Computing Layouts with Deformable Templates

Chi-Han Peng* Arizona State University
Yong-Liang Yang† KAUST
Peter Wonka‡ Arizona State University / KAUST

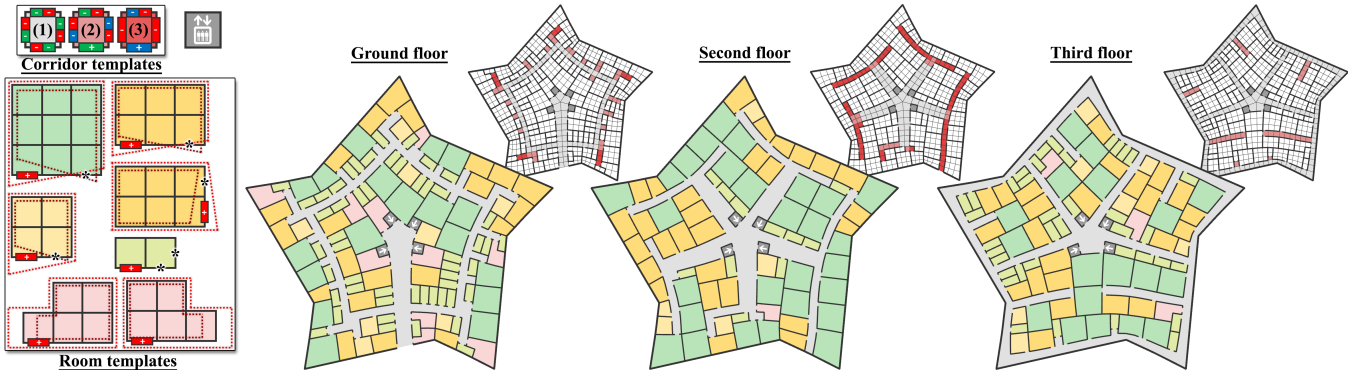

Figure 1: *Our framework is able to generate layouts that meet both accessibility and aesthetic criteria for arbitrarily shaped (e.g., non-axis aligned) domains. Here, we show several floorplan layouts for a pentagon-shaped building. The accessibility criteria specify that all rooms are connected to the elevators via a singly connected corridor. The aesthetic criteria specify the admissible deformations for the rooms. Upper-left: A set of tile templates to build the corridor: (1) predefined locations of the corridor, (2) first-level branches of the corridor, and (3) second-level branches of the corridor. Observe how these topological constraints are enforced by the the edge-color constraints of the tiles. Bottom-left: A set of deformable room templates. The doors are constrained to be at the red edges (alternative locations are marked with a star). Certain rooms are deformable to be quads with two right angles. Right: We show three distinct floorplan designs for different floors of the building. The different designs are achieved by different predefined locations of the corridor (grey parts in the sub-figures in the upper-right corners). The corridor branches and the rooms are then found by our discrete tiling algorithm in a way such that rooms are placed with a higher priority. Finally, the mesh geometry is optimized to minimize the deviations of the rooms from their admissible shapes and the straightness of the room boundaries..*

Abstract

In this paper, we tackle the problem of tiling a domain with a set of deformable templates. A valid solution to this problem completely covers the domain with templates such that the templates do not overlap. We generalize existing specialized solutions and formulate a general layout problem by modeling important constraints and admissible template deformations. Our main idea is to break the layout algorithm into two steps: a discrete step to lay out the approximate template positions and a continuous step to refine the template shapes. Our approach is suitable for a large class of applications, including floorplans, urban layouts, and arts and design.

CR Categories: I.3.5 [Object Modeling]

Keywords: tiling, pattern synthesis, quadrilateral meshes

Links: DL PDF WEB

*e-mail:pchihan@asu.edu
†e-mail:yongliang.yang@kaust.edu.sa
‡e-mail:pwonka@gmail.com

1 Introduction

Layout computation is an essential aspect of computational design. Recent papers tackle various layout problems, such as the layout of texture atlases, streets and parcels, buildings, cartographic maps, floorplans, facades, mosaics, furniture, golf courses, and artistic packings.

We can distinguish two main categories of layouts. The first type allows gaps between templates, but restricts object deformations (typically only rigid deformations are allowed). Examples for this type of layout include furniture [Yu et al. 2011], collage [Huang et al. 2011], mosaics [Kim and Pellacini 2002], and artistic packings [Reinert et al. 2013]. The second type of layout requires that the domain be fully covered without overlap, but tolerates a more general class of template deformations. There exist several specialized versions of this layout problem, most notably tiling a domain or mesh surface with triangles or quads. Other examples are axis-aligned floorplans [Merrell et al. 2010] and urban layouts [Yang et al. 2013]. In this paper, we contribute to the computation of the second type of layouts (called *water-tight* layouts in this paper).

Our first contribution is to model the problem. We generalize existing specialized problem formulations and propose an optimization framework that can be applied to a larger class of applications, allowing for more general templates, and providing more flexibility in specifying desirable solutions. We analyze a set of example applications and propose a way to model the most important constraints and admissible tile deformations in an optimization framework.

Our second contribution is to propose a technical solution to the water-tight layout problem. It is important to note that the first category of layouts is typically much easier to compute because empty

space is allowed. That enables solutions that are built on local mutations (adding, deleting, shifting of templates), e.g., simulated annealing [Yu et al. 2011] and rjMCMC [Yeh et al. 2013], because one valid layout can easily be transformed into another valid layout. By contrast, water-tight layouts do not admit local transformations that can explore the space of valid layouts. Existing approaches use top-down subdivision [Yang et al. 2013] or greedy placement [Park et al. 2007]. Both approaches can be randomized by backtracking or attempting many randomized layouts from scratch. However, none of these approaches can be generalized to our problem formulation. Top-down subdivision cannot sufficiently control the shapes in the resulting domain partition and greedy placement often cannot cover the complete domain. In general, it is actually very difficult to find a single valid solution to a water-tight layout problem. Our idea is to use a two-stage approach. First, we discretize the domain and tile templates into meshes. We can then formulate the approximate placement and deformation of templates in a global optimization framework. Second, we refine the shapes of the templates using continuous optimization.

The third contribution of our work is to show how several layout applications can be formulated by our framework to achieve novel results that cannot be computed with existing layout algorithms.

1.1 Related Work

There are multiple ways to characterize layout algorithms. In our discussion, we classify the work based on the allowable deformation of the elements and the allowed gaps between elements.

The first class of layout algorithms we consider includes algorithms that do not require the domain to be fully covered, i.e., gaps between elements are allowed. For example, in furniture arrangement [Yu et al. 2011; Merrell et al. 2011], shelf filling [Majerowicz et al. 2014], or the design of golf courses [Yeh et al. 2012], the amount of space can be quite large and the challenge of the layout stems from the constraints between different elements. Other layout problems, such as artistic packing layouts [Reinert et al. 2013], decorative mosaics [Hausner 2001], or texture atlas packing [Lévy et al. 2002], try to minimize empty space and can therefore be described as packing problems. Most of these problems can be solved by stochastic methods, such as simulated annealing, MCMC, or rjMCMC.

The second class of layout algorithms allows gaps and deformations. An example of this class is jigsaw image mosaics [Kim and Pellacini 2002] because these mosaic elements are allowed to deform slightly. In the broader sense, probabilistic shape synthesis from part collections could also be considered as layout algorithms in this class. An excellent representative of this type of work is the paper by Kalogerakis et al. [Kalogerakis et al. 2012].

The third class of layout algorithms neither allows gaps nor tile deformations. That usually requires a simple, e.g., rectangular, boundary that is cut into rectangular or square tiles. In the simplest form, elements are exactly the size of one square. That makes it easy to fill the domain, but tiling is typically restricted by relationships between elements, e.g., Wang Tiles. Wang Tiles have been nicely applied in computer graphics for texture synthesis [Cohen et al. 2003] and blue noise generation [Kopf et al. 2006]. The solution space can be further restricted by introducing mandatory neighborhood relationships between tiles. Yeh et al. [Yeh et al. 2013] used examples in facade and urban modeling to illustrate their tiling algorithm that can encode hard and soft constraints. A complexity analysis for tiling was presented by Demaine and Demaine [2007]. In [Fasano 2004], the packing of Tetris-like items in 3D space is modeled as an MIP problem but solved only heuristically. Prokopyev and Karademir formulate the task of tiling a regular grid using polyominos, i.e., singly connected joints of squares, into an integer programming problem [Prokopyev and Karademir 2012]. An interesting link to the next class is the question of how to transform a set of elements so that they can tile a domain. Escherization [Kaplan and Salesin 2000] is a fascinating concept related to this question. We refer to the book by Kaplan [Kaplan 2009] for a broader discussion of tiling theory from a computer graphics perspective.

