Inverse Tiling of 2D Finite Domains

RULIN CHEN, Singapore University of Technology and Design, Singapore and BNBU, China
XUYANG MA, Singapore University of Technology and Design, Singapore
PRAVEER TEWARI, Singapore University of Technology and Design, Singapore
CHI-WING FU, The Chinese University of Hong Kong, Hong Kong
PENG SONG, Singapore University of Technology and Design, Singapore

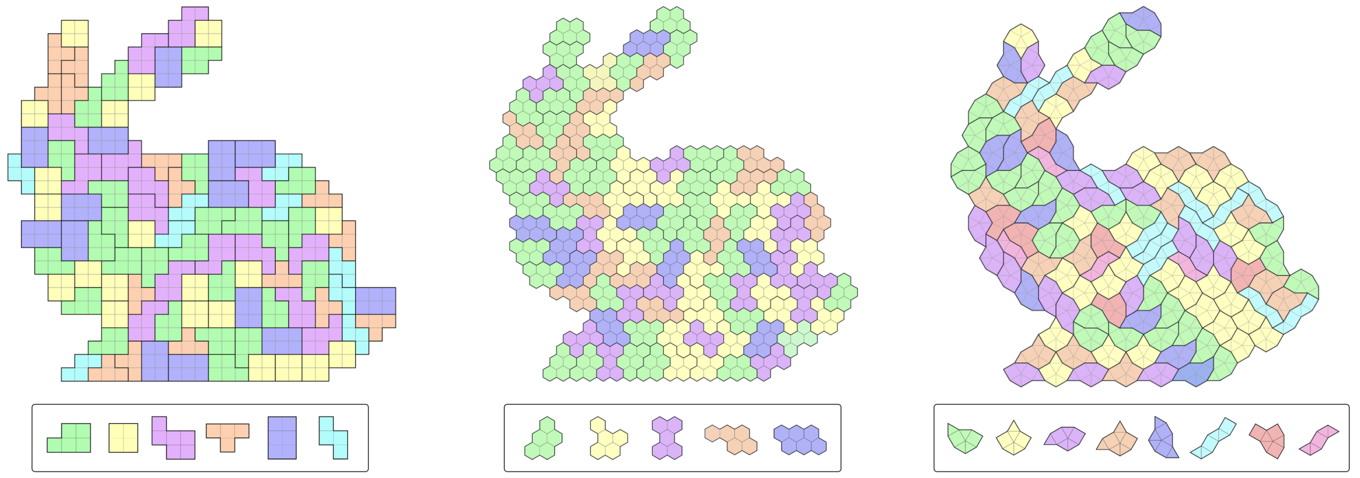


Fig. 1. Three K-hedral tiling results produced by our inverse tiling approach. To effectively tile each of the 2D finite domains shown above, our approach constructs a minimized set of distinct tile shapes called prototiles; see the box below each tiling result. In particular, our approach is versatile. It can handle domains represented by various forms of grid, e.g., a square grid (left), a hexagon grid (middle), and a square-triangle grid (right).

A K-hedral tiling of a 2D finite domain is a covering of the domain with tiles without gaps or overlaps, where each tile is congruent to one of the K distinct shapes called *prototiles*. K, the number of prototiles, is preferred to be as small as possible for congruent tiling appearance and reducing fabrication cost, e.g., by molding. Typically, a forward approach is adopted to produce K-hedral tilings by prescribing a set of prototiles and placing prototile instances (i.e., tiles) to cover the input domain. However, the prescribed prototile set may not be sufficient to tile the domain (for small K) or may lead to tiling results with excessive prototiles more than needed (for large K).

In this work, we formulate a new tiling problem called *inverse tiling* for producing K-hedral tilings in 2D finite domains, where the prototile set is *inversely modeled* to fit the input domain instead of being prescribed. Since the prototile set is unknown, inverse tiling allows exploring a large search space to discover a minimized number of prototiles for tiling the input domain. To solve the inverse tiling problem, we propose a computational approach that progressively builds the prototile set while tiling the input domain, starting from a prototile set with a single element. Once a tiling result is obtained, the approach further tries to reduce the number of prototiles by locally re-tiling the input domain to eliminate prototiles with few instances. We demonstrate the effectiveness of our inverse tiling approach on a variety of finite domains, evaluate its performance in scalability and K minimization, and compare it quantitatively with forward tiling approaches.

CCS Concepts: • **Computing methodologies** → *Shape modeling*.

Additional Key Words and Phrases: K-hedral tiling, inverse tiling, polyform tiling, geometry modeling

ACM Reference Format:
Rulin Chen, Xuyang Ma, Praveer Tewari, Chi-Wing Fu, and Peng Song. 2025. Inverse Tiling of 2D Finite Domains. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3757377.3763834

Authors' Contact Information: Rulin Chen, 1998rlchen@gmail.com, Singapore University of Technology and Design, Singapore and BNBU, China; Xuyang Ma, xuyang_ma@mymail.sutd.edu.sg, Singapore University of Technology and Design, Singapore; Praveer Tewari, tewaripraveer@gmail.com, Singapore University of Technology and Design, Singapore; Chi-Wing Fu, philip.chiwing.fu@gmail.com, The Chinese University of Hong Kong, Hong Kong; Peng Song, peng_song@sutd.edu.sg, Singapore University of Technology and Design, Singapore.

1 Introduction

A tiling is a coverage of a domain using a countable set of regions called tiles, with no overlaps and no gaps [Grünbaum and Shephard 2016]. A *K-hedral tiling* is a tiling, in which every tile is congruent to one of K distinct shapes called *prototiles*. The K-hedral tiling of an infinite domain such as the Euclidean plane has been well

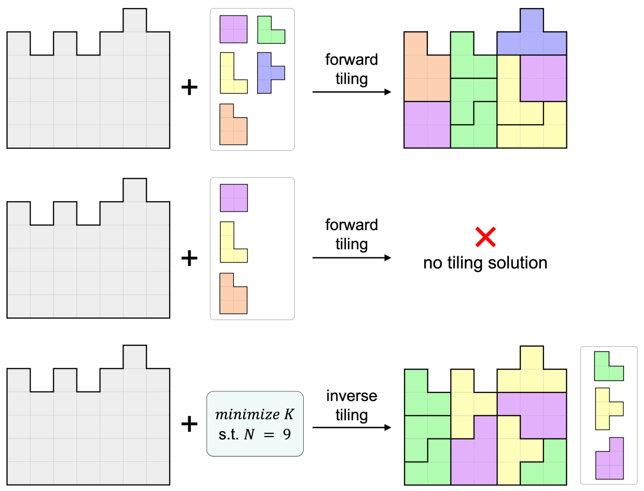


Fig. 2. (Top) Forward tiling assumes a given set of prototiles for tiling the domain. (Middle) When the prototile set is not sufficiently large, forward tiling may not produce a tiling solution. (Bottom) Our inverse tiling approach computes a small set of prototiles (three vs. five in forward tiling) that can tile the input domain with $N = 9$ prototile instances.

studied [Grünbaum and Shephard 2016] and employed in various graphics applications [Kaplan 2009]. In this work, we study K-hedral tiling of 2D finite domains, which often appears in recreation and architecture applications. In practice, a small set of prototiles (i.e., a small K), not only makes the tiling aesthetically pleasant with more congruent appearance, but also reduces the fabrication cost (e.g., by molding) and simplifies the tile assembly process.

To produce a K-hedral tiling for a 2D finite domain, existing approaches typically assume that the *prototile set* is *given*. That means K and the shapes of the K prototiles are prescribed. Hence, the tiling process is to take instances of the prototiles to seamlessly cover the given domain [Garvie and Burkardt 2020; Xu et al. 2020]. This forward tiling process is illustrated in Figure 2 (top). On the one hand, when the prototile set contains a large variety of shapes (large K) , we have more choices, so a tiling solution can often be found. However, the resulting tiling may involve a large variety of prototiles, since K is *not* minimized, so the fabrication cost would increase. On the other hand, when the number of prototiles is not that large (small K), it is uncertain whether the prototile set can tile the domain; see Figure 2 (middle) for an example. To know whether there is any tiling solution requires a time-consuming trial-and-error process over different subsets of a large prototile set.

