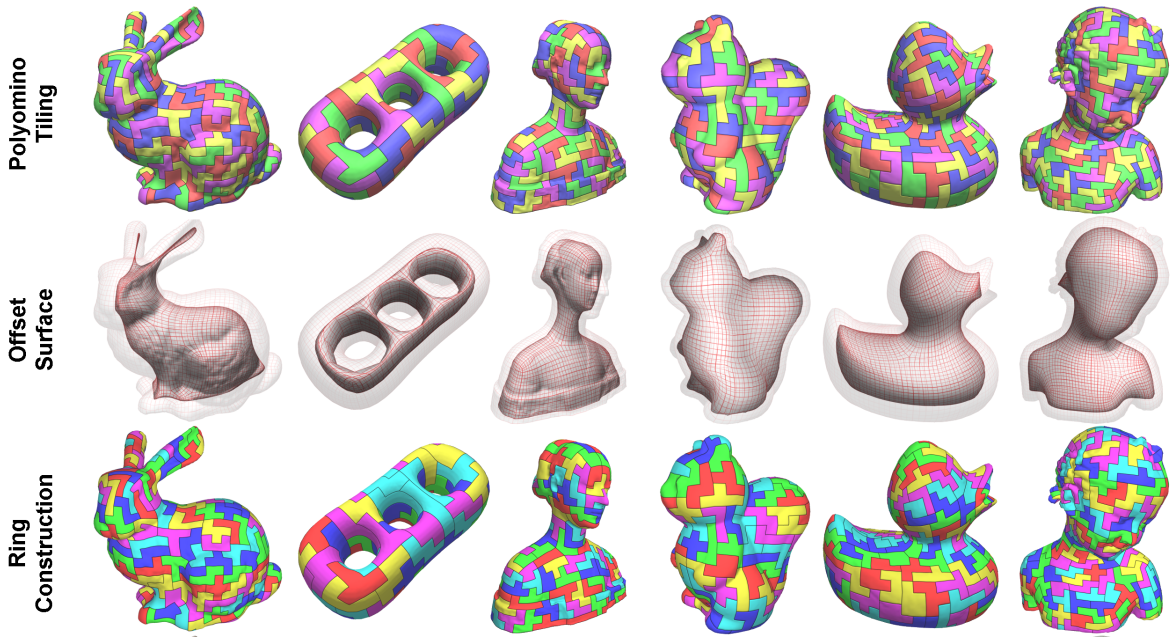


Figure 9: Procedures in the construction of 3D polyomino puzzles from BUNNY, LAURANA, HOLES3, SQUIRREL, DUCK, and BIMBA, (from left to right): polyomino tiling with five colorings (1st row), offset surface (2nd row), and ring-based bottom-up building order.

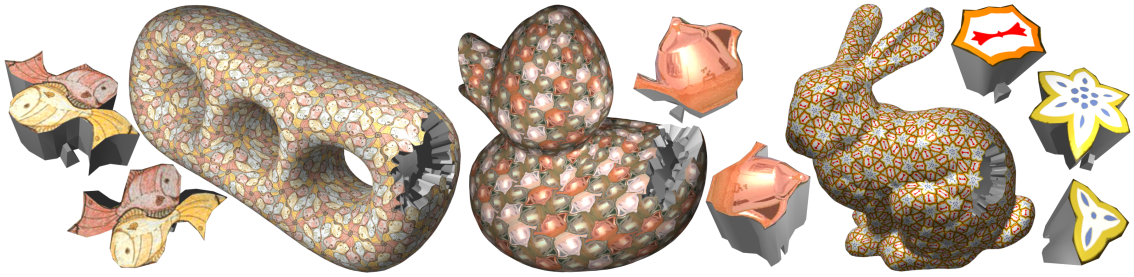


Figure 10: Puzzle models based on semiregular patterns (left to right): fish-shaped, teapot-shaped, and pattern-based puzzle pieces.

top. In addition, the right hand side of Figure 8 shows some close-up views of individual puzzle pieces and demonstrates how they are connected.