The fourth class of layout algorithms does not allow gaps, but it does allow elements to deform. One class of such layouts fills a domain with rectangles of different size. A good example is residential building layouts [Merrell et al. 2010]. While it is easy to fully cover the domain, required neighborhood relationships constrain the problem and make it difficult to solve. It is not clear how these building layout algorithms can be extended to non-rectangular rooms, a challenge we want to tackle in this paper. Facade layouts can also be generated by resizing boxes [Lin et al. 2011; Dai et al. 2013; Bao et al. 2013]. Urban layout design [Aliaga et al. 2008b; Aliaga et al. 2008a; Yang et al. 2013] is an example problem in which the boundary of the domain is more complex and the elements can be deformed in a more general fashion. Because it is difficult to find a single valid solution, Yang et al. used hierarchical splitting and backtracking. Our method can improve upon this previous work by allowing for better control of the distribution of and neighborhood relationships between tiles. Splitting techniques are also useful for computing parcel layouts [Vanegas et al. 2012]. The most popular layout problem with deformable elements is actually quad and triangle meshing. Early work, e.g., [Blacker and Stephenson 1991], tried to fill the domain by incrementally adding quads. These paving-style algorithms illustrate how difficult the problem actually is. The geometric quality usually suffers when two fronts collide with each other and gaps of odd shapes have to be filled. We refer to the survey by Bommes et al. [Bommes et al. 2012] for a broader discussion on quad meshing.

2 Framework Overview

We begin by describing the problem statement as follows. The problem domain is given as a 2D polygon with arbitrary numbers of boundaries defined as closed simple piecewise linear curves. The goal of this framework is to *completely cover* the 2D domain with smaller polygons that have a disk-like topology, each we call a *tile*, such that the geometry of each tile is as close to one prescribed template as possible.

The definition of a tile template is important to our paper. In short, a tile template is defined as one base polygon (with a disk-like topology) under certain admissible transformations. For example, a tile template presenting arbitrary rectangles can be defined as a unit square under two scalings along the square's two edge directions. A detailed definition is given in Section 3.

As mentioned previously, finding valid solutions to the water-tight layout problem in the continuous sense is difficult. Our solution is to break the problem into a discrete part and a continuous part. First, we tessellate both the problem domain and the base polygons of the tile templates into two-manifold polygon meshes. In this way, the continuous complete cover problem is transformed into a discrete *tiling* problem, which is formulated as linear integer (Boolean) programming (Section 4). Second, given a discrete tiling found in the previous step, we consider the positions of the vertices on the tessellated problem domain as continuous variables and solve an optimization problem to further improve the geometrics of the tiles (Section 5). See Figure 2 for an overview.

Quad Mesh Tessellation: We choose (pure) quad meshes as the type of meshes for the discretization. This decision is mainly based on the chosen example applications, e.g., street patterns and floor

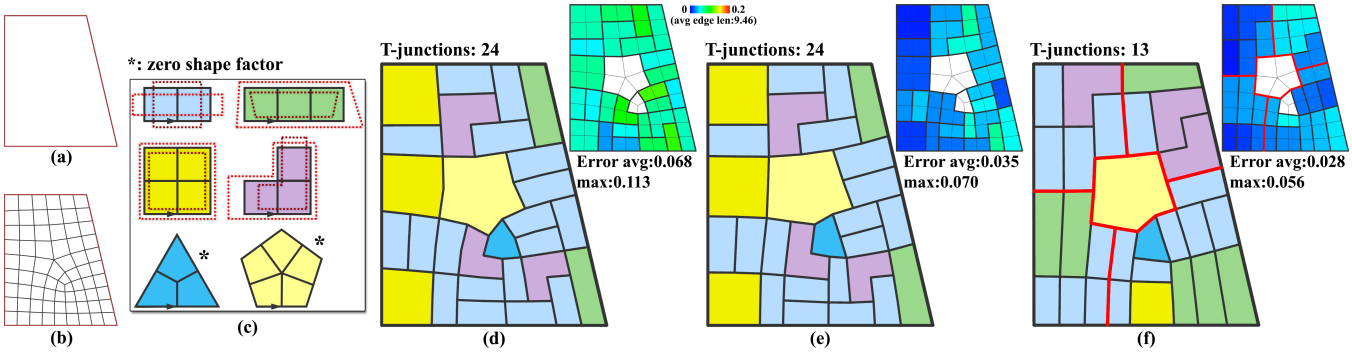

Figure 2: *Overview of our framework. (a) and (b) The problem domain and its quadrangulation. (c) The given tile templates. Each template is shown as its base polygon and its admissible transformations (upper left: rectangles with arbitrary ratios of side lengths, upper right: trapezoids with parallel top and bottom sides, middle left: squares, middle right: J-shaped tiles with arbitrary ratios of lengths of the two ends, bottom left and right: these triangle and pentagon tiles can be arbitrarily transformed. Arrows denote the anchors. (d) A complete tiling with a small average shape registration error is generated by linear integer (Boolean) programming with shape factors included in the weighting scheme. Note that the shapes of the triangle and pentagon tiles are irrelevant (zero shape factors). The shape registration errors are visualized in the upper-right corners. (e) The mesh geometry is optimized to further reduce the shape registration errors. (f) A custom tiling design with user-specified tile boundary constraints (red lines) and regular-junction constraints added to reduce the number of T-junctions.*

plans. We consider tiling on other kinds of meshes, such as triangle meshes and quad-dominant meshes, as interesting directions for future work.

We use the following definitions for a quad mesh. A vertex with valence 4 is considered as *regular*; otherwise, it is *irregular*. For each edge, we distinguish two half-edges that are *opposite* to each other. A half-edge is a *border* half-edge if it does not have a face; otherwise, it is a *non-border* half-edge. We assume that the half-edges circulate around the faces in counter-clockwise order.

3 Tile Templates

A tile template, τ_x, $0 \leq x < N$, where N is the number of tile templates, defines the admissible shapes for a tile. It is defined as a base polygon with a disk-like topology, B_x, and a specification of its admissible transformations, defined later in this section. We first discuss the definition of base polygons.

Recall that we quadrangulate the problem domain and the base polygons into quad meshes prior to the discrete tiling. An essential task of the discrete tiling is to recognize identical copies of the base polygons in the quadrangulated problem domain. This is a graph isomorphism problem, which becomes non-trivial when irregular vertices are considered. For simplicity, we assume that a base polygon (a quad mesh) may have up to one irregular vertex in its interior of which the valence is not a multiple of 4. In this way, the quadrangulation is unique (if it exists) given a prescribed boundary loop configuration. We now define a base polygon as follows:

Definition 3.1 *The base polygon of a tile template is a quad mesh with a disk-like topology, i.e., with exactly one boundary and no handles. The* anchor *of a tile template is one particular non-border half-edge along the base polygon's boundary.*

In practice, a base polygon is specified by its boundary loop alone (see Figure 4a for an example):

Definition 3.2 *The boundary loop of a quad mesh with a disk-like topology, e.g., a base polygon, is defined as the number of boundary edges, plus the number of inner edges adjacent to each boundary vertex in counter-clockwise order, starting at the boundary vertex pointed to by the anchor.*

3.1 Admissible Transformations

An admissible transformation is defined by a sequence of k transformation steps. A transformation step is either a similarity or a rigid transformation, a translation, a rotation, a scaling, or a shearing. The transformation steps only need to be applied to the boundary vertices of a base polygon. The positions of inner vertices are irrelevant and can be found by Laplacian smoothing for visualization purposes. A unique characteristic of our framework is that transformation steps can be specified such that they affect only a subset of vertices. In this way, we can build many interesting non-linear transformations, such as bending.

Next, we describe the individual transformation steps. As a base polygon undergoes a sequence of transformation steps, the next transformation is often defined with respect to the base polygon's current position. A translation can be unconstrained or constrained along one direction determined by two particular boundary vertices. A rotation is done around a center that is determined by one particular boundary vertex. A scaling or a shearing is done along a direction and a center, determined by two particular boundary vertices (for the direction) and one particular boundary vertex (for the center).