In this work, we take a new perspective to look at the production of *K-hedral tiling* and formulate a new tiling problem called *inverse tiling of 2D finite domains*, for which

(i) the prototile set is not given, and is computed (or inversely modeled) to fit the input domain; and
(ii) the number (K) of prototiles needed to tile the domain should be minimized.

Figure 2 (bottom) shows an example, in which a small set of three prototiles is computed for tiling the same domain as the forward tiling example on top, which uses five prototiles. Importantly, as Figure 2 (middle) shows, arbitrarily picking three prototiles may not form a prototile set that can tile the domain. Fundamentally, forward tiling is a well-known NP-complete problem [Moore and Robson 2001]. As for inverse tiling, we have proven that it is an *NP-hard* problem, meaning that it is at least as hard as the forward tiling problem; see the mathematical derivation in Supp. 1.

Obviously, a trivial solution to produce inverse tilings with minimized K would be using a single-square prototile (i.e., monomino) to tile the entire domain, say for the square-grid domain in Figure 2. Then, K is only one. Taking one step forward, we may use monominoes and dominoes to tile the domain, then K is only two. However, the resulting tilings would be less appealing with only simple prototile shapes and contain a huge number of tiles. Hence, as Figure 2 (bottom) illustrates, we take N, the number of tiles (i.e., prototile instances), as a control parameter to facilitate the construction of K-hedral tilings with more interesting prototile shapes. Moreover, we impose a constraint on the prototile size to avoid generating prototiles that are too small or too large.

To make inverse tiling tractable, we assume that the input 2D domain is represented by a grid, in which the grid cells have few (typically one or two) unique shapes; see Figure 2 (left) for an example. Having that, we design a computational approach to progressively build the prototile set while tiling the input domain. To minimize the number (K) of prototiles, the approach starts by randomly distributing N seeds (i.e., instances of a single-cell prototile with $K = 1$) over the input domain. Then, it iteratively enlarges the instances of each prototile in a congruent way, to avoid unnecessary increase in the number of prototiles, until the domain is fully covered by the prototile instances. Once a tiling of the domain is obtained, it further tries to reduce the number of prototiles by locally re-tiling the input domain to eliminate prototiles with few instances.

Specifically, we make the following contributions.

- We formulate a new tiling problem called the *inverse tiling* for producing K-hedral tilings in 2D finite domains, aiming to automatically find a minimized set of prototiles, typically with desired size and/or shape, for tiling the input domain.
- We propose a computational approach for solving the inverse tiling problem, where our approach outputs the number of prototiles, shape of each prototile, and a tiling of the domain using instances of the constructed prototiles.

We show that our inverse tiling approach is able to tile 2D finite domains of a rich variety of shapes and grid structures; see Figure 1. We evaluate the scalability of our approach with respect to the input domain size as well as the number (N) of tiles, and evaluate also the performance of our approach in minimizing the number (K) of prototiles. Further, we compare our inverse tiling approach with existing forward tiling approaches, showing that our approach is able to find tiling solutions with a smaller number of prototiles. Code and data of this paper are available at https://github.com/Linsanity81/InverseTiling.

2 Related Work

Tiling of 2D finite domains. In computer graphics, tiling of the infinite Euclidean plane has been studied in various contexts such as texture synthesis [Cohen et al. 2003], sampling [Ahmed 2019;

Kopf et al. 2006; Ostromoukhov 2007; Ostromoukhov et al. 2004], and decorative pattern generation [Meekes and Vaxman 2021]. In particular, Smith et al. [2024] recently discovered the first true aperiodic monotile called "the hat" for tiling the plane. Instead of tiling the plane, some works study tiling of 2D finite domains using a small set of prescribed tile shapes. Peng et al. [2018] studied tiling of 2D finite domains using squares and regular triangles, and further lifted the 2D tiling onto a 3D surface for designing interesting triangle-quad hybrid meshes. Later, Peng et al. [2019] studied tiling with three specially-chosen rhombic tiles, i.e., 30°-, 60°-, and 90°-rhombus, and generated a variety of tiling results with intricate 2D and 3D checkerboard patterns. Xu et al. [2020] designed a neural optimization to approximate a given 2D finite domain using one or more types of tile shapes, attempting to maximally fill the domain's interior without overlaps or holes.

All the above works take a forward approach to find a feasible tiling of a 2D finite domain, whose boundary is represented by a closed polyline. Due to the simplicity of the prescribed prototile set, such approach can only tile domains whose boundary satisfies certain geometric constraints [Peng et al. 2019, 2018] or generate tilings that maximally cover the input domain [Xu et al. 2020].

Polyform tiling. A polyform is a planar figure constructed by joining together identical basic polygons edge to edge, e.g., squares, regular triangles, and regular hexagons, where the associated polyforms are called the polyomino, polyiamond, and polyhex, respectively [Myers 2019]. Polyform tiling employs instances of one or more (small) polyforms to cover a given (large) polyform, which can be finite or infinite. Among the various polyform tilings, polyomino tiling [Golomb 1994] is a well-known one. Golomb conducted pioneer research on polyomino tiling of an infinite domain (e.g., plane, half plane, and strip) [Golomb 1966, 1970] and a finite domain with a simple boundary like rectangles [Golomb 1989, 1996].

Recently, researchers are interested in the problem of tiling a finite domain (with a complex boundary) using a prescribed set of polyominoes. One motivation is that polyomino tilings can be employed to produce recreational puzzles [Kita 2023; Kita and Miyata 2021; Lo et al. 2009]. A number of computational approaches have been proposed for polyomino tiling. Knuth [2000] formulated polyomino tiling as an exact cover problem and proposed a recursive, nondeterministic, backtracking algorithm called Algorithm X to find all solutions to the problem. Garvie and Burkardt [2020] formulated polyomino tiling as a binary linear programming problem, requiring a known number of instances of each prescribed polyomino. Later, they [Garvie and Burkardt 2022] combined the linear programming approach with a divide-and-conquer strategy to improve the computational efficiency. Misiak [2022] formulated polyomino tiling as a boolean satisfiability problem, where the domain exact-cover condition is translated into a set of boolean formulae.

All the above works take a forward tiling approach, assuming a prescribed set of polyominoes as the given prototile set. However, prescribing a prototile set that can tile a 2D finite domain may not be a tractable task, especially when using high-order polyominoes. This is because the number of possible polyomino shapes increases *exponentially* with the number of squares that the polyomino has [Barequet and Shalah 2022]. For example, the number of free hexominoes (6 squares) is 35, whereas the number of free dodecominoes (12 squares) is 63,600 [Redelmeier 1981]. Compared with forward tiling, our inverse tiling approach has two major differences. First, it automatically discovers a set of tile shapes (i.e., prototiles) to fit the input 2D finite domain, avoiding tedious trial-and-error process of prescribing the prototile set. Second, it minimizes the number of prototiles in the tiling, reducing the cost of realizing the tiling result. These advantages are demonstrated with a quantitative comparison shown in Figure 13.

Escherization. Escher tiling repeats one or few irregular but recognizable figures, such as animals and human faces, to tile the plane. Escherization was initially termed by Kaplan and Salesin [2000] as a problem of finding the shape most similar to a given goal figure, such that the shape can tile the plane. Later, the authors [2004] extended their method to produce dual-shape Escher tilings using dihedral tiling patterns. Recently, Liu et al. studied the problem of modeling dual-shape Escher tilings from user-defined shapes while ensuring these tilings are fabricable [Liu et al. 2020] and deployable [Liu et al. 2024]. Nagata and Imahori [2021] computed satisfactory Escher tile shapes with natural deformations for fairly complex goal figures using as-rigid-as-possible shape modeling. Please refer to [Nagata and Imahori 2024] for a review of research works on Escher tiling.

Escherization is related to inverse tiling, since the Escher tile shape(s) are computed instead of being prescribed. However, there are two major differences. First, Escherization aims to tile the infinite Euclidean plane, whereas inverse tiling aims to tile 2D finite domains. Second, Escherization requires a known number of prototiles (usually one or two) and the goal figure of each prototile is given, while inverse tiling computes a minimized number of prototiles from scratch for tiling an input finite domain.

3 Formulation of Inverse Tiling Problem

Given a 2D finite domain D represented by a grid, our goal is to compute a K-hedral tiling of the domain with minimized K. To start, we first provide a formal definition of the *inverse tiling* problem by elaborating on its input, output, and requirements.