Note: Since rapid prototyping is both costly and lengthy, the completed BUNNY model shown in Figure 1 is a prototype version without tabs and blanks; the completed puzzle model of BUNNY was created using all the other methods including the tiling procedure, the offset surface, the ring-based building order, and the shape optimization, as well as the motion space analysis to ensure the buildability of individual puzzle pieces, so that we can demonstrate a bottom-up building sequence as shown in Figure 1. Hence, we additionally created some of the puzzle pieces with tabs and blanks as demonstrated in Figure 8.

Semiregular and Escher-like puzzle models. Besides polyominoes, we also experimented with extending this work using semiregular patterns [Kaplan 2007; Kaplan and Salesin 2000] as the component shapes to construct the puzzle pieces. Among the 17 symmetric groups [Alexander 1975], we were able to employ the groups p4, p4m, and p4g on quad meshes and the groups p6 and p6m on triangle meshes because they are symmetric and adaptable to the topology of the arbitrary meshes, as demonstrated by Kaplan [2007]. Figure 10 shows three constructed puzzle models: fish-shaped puzzle pieces on HOLES3 (p6), teapot-shaped puzzle pieces on DUCK (p4), and pattern-based puzzle pieces on BUNNY (p6m) with close-up views of some individual puzzle pieces.

User experience. We conducted a prototype experiment with ten users participating in a game employing eight connectable puzzle pieces. We recorded the time taken for each user to join subsequent puzzle pieces, and summarize their performance in Table 2. Due to the three-dimensional nature of the puzzle pieces, most of them found this task rather challenging.

Table 2: Time (in seconds) to join subsequent puzzle pieces.

piece	2nd	3rd	4th	5th	6th	7th	8th
min	18.0	55.2	147.1	153.9	169.0	204.4	215.3
max	208.9	438.0	458.8	1109.7	1139.1	1220.9	1293.5
mean	86.2	203.0	274.1	404.7	453.9	491.9	524.3

Discussion on building order. Early in this work, we attempted to devise a highly flexible building order with the following definition for buildability:

> A puzzle piece is buildable if we can insert it as the last puzzle piece in completing a puzzle model.

Hence, we could in principle have arbitrary building orders. However, to technically ensure a buildable puzzle model against this definition, any puzzle piece could potentially be the last puzzle piece, and we found this condition to be excessively restrictive. We experimented with this buildability definition by applying the simulated annealing method to optimize the shape of all puzzle pieces (instead of considering puzzle pieces ring by ring as in Section 3.3), but found that it is always not possible to ensure every puzzle piece to be buildable, in particular those surrounding concave regions.

We were led instead to require a bottom-up building order, which is a natural building order for models resting on a table, and can suffice to guarantee puzzle piece buildability while allowing a certain degree of building order flexibility.

Limitations.

- *Surface parametrization:* Since our method relies on an input parametrization, we require the quads in the given parametrization to be square-alike so that we can maintain the polyomino appearance for the puzzle pieces.
- *Offset surface:* Our method may not be able to handle 3D models with thin and highly curved parts because the constructed puzzle pieces may not have sufficient thickness, and the resulting shell may not be one-to-one for such regions. We must ensure a non-degenerated offset surface before proceeding to construct the puzzle pieces.
- *Polyomino tiling:* Furthermore, when constructing polyomino tilings with relatively larger numbers of squares (say > 5), it is sometimes not possible to avoid all three mapping ambiguities while attempting to generate a valid polyomino tiling on the parametrized surface, in particular for models with dense numbers of singularities. Since polyomino tiling is undecidable by nature [Golomb 1954], the tiling procedure may never finish in these extreme cases.

5 Summary

In summary, we present in this paper a computer-aided geometric design approach for constructing a new genre of 3D puzzle model, the *3D Polyomino puzzle*, and realize the design via a series of computer modeling methods, including quad-based surface parametrization for dual graph generation and polyomino tiling, the shell construction algorithm for creating an offset surface, the dependency graph and the ring-based ordering method for devising a bottom-up building order, the ruled surface modeling scheme for shaping the geometry of individual puzzle pieces, the motion space analysis technique for optimizing the buildability of puzzle pieces, and the tab and blank construction method to connect puzzle pieces. With these proposed techniques, we can create 3D puzzle models that are not only tangible, but are also buildable and playable. Finally, we employed rapid-prototyping to construct a physical puzzle model for the BUNNY puzzle as a concrete demonstration.