Since we use a unique transformation model, we show examples in Figure 3 to provide intuition on how this model works in practice.

4 Discrete Tiling

Problem Domain Quadrangulation: Given a collection of tile templates (and their base polygons as quad meshes), we quadrangulate the problem domain such that the edge lengths and the aspect ratios of quads roughly match the base polygons. These criteria ensure that we can find graph-isomorphic copies of the base polygons in the quadrangulated domain without large shape deviations. We use the patch-wise quadrangulation algorithm in [Peng et al. 2014] for its capability to produce semi-regular, i.e., most vertices are regular, quadrangulations for domains of arbitrary boundaries.

We now assume the problem domain has been quadrangulated into a quad mesh, M. A *tile* on M is defined as follows.

Definition 4.1 *A* tile, $T_{i,x}$, *is a simply connected set of faces on* M,

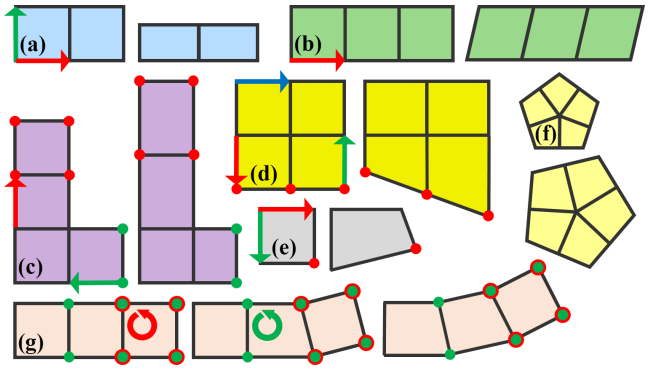

Figure 3: *Transformations for tile templates. (a) Two global scalings along the two edge directions (red and green arrows). The centers for scalings and shearings are at the ends of the arrows. (b) A global shearing along the red edge direction. (c) Two translations for two subsets of vertices (red and green), each is constrained to be along one edge direction. (d) A shearing for a subset of vertices (red) along the red edge direction, followed by a translation along the green edge direction and then a scaling along the blue edge direction for the same subset of vertices. (e) A free translation for a vertex (red), followed by two global scalings along the red and green edges directions. (f) A global similarity transformation. (g) A counter-clockwise bending that preserves the right angles of the corners is approximated by two consecutive counter-clockwise rotations for the red and the green vertices in respective order.*

which is enclosed by a closed loop the same as the boundary loop of B_x (τ_x's base polygon), starting at a half-edge, e_i.

Tile $T_{i,x}$ can be understood as identifying a graph-isomorphic copy of B_x (τ_x's base polygon) on M while aligning τ_x's anchor edge with edge e_i. We say that a tile is *degenerate* if its boundary loop traverses an edge more than once (Figure 4a). We exclude degenerate tiles unless otherwise specified. We now define a *tiling* on M in the following.

Definition 4.2 *A* tiling *on M is a complete or partial cover of M's faces into non-overlapping tiles.*

Finding a tiling can be understood as finding a non-overlapping subset of all possible potential tile placements on M. Similar to the maximal and maximum matchings in graph theory, a tiling is *maximal* if any simply connected subsets of uncovered faces cannot be covered by any tiles. A maximal tiling can be easily found by flooding M with tiles in an arbitrary order. A tiling is *maximum* if it has the smallest possible number (ideally zero) of uncovered faces among all possible tilings on M. A maximum tiling is also maximal, but the reverse is not necessarily true. A tiling is *complete* if all faces are covered.

Integer Programming: We formulate the tiling problem as a linear integer (Boolean) program. We first enumerate all possible tile placements on M. This is done by enumerating all possible combinations of templates and half-edges (as anchors) in the mesh that lead to valid tiles. Note that the positions of templates with inner irregular vertices are limited by the corresponding irregular vertices in the mesh. Now, for every potential placement of tile $T_{i,x}$, we create a Boolean variable of the same name, indicating the presence of the tile in the tiling. For a tiling to be valid, overlapping tiles cannot be present concurrently. This is modeled by adding constraints: $\sum_{T_{i,x} \ni f_k} T_{i,x} \leq 1$, for every face, f_k, on M. We can replace $\leq$ with the $=$ sign if we want to find complete tilings only. The objective function to maximize is modeled as $\sum_{i,x} W_{i,x} T_{i,x}$, where $W_{i,x}$ is the weight associated with $T_{i,x}$. We assume that tile weights are non-negative.

We are now ready to model the tiling problem in linear programming form:

$$\text{Maximize} \quad \sum_{i,x} W_{i,x} T_{i,x} \tag{1}$$

$$\text{Subject to} \quad \sum_{T_{i,x} \ni f_k} T_{i,x} \leq 1, \text{ for every face } f_k \text{ on } M.$$

For a maximum tiling, we set $W_{i,x}$ to be the number of faces in B_x. In this way, the objective value to maximize amounts to the total number of faces covered by the tiling.

Weighting Scheme: We generalize the weighting scheme as follows:

$$W_{i,x} = W_x * factor_0(T_{i,x}) * factor_1(T_{i,x}) \ldots \tag{2}$$

W_x denotes the weight of tile template τ_x, which by default is equal to the number of faces in B_x. W_x can be adjusted to suit user-preferences for each tile template (see Figure 5 for an example). The factors are functions to diverse weights of tiles of the same tile template, to suit application-specific needs, such as:

- A randomization factor, e.g., a random variable ranging from α to 1, $0 \leq \alpha \leq 1$ (the lower the number, the stronger the randomization), to randomize the tilings.
- A shape factor based on the shape registration error of B_x to $T_{i,x}$ (Section 5.1) to give tiles with lower registration errors higher weights and vice versa.

Note that each factor may have different sensitivities to different tile templates. For example, the shape factor can always return 1 for tile templates of which the shapes are irrelevant.

Occurrence Constraints: It is straightforward to impose a lower bound and/or an upper bound of the number of occurrences for tiles of a particular tile template by adding linear constraints: $\sum_i T_{i,K} \geq \alpha$, $\sum_i T_{i,K} \leq \beta$, α and β are the lower and upper bounds of the number of occurrences for tile template τ_K.

4.1 Adjacency Constraints

To impose functional criteria on layouts, it is useful to be able to constrain how tiles are adjacent to each other in a tiling. A commonly used strategy, e.g., edge color-matching constraints in Wang tiles [Cohen et al. 2003], is to assign colors to boundary half-edges of all tile templates' base polygons and require that, in a tiling, every pair of opposing half-edges on M must have the same color. In addition, we allow the color to be *signed* in the sense that a pair of opposing half-edges must have not only the same color but also the opposite signs. Alternatively, the signed color constraints can be enforced only for the edges with positive-signed colors. Lastly, the edge color-matching constraints can also be *soft* in the sense that mis-matchings are less preferred but still admissible. The same concepts can be directly applied to vertices.

It is straightforward to model hard constraints in integer programming, for example, tiles that cannot be present concurrently due to the adjacency constraints are constrained to have a joint occurrence of up to one. However, a direct attempt to model soft constraints would involve higher-order terms, e.g., adding multiplications of Boolean variables presenting tiles that we prefer to appear at the

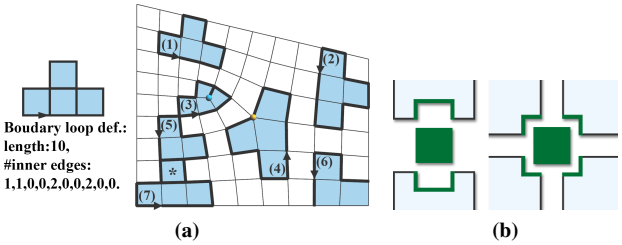

Figure 4: *(a) Potential tile placements of a T-shaped template. Arrows denote the anchors. (1), (2), (4), (5), and (7): admissible tile placements. (3): A degenerate case. (6): Not admissible because the boundary loop cannot form at the anchor. (5) and (7): Two tile placements that overlap at the marked face. In a tiling, at most one of them can be present concurrently. (b) The intuition of using joints to model soft constraints. Left: For an edge-based joint to be present in a tiling, the two adjacent tiles (top and bottom) must have the matching color. Right: For a vertex-based joint to be present in a tiling, all adjacent tiles' boundary vertices (four in this case) must have matching colors.*

same time to the objective function. This approach is especially undesirable for vertices-based constraints. For example, assuming T_0, T_1, T_2, and T_3 are Boolean variables presenting four tiles having adjacent boundary vertices of matching color, we would add a quartic term to the objective function, $T_0T_1T_2T_3$, that equals one only when all four tiles are present concurrently. Inspired by the *joints* commonly used in woodworking, we propose a scheme to model soft constraints using linear terms as follows.