- *Input.*
  - (i) *2D finite domain D.* Domain D is represented by a grid with a closed boundary. Holes and disjoint regions are allowed in the domain.
  - (ii) *Total number of tiles.* The total number of tiles is denoted as N. In practice, N is closely related to the cost of fabricating the tiling and the time to assemble the tiling.
- *Output.*
  - (i) *Prototile set.* The set of prototiles is denoted as $\{t_k\}_{k=1}^{K}$, where t_k is a prototile and K is the total number of prototiles.
  - (ii) *Number of instances of each prototile.* The number of instances of each prototile t_k is denoted as n_k, which should satisfy $\sum_{k=1}^{K} n_k = N$.
  - (iii) *Transformation of each prototile instance.* The jth instance of prototile t_k is denoted as tile $t_{k,j}$, $j \in \{1, 2, ..., n_k\}$. The placement location, orientation, and reflection of each tile $t_{k,j}$ in domain D is represented by a transformation denoted as $T_{k,j}$.

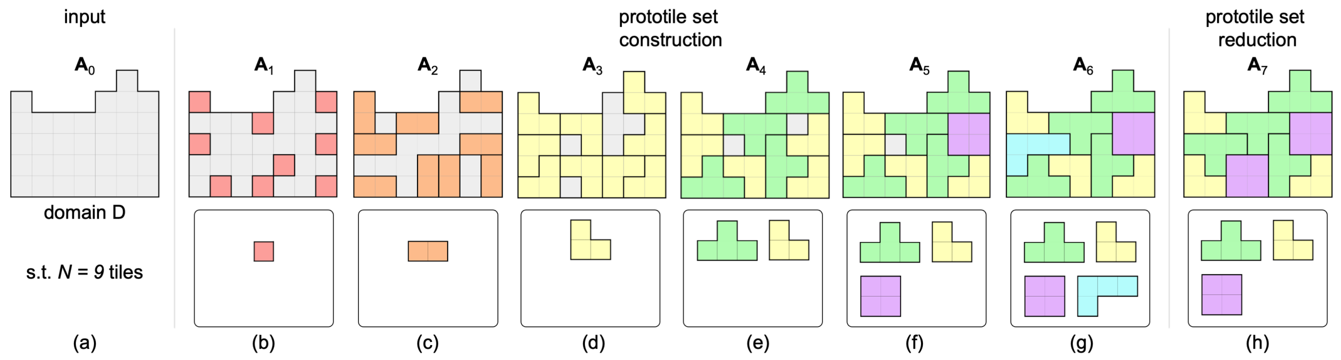


Fig. 3. Overview: our inverse tiling approach consists of two stages: (b-g) prototile set construction and (h) prototile set reduction. Given (a) input domain D, we (b) initialize the prototile set with a single element (i.e., a monomino) and randomly distribute the prototile's $N = 9$ instances over the domain. We (c-g) iteratively enlarge the tiles to cover a larger portion of the domain while maintaining congruence of the tile shapes whenever possible to minimize the number of prototiles, until the domain is fully covered by the tiles. (h) We further reduce the number of prototiles by eliminating one prototile (in cyan) with a single instance via locally re-tiling. Note that each prototile and its instances in a tiling state $\mathbf{A}_i$ are rendered using the same color to show their correspondence.

- *Requirements.*

(i) *Tileable domain.* The computed tiles $\{t_{k,j}\}$ should exactly cover domain D without gaps or overlaps.

(ii) *Minimizing the number of prototiles.* The number of prototiles, K, for tiling domain D should be minimized.

(iii) *Prototile size.* To avoid prototiles that are too small or too large, we specify a range $[C_{\min}, C_{\max}]$ on the number of grid cells occupied by each prototile produced by inverse tiling; e.g., range $[4, 6]$ for a square grid means that we take only tetrominoes, pentominoes, and hexominoes as the prototiles.

(iv) *Prototile shape.* The prototile shape has to be simply connected, according to the definition of tiling [Grünbaum and Shephard 2016]. Besides, we may optionally specify constraints on the prototile shape, e.g., bounding box size $W_{\text{bbox}} \times H_{\text{bbox}}$ where $W_{\text{bbox}} \geq H_{\text{bbox}}$ and convexity. Note that the convexity of a prototile is measured using the ratio of the perimeter of the prototile's convex hull to the perimeter of the prototile itself [Sonka et al. 2011]. Note that checking whether a prototile shape satisfies each of these constraints can be done by a polynomial-time decision procedure.

4 Key Concepts in Inverse Tiling

Overview of our approach. Our inverse tiling approach consists of two stages; see Figure 3 for a running example.

(1) *Prototile set construction.* At the beginning, the prototile set is initialized with a single element (i.e., $K = 1$), which is simply a monomino. We randomly distribute N instances of the monomino as seeds over domain D; see Figure 3(b). Next, we progressively enlarge all the tiles congruently to cover a larger portion of domain D while maintaining $K = 1$; see Figure 3(c). Sometimes, enlarging all the instances of a prototile in a congruent way is not possible; e.g., we cannot enlarge the bottom-right tile in Figure 3(d), since all its adjacent grid cells are occupied. In this case, we choose to enlarge a subset of the prototile's instances in a congruent way, resulting in an increase in the number (K) of prototiles by one; see Figure 3(d&e). We iteratively perform this *tile enlargement operation* until all the grid cells in domain D are covered by the computed tiles; see Figure 3(g).

(2) *Prototile set reduction.* In practice, the constructed prototile set may contain prototiles with few instances (e.g., $n_k = 1$ or 2); see, e.g., the cyan prototile with $n_k = 1$ in Figure 3(g). Hence, we propose to try to eliminate such prototiles via a local re-tiling, aiming to reduce the number of prototiles; see Figure 3(h). In the end, our approach outputs the prototile set $\{t_k\}$, number of instances of each prototile $\{n_k\}$, and transformation of each prototile instance $\{T_{k,j}\}$.

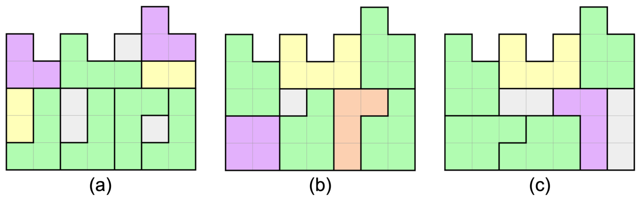


Fig. 4. Three example abortive tiling states. (a) The two yellow tiles cannot be enlarged. Their number of grid cells is less than the minimum allowed number of grid cells $C_{\min} = 3$. (b) The uncovered grid cell (in gray) cannot be assigned to any tile without violating the maximum allowed number of grid cells $C_{\max} = 5$. (c) We cannot assign the uncovered grid cells without violating the bounding box size constraint $W_{\text{bbox}} = 3$ and $H_{\text{bbox}} = 2$.

Next, we introduce various concepts in our approach. First, we introduce the *abortive tiling state* to motivate the *search tree* data structure for prototile set construction. Then, we describe the *tile enlargement operation* for prototile set construction, followed by two requirements on this key operation.

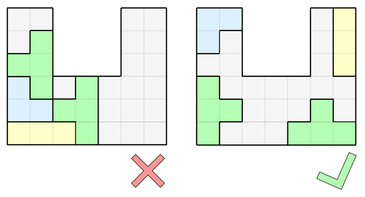
Abortive tiling state. The iterative prototile set construction process can be seen as a sequence of tiling states $\{\mathbf{A}_i\}$, where a tiling state $\mathbf{A}_i$ represents a (partial) covering of domain D by instances of the current set of prototiles. In particular, domain D is the first tiling state $\mathbf{A}_0$, in which all grid cells are not covered. Each iteration of enlarging the tiles transits a tiling state $\mathbf{A}_i$ to its next state $\mathbf{A}_{i+1}$. A tiling state $\mathbf{A}_i$ is called an *abortive tiling state* if (i) there exists at least one uncovered grid cell in domain D; and (ii) enlarging any tile will violate the requirements on the prototile size and/or shape. Figure 4 shows three typical cases of abortive tiling states.