Future work. Our approach constructs puzzle models by decomposing model surfaces into polyomino-like shapes. Essentially, there are other possible decomposition approaches: surface-based or volume-based, as well as different forms of puzzle models. We hope that this work can stimulate further research on such topics.

Acknowledgments. We thank all reviewers for the constructive comments and suggestions that help to improve the clarity of the paper. This work is supported in part by the Nanyang Technological University Startup Grant (Project No. M58020007.500000) and the Research Grants Council of the Hong Kong Special Administrative Region (Project No.: HKUST618208).

References

ALEXANDER, H. 1975. The computer plotter and the 17 ornamental design types. In *SIGGRAPH 1975*, 160–167.

BOIER-MARTIN, I., RUSHMEIER, H., AND JIN, J. 2004. Parameterization of triangle meshes over quadrilateral domains. In *Proc. of Eurographics/ACM SIGGRAPH symp. on Geom. proc.*, 193–203.

BUSS, S. R., AND FILLMORE, J. P. 2001. Spherical averages and applications to spherical splines and interpolation. *ACM Tran. on Graphics 20*, 2, 95–126.

CLARENZ, U., LITKE, N., AND RUMPF, M. 2004. Axioms and variational problems in surface parameterization. *Computer Aided Geometric Design 21*, 8, 727–749.

DONG, S., BREMER, P.-T., GARLAND, M., PASCUCCI, V., AND HART, J. C. 2006. Spectral surface quadrangulation. *ACM Tran. on Graphics (Proc. of SIGGRAPH) 25*, 3, 1057–1066.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *Proc. of ACM SIGGRAPH 95*, 173–182.

FLOATER, M. S., AND HORMANN, K. 2004. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*, Springer, N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds., 259–284.

GARDNER, M. 1957. Mathematical games: About the remarkable similarity between the icosian game and the towers of Hanoi. *Scientific American 196*, 150–156.

GOLOMB, S. W. 1954. Checkerboards and Polyominoes. *The American Mathematical Monthly 61*, 10 (Dec), 287–294.

GOLOMB, S. W. 1994. *Polyominoes: Puzzles, Patterns, Problems, and Packings*. Princeton University Press. Revised edition.

KAPLAN, C. S., AND SALESIN, D. H. 2000. Escherization. In *Proc. of ACM SIGGRAPH 2000*, 499–510.

KAPLAN, C. S. 2007. Semiregular patterns on surfaces. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Sketches*, 78.

MASSARWI, F., GOTSMAN, C., AND ELBER, G. 2007. Papercraft models using generalized cylinders. In *5th Pacific Conf. on Comp. Graphics and App.*, 148–157.

MITANI, J., AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Tran. on Graphics (Proc. of SIGGRAPH) 23*, 3, 259–263.

MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. *ACM Tran. on Graphics (Proc. SIGGRAPH) 26*, 3. Article 45.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Tran. on Graphics (Proc. SIGGRAPH) 26*, 3. Article 78.

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. *ACM Tran. on Graphics (Proc. of SIGGRAPH) 23*, 3, 635–643.

PUTTER, G., 1998. Gerard's universal polyomino solver 1.4. `http://www.xs4all.nl/~gp/PolyominoSolver/Polyomino.html`.

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

SHATZ, I., TAL, A., AND LEIFMAN, G. 2006. Paper craft models from meshes. *Visual Computer 22*, 9, 825–834.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. *ACM Tran. on Graphics (Proc. of SIGGRAPH) 23*, 3, 853–860.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing quadrangulations with discrete harmonic forms. In *Proc. of Eurographics symp. on geom. proc.*, 201–210.

WEYRICH, T., DENG, J., BARNES, C., RUSINKIEWICZ, S., AND FINKELSTEIN, A. 2007. Digital bas-relief from 3D scenes. *ACM Tran. on Graphics (Proc. of SIGGRAPH) 26*, 3. Article 32.