4.1.1 Soft Edge-Based Constraints

We first define imaginary elements called *joints* as follows. For every non-border edge on M, E_i, there exist k joints, $J_{i,j}$, where $0 \leq j < k$ is the color index of the joint and k is the total number of colors. With the same intuition as for the woodworking joints, the necessary condition for $J_{i,j}$ to be present in a tiling is that the two tiles adjacent to E_i both have the matching color (Figure 4b, left). In other words, $J_{i,j}$ cannot be present concurrently with every tile that is adjacent to E_i but does not have the matching color. We then model the edge joints into the integer programming as Boolean variables of the same name and extend the objective function as:

$$\text{Maximize} \quad \sum_{i,x} W_{i,x}T_{i,x} + \sum_{i,j} W_J J_{i,j}, \tag{3}$$

where $0 < W_J$ is the weight for edge joints. Now, the integer programming also optimizes the number of edge joints that appeared in a tiling in a weighted sense (determined by W_J), which is roughly reversely proportional to the number of mis-matched edge colors.

4.1.2 Soft Vertex-Based Constraints

We can assign a color to every boundary vertex of all tile templates' base polygons and model soft color-matching constraints for vertices in a similar fashion (Figure 4b, right). The integer programing now also optimizes the number of vertices on M such that the colors of all its adjacent tiles' boundary vertices match. A useful scenario is described in the following.

Regular Junction-Matching Constraints: A tiling distinguishes a subset of edges on M that are parts of the tile boundaries, which we call the *boundary network* defined by the tiling. Interestingly, based on our assumptions that the problem domain tessellation is two-manifold and that degenerate-case tiles are excluded, a boundary network resembles a coarser polygonal mesh of faces of arbitrary degrees (i.e., the tiles) imposing on M. From this perspective, the connectivity, i.e., the valence of the vertices on the boundary network, becomes a quality criteria for tilings.

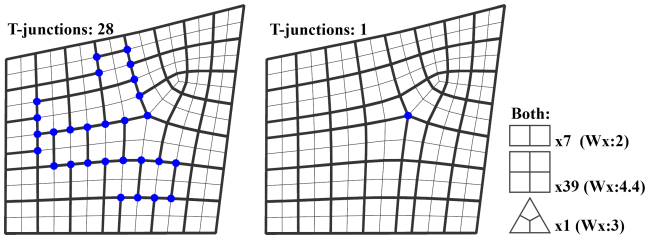

Figure 5: *We can improve the regularity of the resulting boundary networks of tilings by imposing regular-junction constraints. Here we show two tilings with the same numbers of tiles, without (left) and with (right) regular-junction constraints. Note that we use a higher weight for the 2x2 templates to compute tilings of which 2x2 templates are placed with a higher priority than other templates.*

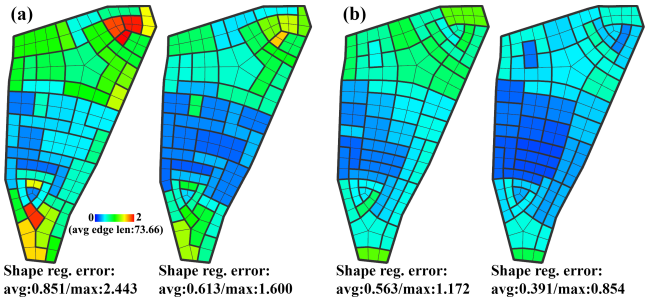

Figure 6: *(a) Left: A tiling generated without shape factors in the weighting scheme, before (left) and after (right) a shape optimization. (b) A tiling generated with shape factors included in the weighting scheme. In summary, by including shape factors into the discrete tiling calculation, we can compute tilings with smaller shape registration errors.*

For a tiling, a *regular junction* is a regular non-border vertex on M with the number of adjacent tile boundary edges equal to 4. Note that the vertex resembles a T-junction on the boundary network if the number equals 3 and a degenerated vertex if the number equals 2 or 1. Depending on the application, we may prefer tilings with more regular junctions. Since it is often not possible to have a tiling in which all junctions are regular, we impose this preference in a soft sense. For this goal, for the base polygon of every tile template, we assign a particular color to every *convex corner*, i.e., a boundary vertex adjacent to no inner edge on the base polygon. In this manner, the number of regular junctions, which are formed by four adjacent convex corners, is also optimized by integer programming. See Figure 5 for an example.

5 Geometric Optimization

Recall that the goal of our framework is to completely tile a domain with templates, each of which can transform in a specific way defined by a sequence of transformation steps (see Section 3.1). In practice, a perfect solution may not be feasible. We instead look for solutions such that the sum of squared distances between vertices of the transformed templates and the corresponding tiles on M is minimized, weighted by each tile's sensitivity to its shape.

We achieve this goal in two stages. First, we include the shape error of each potential tile placement, found by a method to register the shape of a template to the corresponding tile under admissible transformations (Section 5.1), into the weighting scheme of the discrete tiling method. In this way, tilings with lower sums of shape errors can be computed. See Figure 6 for a comparison. Second, given a tiling with tiled templates, we further optimize vertex positions of M, such that the boundary of each tile favors the shape of its corresponding template under the predefined admissible transformations in a local/global sense (similar to the projection-based approach in [Bouaziz et al. 2012]).

5.1 Shape Registration

Given a template's base polygon, its sequence of transformation steps, and a corresponding tile on M with a one-to-one correspondence of the boundary vertices, we register the base polygon to the tile under the predefined sequence of transformation steps in such a way that the sum of squared distances between corresponding vertices is minimized. We register the sequence in the reverse order. Each transformation is registered by solving a least-squares system of the squared distances between corresponding vertices. This system has a closed-form solution. Details can be found in Appendix A. We denote the error of a shape registration as the average of the distances between corresponding vertices in the base polygon and the corresponding tile.

For transformations consisting of multiple steps, e.g., bending, a single pass of registration may not achieve the desirable quality. Since all transformation steps are based on relative coordinate systems, i.e., centers and directions, we can register the sequence of transformation steps repeatedly, in which each pass uses the transformed base polygon from the previous pass, until convergence or a time limit is reached. The resulting transformed base polygons typically have lower registration errors and still approximate the ground truths of single-pass transformations nicely, depending on the specific transformation step sequences. Convergence (in terms of the shape registration error) is guaranteed because the registration of each step either decreases or maintains the error. An analysis is shown in Figure 4 in the additional materials.

In summary, the main gist of our transformation framework is to build non-linear transformations, which are generally non-convex and difficult to register, by sequences of transformation steps such that each can be registered with a convex, closed-form solution.

Size-preserving constraints: The magnitude of each transformation step, e.g., the distance of a translation, the angle of a rotation, and the factor of a scaling or shearing, can be constrained to lie within a specified range. An alternative method is to apply an additional non-uniform scaling to the transformed base polygon to constrain its bounding box to be within an acceptable range. All our examples are generated with this method.

5.2 Global Shape Optimization

Given a tiling on M, we find the registered shape of the base polygon for each tile by the aforementioned method. We denote the position of the k-th vertex (beginning at the one pointed to by the anchor) of the tile's registered base polygon as $V_{i,x,k}$. We say that $V_{i,x,k}$ is a registered position for the k-th vertex of tile $T_{i,x}$ on M. Each vertex on M, v_n, $0 \leq n < N$, where N is the number of vertices on M, can correspond to multiple registered positions, one for each adjacent tile of which v_n is a boundary vertex. These registered positions may not agree. There exist many ways to determine the position of v_n given its registered positions, for example, by moving to the one that is closest to v_n's current position, i.e., snapping. We choose a weighted average scheme for its robustness and simplicity, formulated as a continuous quadratic optimization problem:

$$\text{Minimize} \quad \sum_{n,i,x} W_{i,x}(v_n - R_{n,i,x})^2 \tag{4}$$

where $W_{i,x}$ is the weight for tile $T_{i,x}$ and $R_{n,i,x}$ is the registered position for vertex v_n by tile $T_{i,x}$. Note that a tile does not contribute more than one position to a vertex because we exclude degenerate-case tiles. The per-tile weighting is useful in giving tiles different influences on the shape optimization. A typical strategy is to give tiles of flexible shapes smaller weights, e.g., the pentagon and triangle tiles in Figure 2 and the flexible single-quad tiles in Figure 10.