Search tree for prototile set construction. Due to the existence of abortive tiling states, given an arbitrary tiling state $\mathbf{A}_i$, we cannot guarantee that it can always transit to the next valid (i.e., non-abortive) tiling state $\mathbf{A}_{i+1}$. Therefore, we propose to use a search tree with backtracking to iterate through valid tiling states while avoiding abortive tiling states for prototile set construction.

Our key idea is to build and maintain a tree data structure, where the root node is the first tiling state $\mathbf{A}_0$ and each node in level $i > 0$ represents a candidate of a tiling state $\mathbf{A}_i$. For each node in level i, we generate a set of its childs denoted as $\{\mathbf{C}_j\}$, where $\mathbf{C}_j$ is a candidate of the next tiling state $\mathbf{A}_{i+1}$. Our approach ranks these candidate tiling states $\{\mathbf{C}_j\}$ based on two criteria: (i) fewer number of prototiles; and (ii) fewer number of uncovered grid cells, where we prioritize the first criterion over the second one. In case no valid child can be generated from a node in the tree, we backtrack the tree to try other nodes without restarting the whole prototile set construction process. We denote the size of $\{\mathbf{C}_j\}$ as m. A large m needs more time to generate $\{\mathbf{C}_j\}$ but offers more choices for ranking and backtracking. We set $m = 15$ by default in our experiments.

Tile enlargement for prototile set construction. The key operation in our approach is to enlarge the tiles in a tiling state $\mathbf{A}_i$ to cover more grid cells in domain D in the next tiling state $\mathbf{A}_{i+1}$. Importantly, the tile enlargement operation should satisfy requirements (i)-(iv) of inverse tiling presented in Section 3. In particular, the last two requirements on prototile size and shape are naturally satisfied, thanks to our *iterative* prototile set construction strategy (see Section 5.1). For example, we can ensure that the shape of each enlarged tile remains simply connected by including only uncovered grid cell(s) adjacent to the tile; see Figure 3. However, the first two requirements (i.e., tileable domain and minimizing K) are harder to satisfy, since they are global requirements. Hence, we propose two requirements on the (local) tile enlargement operation, helping to meet the two (global) requirements of inverse tiling, respectively:

(i) Enlargeable tiles requirement. To meet the requirement of tileable domain, our idea is to leave more space around the tiles, such that we can make room to enlarge them in subsequent tiling states. This idea is called the *enlargeable tiles requirement.* By this requirement, we can reduce the chance of generating abortive tiling states, thus reducing the number of backtracking needed. To this end, we propose to distribute initial seed tiles more evenly (see Section 5.1). Also, we propose to enlarge tiles in a way that the enlarged tiles still have some adjacent grid cells uncovered. The inset shows a counter example, where only half of the tiles are enlargeable, and a desired example, where all the tiles are enlargeable.



(ii) Congruent tiles requirement. To help to minimize K in the final tiling result, we propose the following greedy strategies performed at each tiling state.

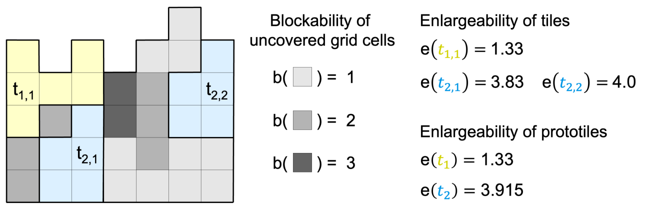


Fig. 5. We show the blockability (b) value (for $I = 2$) of each uncovered grid cell, enlargeability (e) value of each tile, and enlargeability (e) value of each prototile, for a tiling state with 3 tiles and 2 prototiles.

(1) *Constructing congruent tiles.* For each prototile t_k in state $\mathbf{A}_i$, we try to enlarge all its instances congruently whenever possible, since doing so keeps the number of prototiles unchanged in state $\mathbf{A}_{i+1}$; see Figure 3(c&d). Otherwise, we enlarge a subset of prototile t_k's instances congruently; see Figure 3(d&e).
(2) *Matching existing prototiles.* For a small prototile t_k in state $\mathbf{A}_i$, we try to enlarge it, so that the enlarged prototile t_k has the same shape as an existing prototile t_j in state $\mathbf{A}_i$, aiming to reduce the number of prototiles by one in state $\mathbf{A}_{i+1}$. If this cannot be achieved, we try to enlarge the small prototile t_k such that its whole shape matches part of an existing large prototile t_j. By doing so, prototile t_k may have the same shape as prototile t_j after a few more tile enlargement operations on t_k.

5 Inverse Tiling Approach

We present our two-stage inverse tiling approach by composing the concepts in Section 4, where the prototile set construction is the main stage and the prototile set reduction is an optional stage. The main stage enables efficiently solving the inverse tiling problem by significantly reducing the search space; see Supp. 2 for an analysis.

5.1 Stage 1: Prototile Set Construction

5.1.1 Initializing Tiles. To initialize N tiles in the input domain D, we pick N grid cells as seeds in the domain using Poisson disk sampling to meet the enlargeable tiles requirement, where each initial tile occupies a single seed grid cell; see Figure 3(a&b). Our implementation uses a dart-throwing process [Lagae and Dutré 2008] to iteratively generate seeds. At each iteration, we randomly select an uncovered grid cell as a seed candidate and check if it satisfies the requirement of the minimum distance between existing seeds, in which the minimum distance is set to $\sqrt{M/N}$ by default and M is the total number of grid cells in the domain. If the seed candidate does not satisfy the requirement, we discard it and generate another candidate until the distance requirement is satisfied.

An alternative approach to generating seeds is to regularly or semi-regularly distribute seeds over the input domain D. Taking the square-grid domain as an example, the number of seeds is controlled by two parameters, i.e., the horizontal and vertical distances between adjacent seeds. We found that this alternative approach works well in practice, especially when the input domain has a regular shape. However, this approach may not be able to generate exactly N seeds for a given input domain.

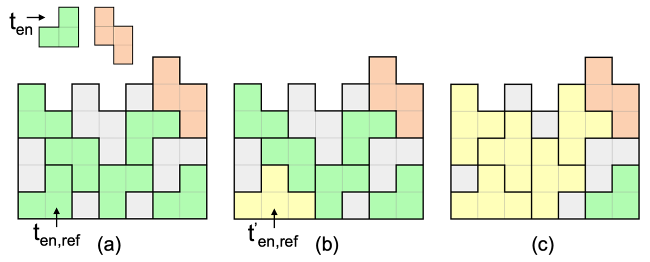


Fig. 6. Enlarging part of the tiles congruently. We first (a) select prototile t_{en} (in green color) from existing prototiles $\{t_k\}$, and then select prototile instance $t_{en,ref}$ as the reference tile. Next, we (b) enlarge the reference tile $t_{en,ref}$ to form the target tile $t'_{en,ref}$ (in yellow color) by including an adjacent uncovered grid cell. We further (c) try to enlarge each of the remaining instances of the prototile t_{en} to make it have the same shape as the target tile $t'_{en,ref}$.

5.1.2 Enlarging Tiles. For a tiling state $\mathbf{A}_i$, we enlarge all or part of the tiles in a congruent way to transit to the next tiling state $\mathbf{A}_{i+1}$. To meet the enlargeable tiles requirement, our idea is that the enlargement of a tile should minimize its potential blocking impact to its neighboring tiles. Hence, we propose the blockability measure for uncovered grid cells and the enlargeability measure for tiles and prototiles, such that we later can use these measures to guide the tile enlargement operation. All the measures are computed once for each tiling state $\mathbf{A}_i$:

(i) Blockability of an uncovered grid cell. We compute the blockability value denoted as $b(x)$ for an uncovered grid cell x as

$$b(x) = \text{number of tiles intersecting with } I\text{-ring grid cell neighborhood of } x, \quad (1)$$

where I is set to 3 in our implementation, but can be adjusted according to the size of the input domain. Note that we consider edge-sharing but not corner-sharing when locating grid cell neighbors using a breadth-first search. Figure 5 visualizes the blockability of uncovered squares for $I = 2$. In our approach, we prioritize choosing uncovered grid cells with low blockability for enlarging tiles.

(ii) Enlargeability of a tile. The enlargeability of a tile $t_{k,j}$ in a tiling state is computed as

$$e(t_{k,j}) = \sum_{x_u \in \text{adjacent\_uncovered}(t_{k,j})} \frac{1}{b(x_u)}, \quad (2)$$

where x_u is an uncovered grid cell adjacent to tile $t_{k,j}$ and $b(x_u)$ is the blockability value of x_u. Since tiles with low enlargeability are likely to be blocked by other tiles, we prioritize to choose them for performing the enlargement operation.

(iii) Enlargeability of a prototile. For each prototile t_k, its enlargeability is computed as

$$e(t_k) = \frac{1}{n_k} \sum_j e(t_{k,j}) \quad (3)$$

where tile $t_{k,j}$ is the jth instance of prototile t_k and n_k is the number of instances of prototile t_k; see Figure 5 for examples.

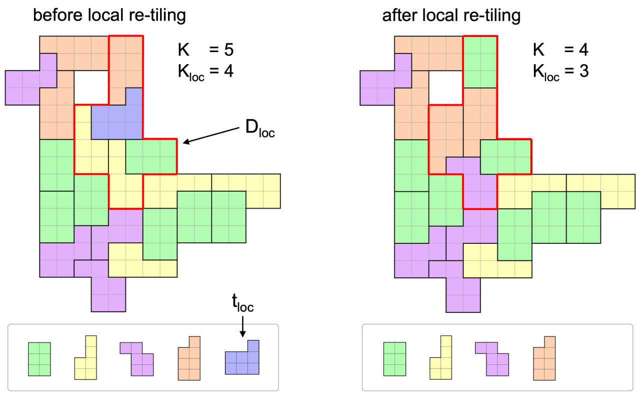


Fig. 7. Re-tiling a local region to reduce the number of prototiles. (Left) We first select prototile t_{loc} (in blue) that has a single instance, and then identify a region D_{loc} around its instance (with a red boundary). (Right) Next, we re-tile region D_{loc} using a modified version of the prototile set construction method, guided by the prototiles of remaining domain $D_{rem} = D - D_{loc}$.

Generating candidates of the next tiling state. Given a tiling state $\mathbf{A}_i$, to proceed to the next tiling state $\mathbf{A}_{i+1}$, we build the search tree by generating a set of candidates of $\mathbf{A}_{i+1}$ and choose among the candidates based on the two criteria presented in Section 4. The candidate generation procedure has the following four steps; please refer to Supp. 3 for details.

(i) *Selecting a prototile t_{en}.* We select a prototile, say t_{en}, from the current set of prototiles $\{t_k\}$ for enlarging its tiles (prototile instances) based on the following three criteria (see Figure 6(a)): small prototile size, a large number of prototile instances, and low enlargeability.

(ii) *Selecting a prototile instance $t_{en,\,ref}$.* After choosing prototile t_{en}, we further pick one of its instances, say $t_{en,\,ref}$, to form a reference tile; see Figure 6(a). We prioritize selecting $t_{en,\,ref}$ as a prototile instance with low enlargeability.

(iii) *Determining the target tile $t'_{en,\,ref}$.* After choosing prototile instance $t_{en,\,ref}$, we determine the shape of target tile $t'_{en,\,ref}$ to make it either match an existing prototile or part of an existing prototile, guided by the congruent tiles requirement. If this cannot be achieved, we determine the shape of target tile $t'_{en,\,ref}$ by assigning an adjacent uncovered grid cell with low blockability to tile $t_{en,\,ref}$; see Figure 6(b).

(iv) *Enlarging the prototile instances $\{t_{en,\,j}\}$.* Once the shape of target tile $t'_{en,ref}$ is determined, we try to enlarge each of the remaining prototile instances in $\{t_{en,\,j}\}$ to make it have exactly the same shape as $t'_{en,\,ref}$, to meet the congruent tiles requirement; see Figure 6(c).

If the attempt to generate a set of candidates of the next tiling state leads to an empty set, our algorithm will backtrack to an earlier tiling state in the search tree.

5.2 Stage 2: Prototile Set Reduction

Tiling results generated by the prototile set construction stage in Section 5.1 may contain prototiles with few instances. In the prototile set reduction stage, we locally re-tile the result to attempt to

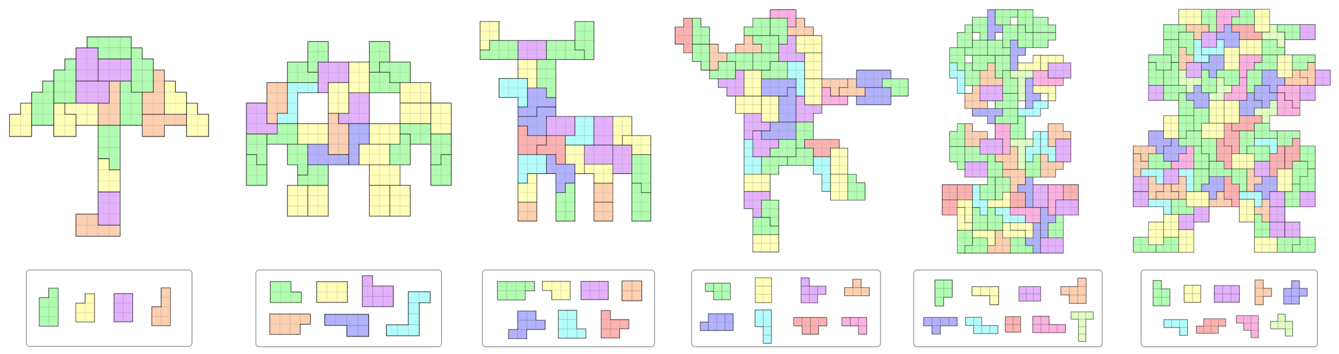


Fig. 8. Tiling results produced by our approach on input square-grid domains of various shapes, from left to right: Umbrella, Robot, Deer, Mega Man, Flower, and Mario.

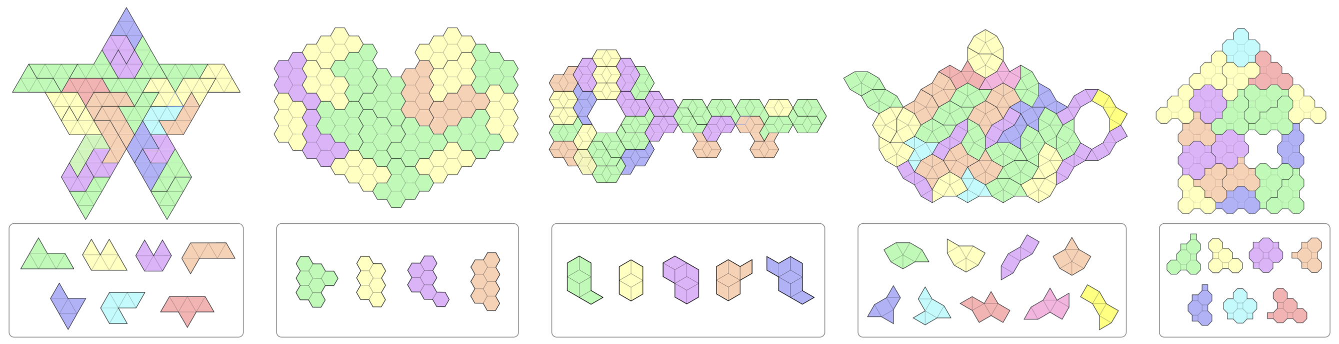


Fig. 9. Tiling results produced by our approach on input domains represented by different kinds of grids, from left to right: a triangle, hexagon, rhombus, square-triangle, and octagon-square grid.

eliminate such prototiles, aiming to reduce the size of the prototile set. We design the local re-tiling operation as follows:

(i) *Select a prototile.* We select a prototile, say t_{loc}, with $n_{\text{loc}} \leq 3$, where n_{loc} is the number of instances of the prototile; see Figure 7 (left) for an example.

(ii) *Identify the associated region.* We split domain D into two regions D_{loc} and D_{rem}. D_{loc} includes all instances of prototile t_{loc} and the one-ring tile neighbors of each instance, whereas $D_{\text{rem}} = D - D_{\text{loc}}$, which denotes the remaining region in domain D. Note that region D_{loc} may be disconnected, since prototile t_{loc} may have more than one instance (i.e., $n_{\text{loc}} > 1$).

(iii) *Re-tile the region.* We re-tile region D_{loc} using a modified version of the prototile set construction method in Section 5.1, in which we prioritize enlarging tiles $\{t^{\text{loc}}_{k,j}\}$ in region D_{loc} to match the prototiles in the remaining domain D_{rem}.