After the vertex positions on M are updated by solving equation 4, we perform shape registrations for all tiles again to update the registered positions of the vertices. The procedure is repeated until the sum of the shape registration errors of all tiles is below a threshold or has reached a local minimum.

Boundary constraints: Additional constraints are added to preserve the fidelity of the mesh boundaries. For each border vertex, b_m, $0 \leq m < B$, where B is the number of border vertices on M, we identify its two adjacent border vertices, b_{m0} and b_{m1}. If b_{m0}, b_m, and b_{m1} are co-linear by a threshold, we constrain b_m to be on the straight line with its slope defined by b_{m0} and b_{m1} and passing through b_m. Otherwise, b_m is constrained at its current position. In this way, border vertices are allowed to move, leading to more degrees of freedom for the shape optimization, without sacrificing the fidelity of the mesh boundaries.

Straightening constraints: Recall that a tiling also defines a boundary network, i.e., the edges belonging to tile boundaries, of which the aesthetics is largely determined by the shape optimization. In our applications, we prefer lines in the boundary network that are already close to linear to be straightened. For this goal, for every vertex on M, v_n, we identify pairs of its adjacent vertices, v_{n0} and v_{n1}, such that v_{n0}, v_n, and v_{n1} are co-linear by a threshold. For every such triple, a quadratic term that measures the deviation of v_n from the average of v_{n0} and v_{n1}, $W_s(v_n - (v_{n0} + v_{n1})/2)^2$, where W_s is the weight for straightening constraints, is added to equation 4.

6 Results and Applications

We use CGAL [cgal 2012] for the half-edge mesh framework and Gurobi [Gurobi Optimization 2014], a specialized (mixed) integer programming solver, for solving the discrete tiling problems. All tests are done on a 2.1GHZ quad core CPU, 8GB RAM machine.

Comparison to local methods for the discrete tiling problem: A key advantage of our approach is to formulate the discrete tiling problem into an integer programming form, which enables us to solve it with specialized integer programming solvers. To demonstrate this advantage, we compare our results with those of a typical stochastic search method that does greedy placement of tiles on an advancing front, starting at a randomly selected face on the mesh border, with increasing priorities for tiles that are locally preferable, i.e., with more adjacent edges to already tiled faces, over time, i.e., simulated annealing. We run this method multiple times until a full tiling is found or a time limit is reached. The results are shown in Figure 7. In summary, by harnessing the power of specialized solvers, our approach is magnitudes faster than naive stochastic methods, enabling us to solve problems that are prohibitively expensive for local methods.

Floorplans: Our approach is suitable for generating floorplan layouts of large facilities, e.g., offices, hospitals, and parking lots.

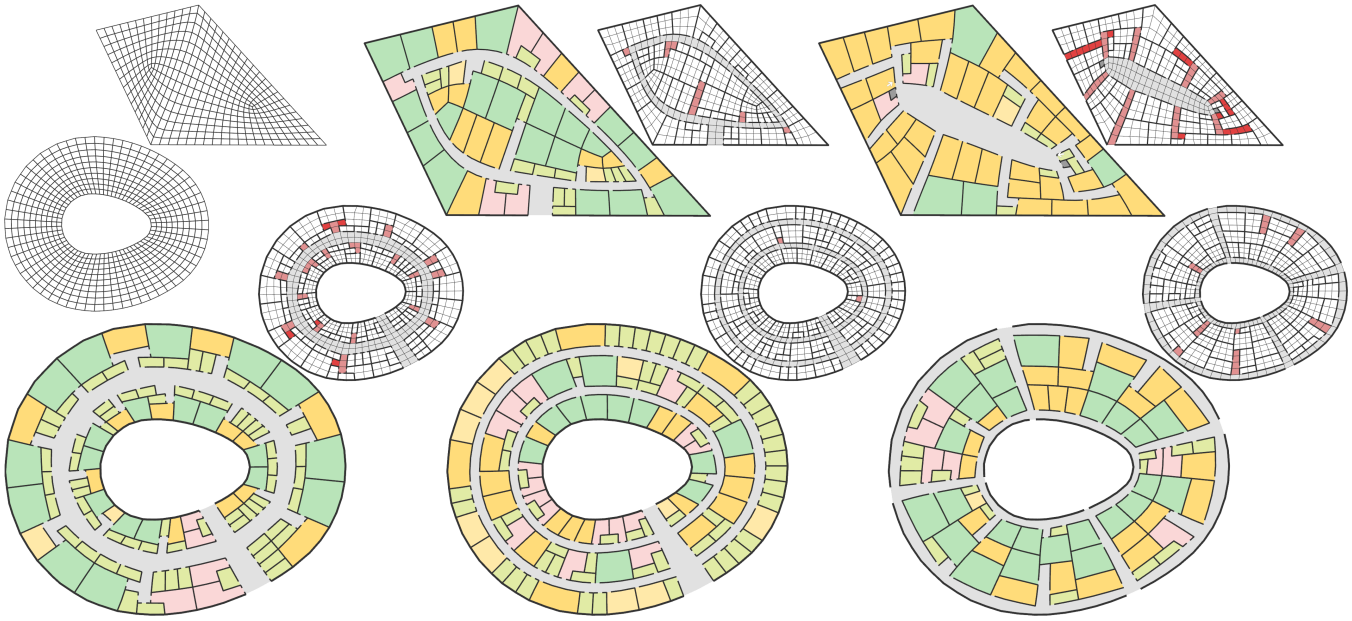

Figure 8: *Top and bottom rows: two floorplan examples using the corridor and room templates defined in Figure 1.*

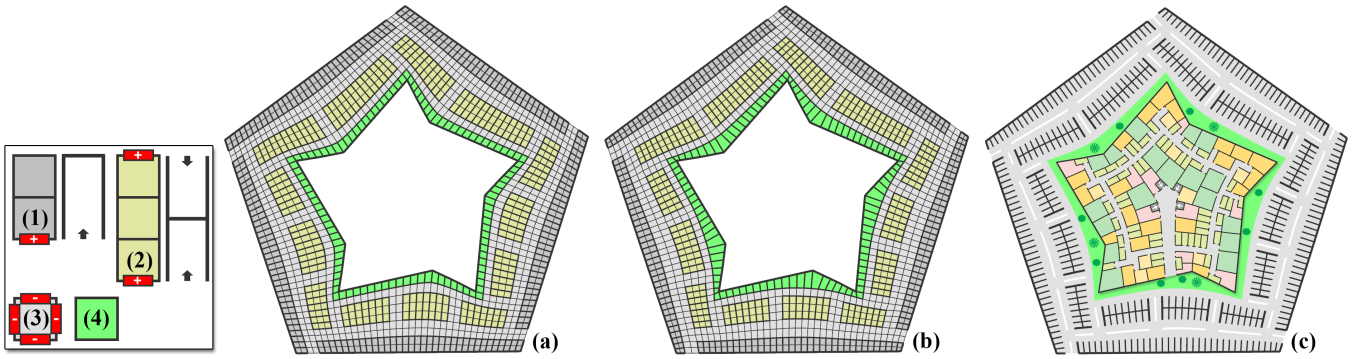

Figure 9: *Parking lot design using a set of templates with edge color constraints to enforce the accessibility criteria: (1) and (2): templates for a full-sized and a twin compact-sized parking lot. Edges that must access roads are given the positive red color. (3) Road tiles of which every edge is given a negative red color for parking lot access. (4) Green tiles with flexible shapes. (a) and (b): The tiling before and after a geometric optimization. Road tiles are specified by the user. We allow only similarity transformations for the parking lot and road tiles. However, the concave spaces that were difficult to use are accommodated by the green tiles with flexible shapes. (c) A stylization of the tiling obtained by manually adding trees and replacing tiles by (warped) 2D vector textures.*

Compared with floorplans for single-house residential buildings, the problem domains are often much larger and have predefined building footprints of arbitrary shapes. The task can be summarized as putting as many copies of a few given room/lot templates as possible into a problem domain in a water-tight manner. Each room/lot template comes with a predefined shape and admissible transformations. Moreover, the layouts need to be functional in terms of accessibility; for example, all rooms/lots should be connected to the exits of the building via a singly connected corridor. Overall, all these criteria amount to a global optimization problem with both discrete and continuous components. Our approach is able to find novel solutions. See Figure 1, 8, and 9 for examples.