(iv) *Confirm the re-tiling.* We accept the re-tiling on the original tiling result, if the number of prototiles of the entire domain D is reduced. Otherwise, we undo the re-tiling.

We iterate the local re-tiling operation until the number of prototiles of domain D cannot be reduced. Figure 7 shows an example tiling result before and after the local re-tiling operation, where the number of prototiles is reduced by one.

6 Results

We implemented our inverse tiling approach in C++ and ran it on a desktop computer with 3.6GHz 8-Core Intel processor and 16GB RAM. To generate a tiling result, we need to specify the input domain D (with M grid cells) and the number of tiles N. Also, the prototile size constraint $[C_{\min}, C_{\max}]$ is set as $C_{\min} = \lfloor \alpha_{\min} M/N \rfloor$ and $C_{\max} = \lceil \alpha_{\max} M/N \rceil$, where $\alpha_{\min}$ and $\alpha_{\max}$ are constants set as 0.8 and 1.2, respectively, by default. For each result, we produce the computed prototiles $\{t_k\}$ and a tiling of the input domain by $\{t_k\}$. To present the results, each prototile t_k is colored based on its number of instances n_k: green for the largest n_k, yellow for the second-largest, purple for the third, and so on.

Tiling results. We tested our inverse tiling approach on input domains of various shapes represented by a square grid in Figure 8 and on input domains represented by different kinds of grids (with one or two unique cells) in Figure 1 and 9. For each of these results, our approach computes a small number of prototiles (e.g., $K \leq 9$) to tile the input domain; see Table 1 for the statistics and timings of these results. Our inverse tiling approach allows to tile an input domain with a disconnected shape, and to control the shape (e.g., convexity and bounding box) of the generated prototiles; please refer to Supp. 4 for more tiling results. The accompanying video visualizes the inverse tiling process of some results.

Table 1. Statistics and timings. We report the input domain, grid type, number of grid cells in the domain (M), number of tiles (N), prototile size constraint ($[C_{\min}, C_{\max}]$), number of prototiles before local re-tiling (K_{constr}), number of prototiles in the tiling result (K), average number of instances of each prototile (N/K), timings for running stage 1 in Section 5.1 (T_{constr}), stage 2 in Section 5.2 (T_{reduce}), and the whole method in Section 5 (T_{total}).

Fig	Input Domain	Grid	M	N	$C_{\min}$	$C_{\max}$	K_{constr}	K	N/K	T_{constr} (mins)	T_{reduce} (mins)	T_{total} (mins)
1	Bunny	Squ	484	100	4	6	9	6	16.7	32.6	12.7	45.3
		Hex	509	100	5	6	12	5	20.0	43.0	9.7	52.7
		Squ-Tri	756	125	5	7	20	8	15.6	49.7	21.6	71.2
8	Umbrella	Squ	120	20	5	7	6	4	5.0	0.6	0.3	0.9
	Robot		202	35	5	7	9	6	5.8	13.3	1.8	15.1
	Deer		180	30	4	7	9	7	4.3	7.2	0.9	8.1
	Mega Man		274	50	4	8	16	8	6.3	35.8	13.9	49.7
	Flower		408	80	4	6	14	9	8.9	55.0	23.4	78.4
	Mario		604	120	4	6	18	9	13.3	119.6	16.1	135.7
9	Kite	Tri	172	30	5	6	9	7	4.3	10.6	2.1	12.6
	Heart	Hex	130	20	6	7	6	4	5.0	8.5	1.0	9.5
	Key	Rho	190	35	5	7	8	5	7.0	12.6	5.3	17.8
	Teapot	Squ-Tri	242	42	5	6	12	9	4.7	37.4	8.2	45.6
	House	Oct-Squ	134	20	6	8	8	7	2.9	28.2	6.8	34.9

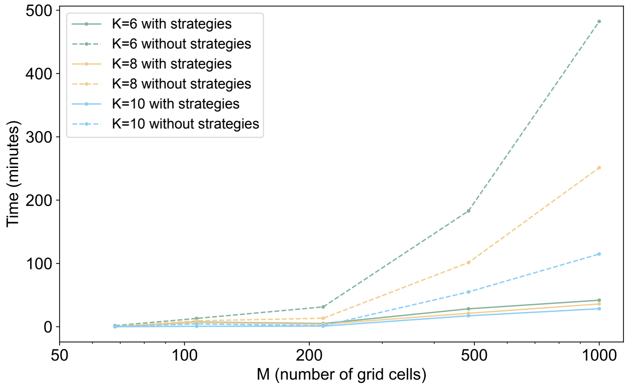


Fig. 10. Experiment to evaluate the scalability of our inverse tiling approach for tiling Bunny in different resolutions. We compare our approach (in solid curves) with a baseline approach (in dashed curves) that uses our computational framework without our proposed strategies.

Evaluating scalability of our approach. We conducted an experiment to evaluate the scalability of our inverse tiling approach by comparing it with two baseline approaches. In this experiment, the task is to find K-hedral tilings of Bunny in different resolutions (i.e., different numbers (M) of grid cells). The first baseline is a randomized search of all possible partitions of the input domain into N pieces (i.e., tiles) to find the K-hedral tiling result. Due to the huge search space (i.e., $O(N^M)$), this approach can only find tiling results for small M and N. The second baseline is a randomized search within our computational framework, where the proposed strategies (i.e., blockability and enlargeability in Section 5.1) are disabled. Figure 10 shows that our approach is more scalable with respect to M (i.e., the input domain size), thanks to the framework in Section 4 and the strategies in Section 5.1. Please refer to Supp. 5 for associated visual results and statistics of this experiment.

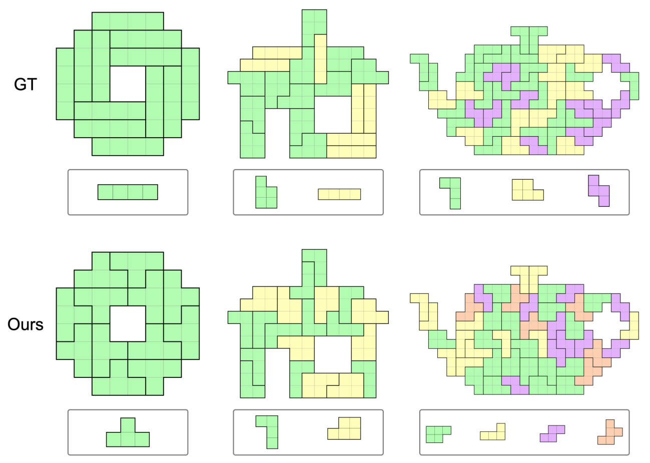


Fig. 11. Experiment to evaluate the performance of our inverse tiling approach in terms of minimizing the number (K) of prototiles by comparing our tiling results (bottom) with the ground-truth results with a minimal K (top), from left to right: Coin, House, and Teapot. From left to right, the time taken to generate the ground-truth results is 0.02, 0.78, and 2.74 minutes while the time taken to generate our corresponding results is 0.11, 2.31, and 15.35 minutes, respectively.

This experiment fixes the input domain shape, i.e., Bunny. In practice, we observed that the scalability of our approach with respect to the number (N) of tiles is closely related to the complexity of the input domain shape. In detail, an input domain with more intricate shape (e.g., complex boundary, many holes in the domain) typically makes our approach less scalable with respect to N.

Evaluating K minimization of our approach. We conducted an experiment to evaluate the performance of our inverse tiling approach in terms of minimizing the number (K) of prototiles by comparing our results with ground truths. In this experiment, the task is to tile each of the three input domains (i.e., Coin, House, and Teapot) using tetrominoes (4 squares) and/or pentominoes (5 squares). Figure 11 shows the ground truth and our result for each domain. We can see that our inverse tiling approach is able to find the minimal K for Coin ($K = 1$) and House ($K = 2$), and a minimized K that is close to the ground truth for Teapot ($K = 4$ vs $K = 3$). Please refer to Supp. 6 for details of this experiment, including how the ground-truth result is computed.