Urban pattern layout: In [Yang et al. 2013], urban pattern layouts are generated in two stages. First, the problem domain is partitioned into sub-regions along cross-field streamlines. Second, each sub-region is turned into parcels by template matching. While the resulting patterns are of high geometric quality, there is a lack of control over the connectivity of the street network, i.e., boundaries between sub-regions. In Figure 10 and 11, we show that our tiling-based approach is an important improvement to streamline-based sub-region partitioning, enabling users to control the connectivity of the street networks and the occurrences of templates, and to find non-trivial design solutions with templates of arbitrary, non-rectangular shapes.

Arts and design: Our approach is a powerful solution-finding tool for tiling-based designs, e.g., using the Tetris tiling set (Figure 12). As future work, we would like to explore more design options that also take advantage of the adjacency constraints. One such example is shown in Figure 5 in the additional materials.

Performance: The timing statistics of our results are presented in Table 1. Overall, the Gurobi solver delivers amazing performance. The execution time depends more on the characteristics of the problem, e.g., the number of templates and colors for the soft adjacency constraints, than on the number of quads in the mesh. Hard con-

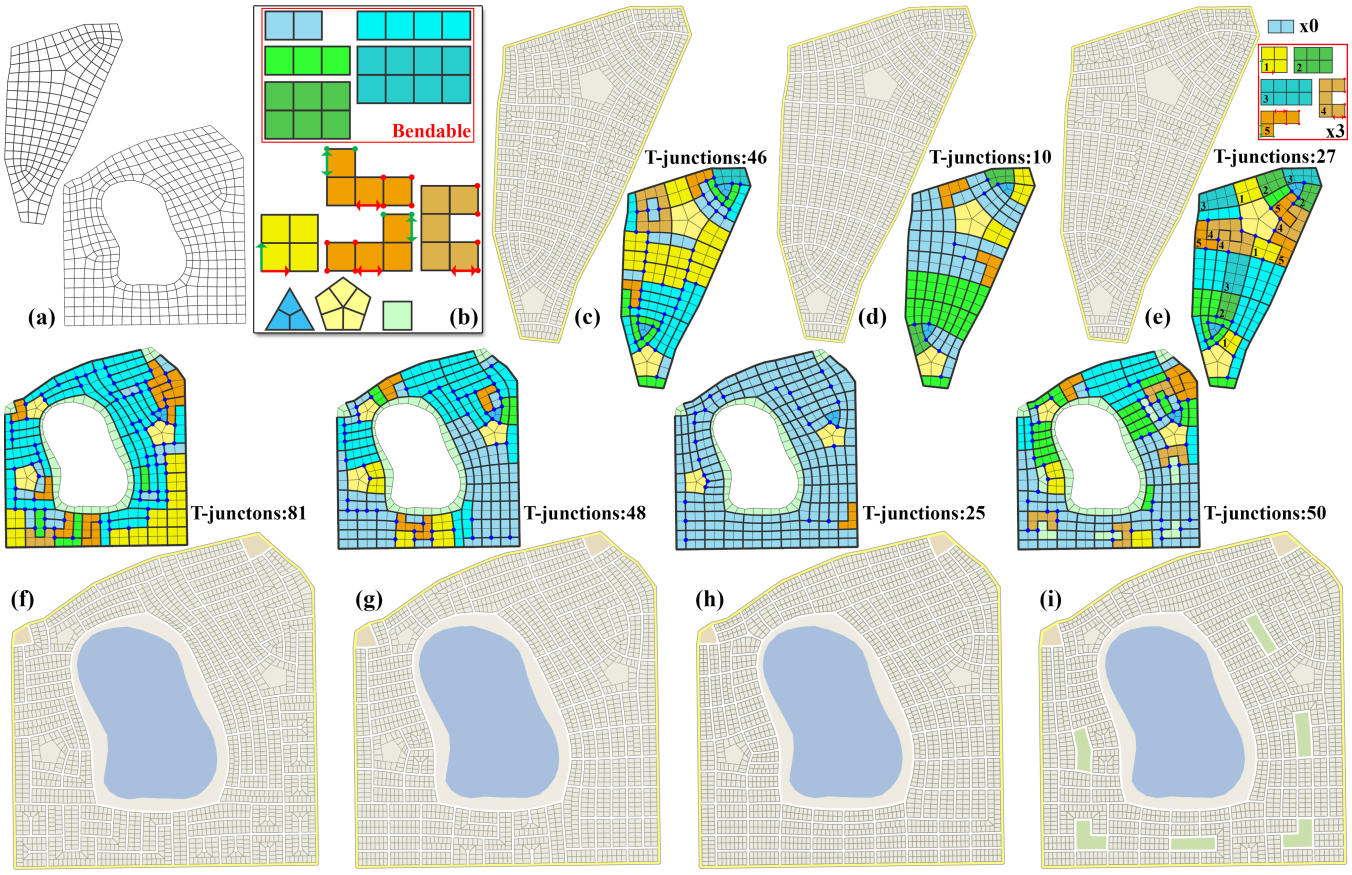

Figure 10: *Urban layout design. (a) The quadrangulations of two problem domains from [Yang et al. 2013]. (b) We create a set of tile templates that includes more general tiles as can be handled by Yang et al.: 1) a set of bendable tiles, 2) L-shaped, J-shaped, and U-shaped tiles of which each end can stretch individually, 3) 2x2 square tiles that can scale non-uniformly, 4) triangle-shaped and pentagon-shaped tiles that can only scale uniformly, and 5) single-quad tiles with lower weights for shape optimization (these tiles are used to model lake shores and parks with flexible shapes). Each tile may take a further similarity registration. (c) to (e): Three different designs for the first problem domain. We can create layouts that favor variety (c) and regularity (d), i.e., fewer numbers of T-junctions, by controlling the weights for the regular-junction constraints. (e) We can precisely control the numbers of occurrences for each tile template. (f) to (i): Four different designs for the second problem domain. (f) to (h): Designs with increasing regularity, at the cost of decreasing variety. The second one (g) is a nice example that strikes a balance. (i) A design with user-specified locations of parks.*

straints, e.g., hard edge color constraints and boundary constraints, have little impact on the performance. Note that sub-optimal solutions are also valid (water-tight and non-overlapping) tilings; therefore, for difficult problems, we resort to nearly-optimal solutions computed under reasonable time limits (Figure 7 in the additional materials). The times to calculate shape factors for all potential tile placements by shape registrations can be significant. Finally, it takes about 20 to 30 seconds (around 40 to 60 iterations) for the geometric optimization in most of our results.

Limitations: Scalability is the main limitation of our approach. In general, the execution time increases rapidly when the problem size, e.g., the numbers of templates, colors for the soft adjacency constraints, and quads in the mesh, becomes large. To partially address this problem, there exist several known heuristics to improve the execution time of solving tiling problems ([Prokopyev and Karademir 2012]). For example, we can partition the domain into subdomains and solve them separately. See Figure 6 in the additional materials for an analysis.

Another limitation is the dependence on the initial quadrangulations. Indeed, the solution space of possible layouts is limited by how the domain is quadrangulated in the first place. There exist several approaches to explore quadrangulations of a given domain in a combinatorial (e.g., [Peng et al. 2014]) or field-based (e.g., [Liu et al. 2011]) sense, effectively enabling users to explore layout results of different quadrangulations. See Figure 13 for examples. On the positive side, by starting with an initial quadrangulation such that the edge lengths and the aspect ratios of the quads roughly match the templates, many "bad" solutions, e.g., of which the orientations of tiles and the distributions of templates with inner irregular vertices are incompatible with the problem domain, are filtered out.

7 Conclusion and Future Work

In conclusion, our approach takes a novel tiling-based approach to tackle the problem of generating water-tight layouts with deformable templates. By formulating the tiling problem as a global optimization problem solved by a specialized solver, we are able to find non-trivial combinatorial solutions that would be prohibitively expensive for local search-based methods. Furthermore, by including the shape errors of potential tile placements into the optimization formulation, tilings with overall better shapes can be computed.

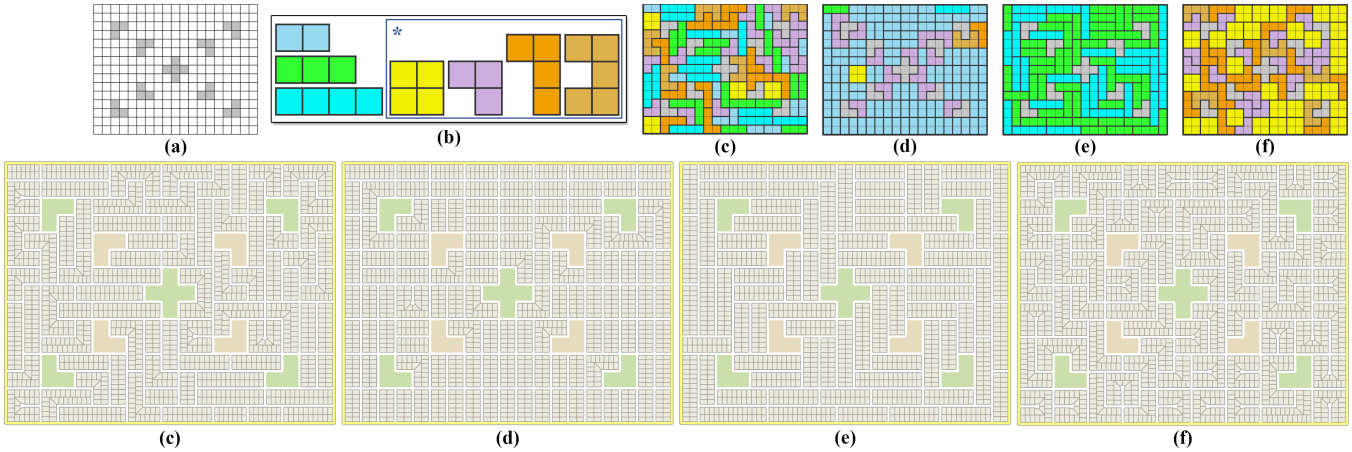

Figure 11: *Urban layout design in a regular grid setting. (a) The quadrangulated problem domain. User-specified facility areas are marked in grey. (b) Tile templates. (c) A stochastic-looking design generated using zero weight for the regular-junction constraints and a small randomization factor to perturb the tile placements. (d) A regular-looking design generated using a large weight for the regular-junction constraints; furthermore, each template is constrained to appear at least once (to avoid overly monotonic-looking solutions). (e) A design using templates of longer (3x1 and 4x1) strips only. (f) A design using templates in the marked sub-group only.*

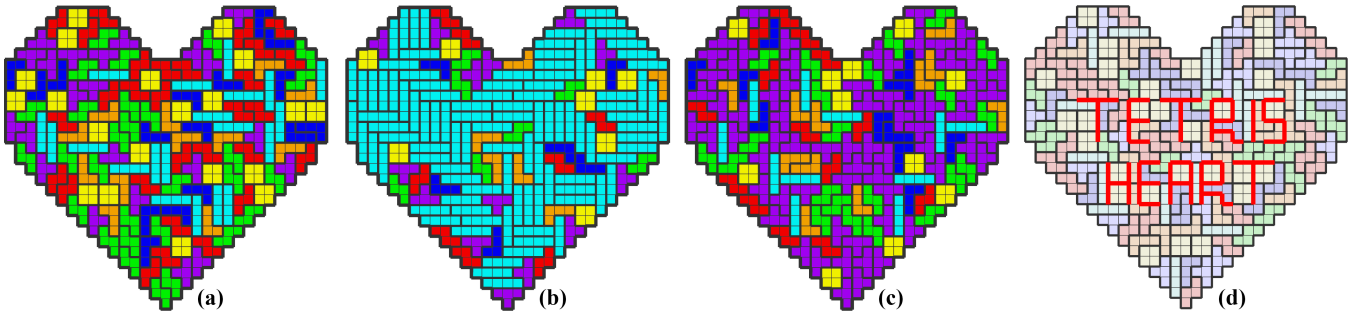

Figure 12: *Tetris tiling design with a stochastic distribution by a large randomization factor (a), higher priorities for the 4x1 tiles (b), higher priorities for the T-shaped tiles (c), and with engraved characters imposed by boundary constraints (d).*

Lastly, the geometry can be further improved by a continuous optimization that improves the shape fidelity of the tiles and the aesthetic of the boundary network.

As future work, we would like to explore other choices of problem domain tessellations, for examples, quad-dominant meshes and triangles meshes. This leads to more flexibility and interesting applications. One promising venue for applications is mesh processing, e.g., quadrangulation of triangle meshes (by placing quad-shaped tiles on triangle meshes) and requadrangulations (by placing quads of different shapes on quad meshes). Another interesting direction is to explore layouts on domains other than 2D polygons. In fact, our discrete tiling algorithm is immediately applicable for general surface meshes, e.g., a bunny (a sphere-like object) and a torus (see Figure 14). Beyond tiling on surface meshes, we can tile volumetric tiles in volumetric meshes. This may be useful for 3D spatial layout designs such as floorplans in multi-story buildings.

Appendix A: Transformation Step Registration

Recall that a transformation step applies to a subset of the boundary vertices of the template's base polygon. We denote this subset of vertices in the base polygon and the corresponding tile on M as v_i and V_i (v_i corresponds to V_i), $0 \leq i < n$, where n is the number of vertices in the subset. A similarity transformation is registered by solving a least-squares system of the distances between corresponding vertices: $V_i \mapsto Av_i + T$, $0 \leq i < n$, where A is a 2x2 matrix, $A_{11} = A_{22}$, $A_{12} = -A_{21}$, and T is a translation vector. A rigid transformation is registered similarity, with the additional constraint $A_{11}^2 + A_{22}^2 = 1$. A translation is registered trivially. A rotation along a center is registered by solving a similarity transformation without translation and extracting the rotation angle as $arcos(A_{11}/\sqrt{A_{11}^2 + A_{12}^2})$, using positions of v_i and V_i translated by the negative of the center. A scaling along a direction and a center is registered by solving a least-squares system: $V_i \mapsto Av_i$, $0 \leq i < n$, where A is a 2x2 matrix, A_{11} is the scaling factor, $A_{12} = A_{21} = 0$, and $A_{2,2} = 1$, i.e., a scaling along the x-axis, using positions of v_i and V_i translated by the negative of the center and then rotated by the negative of the angle between the direction and the x-axis. A shearing is registered similarly. The only difference is that matrix A now resembles a shearing along the x-axis, i.e., $A_{11} = A_{22} = 1$, A_{12} is the shearing factor, and $A_{21} = 0$.

Acknowledgments

We thank the anonymous reviewers for their insightful comments, Yoshihiro Kobayashi and Christopher Grasso for the renderings, and Virginia Unkefer for the proofreading. This research is supported by the NSF #0643822 and the Visual Computing Center at King Abdullah University of Science and Technology.

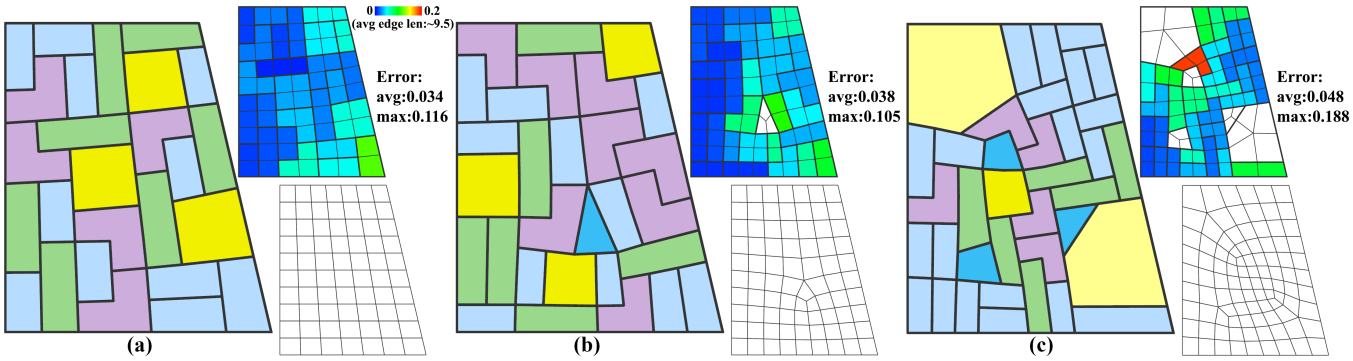

Figure 13: *(a) to (c): Layout results of alterntative initial quadrangulations of the same problem domain and tile templates in Figure 2. The initial quadrangulations are shown in the bottom-right corners and the shape registration errors are shown in the upper-right corners. (a) A regular tiles-only layout based on a fully regular quadrangulation. However, the triangle and pentagon tiles are excluded and now there is a slight deviation of tile areas between the top and bottom of the domain. (b) A layout based on a quadrangulation that can only be tiled with at most one triangle or pentagon tile (because the two irregular vertices are too close). Note that there is no pentagon tile around the valence-5 vertex. (c) A layout based on a quadrangulation with more irregular vertices.*