Comparison with forward tiling approaches. We compare our inverse tiling approach with three forward tiling approaches [Mišiak 2022] (i.e., backtracking (BT), integer linear programming (ILP), and boolean satisfiability (SAT)) in terms of minimizing the number of prototiles. There are three tasks in this experiment, i.e., tiling the Bunny with three different prototile size constraints $[C_{\min}, C_{\max}]$: $[2, 5]$, $[5, 8]$, and $[8, 11]$. For a fair comparison, the sets of allowable tile shapes are formed by enumerating polyominoes that satisfy the prototile size constraint for each task, resulting in sets of 20, 517, and 22165 allowable tile shapes, respectively. For each task, we ran each of the three forward tiling approaches as well as our inverse tiling approach for at most 24 hours. Figure 13 reports the computation

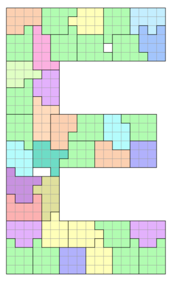
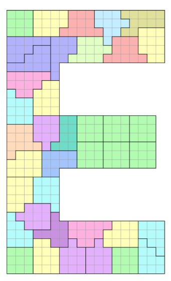
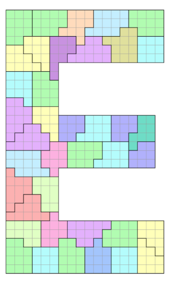
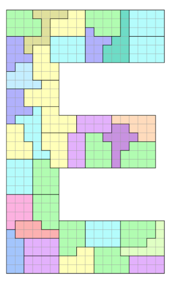
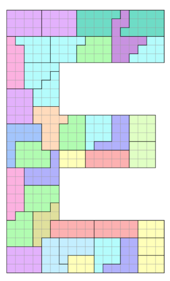
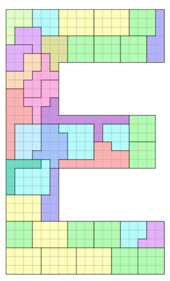
$[C_{min}, C_{max}]$	[10, 10]	[9, 11]	[8, 12]	[7, 13]	[6, 14]	[5, 15]
						
Time (min)	/	74.83	11.71	2.15	0.32	0.21

Fig. 12. Tiling results with $N = 39$ tiles and $K = 15$ prototiles produced by our inverse tiling approach on E, an input domain with $M = 390$ grid cells, under different prototile size constraints (see the top row in the table). Note that our approach was not able to generate a complete tiling result with $K = 15$ prototiles within 24 hours when the prototile size constraint is too restrictive (i.e., $C_{min} = C_{max} = 10$); see the leftmost for a failure case (i.e., an abortive tiling state) with 3 uncovered grid cells.

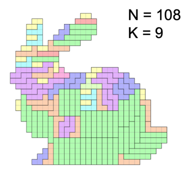
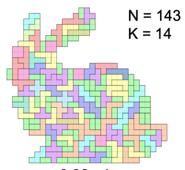
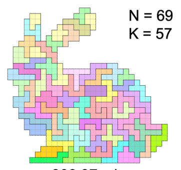
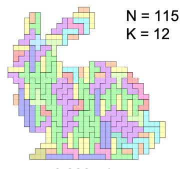
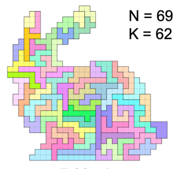
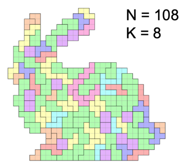
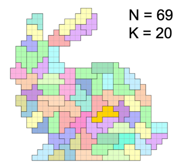
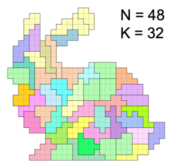
	Task #1	Task #2	Task #3
	$[C_{min}, C_{max}] = [2, 5]$	$[C_{min}, C_{max}] = [5, 8]$	$[C_{min}, C_{max}] = [8, 11]$
BT	 N = 108 K = 9 0.26 min		
ILP	 N = 143 K = 14 0.36 min	 N = 69 K = 57 909.97 min	
SAT	 N = 115 K = 12 0.093 min	 N = 69 K = 62 7.66 min	
Ours	 N = 108 K = 8 8.72 min	 N = 69 K = 20 47.61 min	 N = 48 K = 32 105.67 min

Fig. 13. Comparing our inverse tiling approach (bottom) with three forward tiling approaches for tiling Bunny under three different prototile size constraints. An empty table cell means the approach cannot find a tiling solution for that task within 24 hours.

time, number of tiles N, and number of prototiles K, for each tiling solution. Compared with the three forward tiling approaches, our inverse tiling approach is able to generate solutions for all the three tasks, demonstrating the scalability of our approach with respect to the number of allowable tile shapes. Moreover, for each task, our inverse tiling approach is able to generate a tiling solution with the smallest number of prototiles, showing the superior performance of our approach in terms of minimizing K.

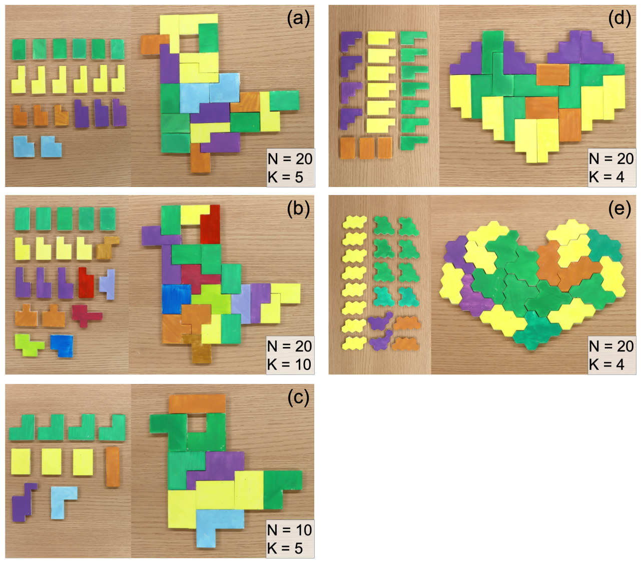
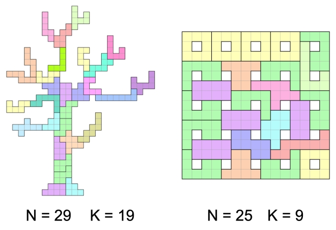


Fig. 14. 2D assembly puzzles designed using our inverse tiling approach. The level of difficulty of the puzzles is related to (a&c) the number (N) of tiles, (a&b) number (K) of prototiles, and (d&e) grid type (square vs hexagon). For each puzzle, we show its component pieces and assembled shape.

Application to puzzles. Our inverse tiling approach can be used to design 2D assembly puzzles, where many puzzle pieces have congruent shapes. We fabricated our tiling results using 3D printing and painted the fabricated tiles with the same shape using the same color; see Figure 14. We conducted an informal user study, in which six participants were asked to play with the five puzzles by assembling the tiles to form the target shape. The participants solved most puzzles within 10 to 50 minutes. We observed that a puzzle's level of difficulty is related to the number of pieces (i.e., number of tiles N), number of distinct pieces (i.e., number of prototiles K), and grid types (square grid vs hexagon grid). Our inverse tiling approach allows controlling all these aspects, enabling one to design a personalized puzzle with varying levels of difficulty. Please refer to Supp. 7 for details about the user study.

Limitations. Our inverse tiling approach has some limitations. First, our approach may not be able to generate a K-hedral tiling result when the constraints specified on the prototile size/shape are too restrictive. Figure 12 (leftmost) shows a failure example of our approach when searching for a tiling result under a very tight constraint on the prototile size (i.e., $C_{\min} = C_{\max} = 10$). This issue can be alleviated by relaxing the constraints on the prototile size/shape; see Figure 12. Second, our approach may fail to generate tiling results with a small number of prototiles for input domains that do not have a large internal area since enlarging tiles congruently may easily fail on such domains; see the inset (left) for an example. Third, our approach lacks an understanding of the input domain's global shape properties such as symmetry. Hence, it is not able to leverage these global shape properties for minimizing the number of prototiles; see the inset (right) for an example where a $K = 1$ tiling result exists.



7 Conclusion

This paper presents inverse tiling of 2D finite domains, a new perspective to creating K-hedral tilings, where the tile shapes are inversely modeled to fit the input domain rather than being prescribed. To minimize the number (K) of prototiles, our approach first constructs a small prototile set and then further reduces the set. Experimental results show that our inverse tiling approach is able to tile domains of various shapes and forms. Comparisons with forward tiling approaches show that our inverse tiling approach is able to generate tiling results with a smaller prototile set and is more scalable with respect to the number of allowable tile shapes.

Future work. Our work opens up many interesting directions for future research. First, our inverse tiling approach can be generalized for Escherization by adding figures to the generated tiles. Second, our approach focuses on the problem of tiling 2D finite domains. In the future, we would like to generalize our approach to tile 3D finite domains such as voxelized shapes. Third, we plan to study applying inverse tiling for rationalization of architecture, by modeling architectural assemblies composed of parts with congruent shapes [Chen et al. 2023]. Lastly, we are interested in extending our inverse tiling approach to model layouts with deformable prototiles [Peng et al. 2014] and exploring its applications in urban layout design and pattern generation [Jiang et al. 2015].

Acknowledgments

We thank the reviewers for their valuable comments. This work was supported by the Singapore MOE AcRF Tier 2 Grant (MOE-T2EP20123-0016).

References

Abdalla G. M. Ahmed. 2019. Sampling with Pinwheel Tiles. In *Proc. the 37th Annual Conference on Computer Graphics and Visual Computing*. 147–148.

Gill Barequet and Mira Shalah. 2022. Improved Upper Bounds on the Growth Constants of Polyominoes and Polycubes. *Algorithmica* 84 (2022), 3559–3586.

Rulin Chen, Pengyun Qiu, Peng Song, Bailin Deng, Ziqi Wang, and Ying He. 2023. Masonry Shell Structures with Discrete Equivalence Classes. *ACM Trans. on Graph. (SIGGRAPH)* 42, 4 (2023), 115:1–115:12.

Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. 2003. Wang Tiles for Image and Texture Generation. *ACM Trans. on Graph. (SIGGRAPH)* 22, 3 (2003), 287–294.

Marcus R. Garvie and John Burkardt. 2020. A New Mathematical Model for Tiling Finite Regions of The Plane with Polyominoes. *Contributions to Discrete Mathematics* 15, 2 (2020), 95–131.

Marcus R. Garvie and John Burkardt. 2022. A Parallelizable Integer Linear Programming Approach for Tiling Finite Regions of the Plane with Polyominoes. *Algorithms* 15, 5 (2022), 164:1–164:41.

Solomon W. Golomb. 1966. Tiling with Polyominoes. *Journal of Combinatorial Theory* 1, 2 (1966), 280–296.

Solomon W. Golomb. 1970. Tiling with Sets of Polyominoes. *Journal of Combinatorial Theory* 9, 1 (1970), 60–71.

Solomon W. Golomb. 1989. Polyominoes Which Tile Rectangles. *Journal of Combinatorial Theory, Series A* 51 (1989), 117–124.

Solomon W. Golomb. 1994. *Polyominoes: Puzzles, Patterns, Problems, and Packings (Revised and Expanded Second Edition)*. Princeton University Press.

Solomon W. Golomb. 1996. Tiling Rectangles with Polyominoes. *The Mathematical Intelligencer* 18, 2 (1996), 38–47.

Branko Grünbaum and Geoffrey Colin Shephard. 2016. *Tilings and Patterns (Second Edition)*. Dover Publications.

Caigui Jiang, Chengcheng Tang, Amir Vaxman, Peter Wonka, and Helmut Pottmann. 2015. Polyhedral Patterns. *ACM Trans. on Graph. (SIGGRAPH Asia)* 34, 6 (2015), 172:1–172:12.

Craig S. Kaplan. 2009. *Introductory Tiling Theory for Computer Graphics*. Springer.

Craig S. Kaplan and David H. Salesin. 2000. Escherization. In *Proc. SIGGRAPH*. 499–510.

Craig S. Kaplan and David H. Salesin. 2004. Dihedral Escherization. In *Proc. Graphics Interface*. 255–262.

Naoki Kita. 2023. Dissection Puzzles Composed of Multicolor Polyominoes. *Comp. Graph. Forum (Pacific Graphics)* 42, 7 (2023), e14968:1–e14968:9.

Naoki Kita and Kazunori Miyata. 2021. Computational Design of Polyomino Puzzles. *The Visual Computer (CGI)* 37, 4 (2021), 777–787.

Donald E. Knuth. 2000. Dancing Links. In *Millennial Perspectives in Computer Science*. 187–214.

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-Time Blue Noise. *ACM Trans. on Graph. (SIGGRAPH)* 25, 3 (2006), 509–518.

Ares Lagae and Philip Dutré. 2008. A Comparison of Methods for Generating Poisson Disk Distributions. *Comp. Graph. Forum* 27, 1 (2008), 114–129.

Xiaokang Liu, Lin Lu, Lingxin Cao, Oliver Deussen, and Changhe Tu. 2024. Auxetic Dihedral Escher Tessellations. *Graphical Models* 133 (2024), 101215:1–101215:10.

Xiaokang Liu, Lin Lu, Andrei Sharf, Xin Yan, Dani Lischinski, and Changhe Tu. 2020. Fabricable Dihedral Escher Tessellations. *Computer-Aided Design* 127 (2020), 102853:1–102853:9.

Kui-Yip Lo, Chi-Wing Fu, and Hongwei Li. 2009. 3D Polyomino Puzzle. *ACM Trans. on Graph. (SIGGRAPH Asia)* 28, 5 (2009), 157:1–157:8.

Merel Meekes and Amir Vaxman. 2021. Unconventional Patterns on Surfaces. *ACM Trans. on Graph. (SIGGRAPH)* 40, 4 (2021), 101:1–101:16.

Dávid Mišiak. 2022. *Computational Approaches to Polyomino Tiling Problems.* Bachelor's Thesis. Comenius University in Bratislava, Bratislava, Slovakia. Source code at https://github.com/davidmisiak/tiler.

Cristopher Moore and John Michael Robson. 2001. Hard Tiling Problems with Simple Tiles. *Discrete & Computational Geometry* 26, 4 (2001), 573–590.

Joesph Myers. 2019. Polyomino, polyiamond, and polyhex tiling. https://www.polyomino.org.uk/mathematics/polyform-tiling/.

Yuichi Nagata and Shinji Imahori. 2021. Escherization with Large Deformations Based on As-Rigid-As-Possible Shape Modeling. *ACM Trans. on Graph.* 41, 2 (2021), 11:1–11:16.

Yuichi Nagata and Shinji Imahori. 2024. Creation of Dihedral Escher-like Tilings based on As-Rigid-As-Possible Deformation. *ACM Trans. on Graph.* 43, 2 (2024), 18:1–18:18.

Victor Ostromoukhov. 2007. Sampling with Polyominoes. *ACM Trans. on Graph. (SIGGRAPH)* 26, 3 (2007), 78:1–78:6.

Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. 2004. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Trans. on Graph. (SIGGRAPH)* 23, 3 (2004), 488–495.

Chi-Han Peng, Caigui Jiang, Peter Wonka, and Helmut Pottmann. 2019. Checkerboard Patterns with Black Rectangles. *ACM Trans. on Graph. (SIGGRAPH Asia)* 38, 6 (2019), 171:1–171:13.

Chi-Han Peng, Helmut Pottmann, and Peter Wonka. 2018. Designing Patterns using Triangle-Quad Hybrid Meshes. *ACM Trans. on Graph. (SIGGRAPH)* 37, 4 (2018), 107:1–107:14.

Chi-Han Peng, Yong-Liang Yang, , and Peter Wonka. 2014. Computing Layouts with Deformable Templates. *ACM Trans. on Graph. (SIGGRAPH)* 33, 4 (2014), 99:1–99:11.

D. Hugh Redelmeier. 1981. Counting Polyominoes: Yet Another Attack. *Discrete Mathematics* 36, 2 (1981), 191–203.

David Smith, Joseph Samuel Myers, Craig S. Kaplan, and Chaim Goodman-Strauss. 2024. An Aperiodic Monotile. *Combinatorial Theory* 4, 1 (2024), 6:1–6:91.

Milan Sonka, Vaclav Hlavac, and Roger Boyle. 2011. *Image Processing, Analysis and Machine Vision.* Springer.

Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. 2020. TilinGNN: Learning to Tile with Self-supervised Graph Neural Network. *ACM Trans. on Graph. (SIGGRAPH)* 39, 4 (2020), 129:1–129:16.