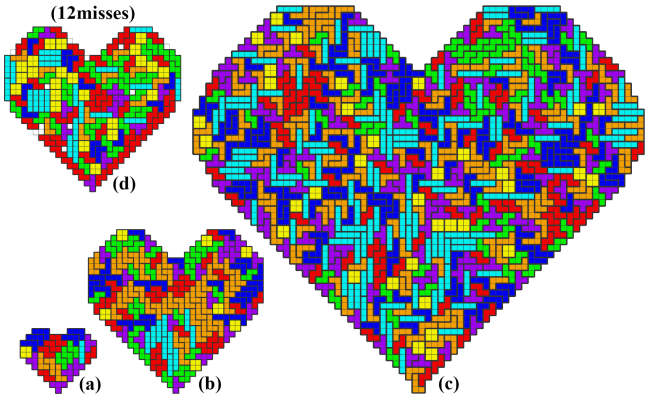

Figure 7: *Performance comparison to local methods. We test the Tetris tiling on three domains with increasing sizes: 92 faces (a), 592 faces (b), and 2628 faces (c). Tilings shown in (a), (b), and (c) are generated by our approach without randomization factors. For (a), it takes 0.14 seconds for our solver to find a full tiling. For the greedy placement method, it takes 35.55 seconds and 6.60 seconds without and with simulated annealing. For (b), it takes 0.54 seconds for our solver to find a full tiling. For the greedy method with simulated annealing, the best result we obtained within a 1000 second time limit is shown in (d), which still has 12 missing faces. For (c), it takes 71.46 seconds for our solver to find a full tiling. The best result by the greedy placement methods within a 1000 second time limit has 72 missing faces.*

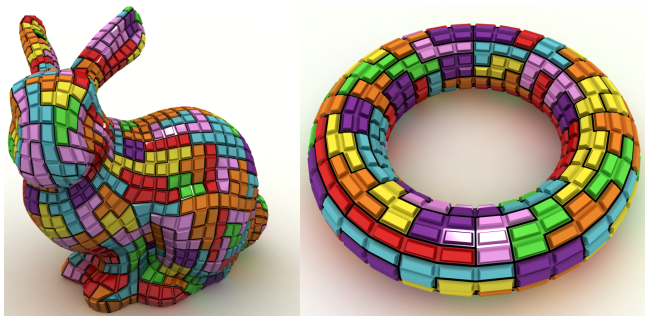

Figure 14: *Tetris tiling designs on general surface meshes.*

References

ALIAGA, D., VANEGAS, C., AND BENES, B. 2008. Interactive Example-Based Urban Layout Synthesis. *ACM Trans. Graph. 27*, 5.

ALIAGA, D. G., BENEŠ, B., VANEGAS, C. A., AND ANDRYSCO, N. 2008. Interactive Reconfiguration of Urban Layouts. *IEEE Computer Graphics and Applications 28*, 3, 38–47.

BAO, F., SCHWARZ, M., AND WONKA, P. 2013. Procedural facade variations from a single layout. *ACM Trans. Graph. 32*, 1, 8.

BLACKER, T. D., AND STEPHENSON, M. B. 1991. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering 32*, 4, 811–847.

BOMMES, D., LVY, B., PIETRONI, N., PUPPO, E., A, C. S., TARINI, M., AND ZORIN, D. 2012. State of the art in quad meshing. In *Eurographics STARS*.

BOUAZIZ, S., DEUSS, M., SCHWARTZBURG, Y., WEISE, T., AND PAULY, M. 2012. Shape-up: Shaping discrete geometry with projections. *Comp. Graph. Forum 31*, 5 (Aug.), 1657–1667.

CGAL, 2012. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Trans. Graph. 22*, 3, 287–294.

DAI, D., RIEMENSCHNEIDER, H., SCHMITT, G., AND VAN GOOL, L. 2013. Example-based facade texture synthesis.

DEMAINE, E. D., AND DEMAINE, M. L. 2007. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graph. Comb. 23*, 1 (Feb.), 195–208.

FASANO, G. 2004. A mip approach for some practical packing problems: Balancing constraints and tetris-like items. *Quarterly*

Properties: tiling	faces/ edges	tiles	tile/ joint vars	shape/ rand. weights	Timing: (seconds) shape reg.	first sol.⋆	shown sol.
Fig 1,1	605/1265	241	31466/0	0.01/0	59.40	46	100*(0.03%)
Fig 1,2	605/1265	211	31358/0	0.01/0	76.78	209	400*(0.09%)
Fig 1,3	605/1265	270	31698/0	0.01/0	76.71	27	100*(0.01%)
Fig 2,1	71/159	28	505/0	0.5/0.05	1.55	1	0.09
Fig 2,2	71/159	28	505/53	0.5/0.05	1.67	1	0.12
Fig 5,1	173/373	47	465/0	0/0.01	N/A	1	0.03
Fig 5,2	173/373	47	465/145	0/0.01	N/A	1	0.06
Fig 8,t1	397/828	131	20740/0	0.01/0	49.62	31	31.78
Fig 8,t2	397/828	170	20746/0	0.01/0	64.68	68	200*(0.05%)
Fig 8,b1	620/1302	293	31818/0	0.01/0	80.06	70	200*(0.02%)
Fig 8,b2	620/1302	265	44268/0	0.01/0	81.80	8	9.56
Fig 8,b3	620/1302	254	31872/0	0.01/0	86.34	41	200*(0.01%)
Fig 9	1341/2831	941	24987/0	0/0	N/A	2	2.05
Fig 10,c	169/368	48	2405/0	0.5/0	8.25	1	1.13
Fig 10,d	169/368	66	2405/136	0.5/0	8.10	1	4.03
Fig 10,e	169/368	39	2405/136	0.5/0	8.23	2	7.17
Fig 10,f	332/721	123	4073/0	0.5/0	12.58	1	0.36
Fig 10,g	332/721	147	4073/266	0.5/0	12.55	1	2.01
Fig 10,h	332/721	185	4073/266	0.5/0	12.41	1	1.69
Fig 10,i	332/721	164	4073/266	0.5/0	12.99	1	2.11
Fig 11,c	285/604	106	4770/0	0/0.05	N/A	1	0.25
Fig 11,d	285/604	145	4770/252	0/0.05	N/A	1	2.49
Fig 11,e	285/604	105	4770/0	0/0.05	N/A	1	0.05
Fig 11,f	285/604	96	4770/0	0/0.05	N/A	1	0.38
Fig 7,1	92/209	23	1196/0	0/0	N/A	1	0.14
Fig 7,2	592/1249	100	9770/0	0/0	N/A	1	0.54
Fig 7,3	2628/5396	657	46745/0	0/0	N/A	71	71.46
Fig 12,a	592/1249	148	9770/0	0/0.2	N/A	1	10*(0.45%)
Fig 12,b	592/1249	148	9770/0	0/0.2	N/A	1	10*(0.64%)
Fig 12,c	592/1249	148	9770/0	0/0.2	N/A	1	10*(1.55%)
Fig 12,d	592/1249	148	9770/0	0/0.2	N/A	1	10*(1.9%)
Fig 14,1	1446/2892	360	25031/0	0/0	N/A	N/A†	1000*(0.42%)
Fig 14,2	512/1024	128	9728/0	0/0	N/A	0	10*(0.59%)

⋆ Gurobi report times to achieve the first feasible solutions in whole seconds.

† The bunny has 1446 faces thus a full Tetris tiling is impossible.

Table 1: *Timing statistics. We show the number of faces in the mesh, numbers of tiles, numbers of Boolean variables for the potential tile placements and (vertex-based) joint constraints, the weights for shape and randomization factors (in a 0 to 1 scale), times (seconds) for calculating shape errors for all potential tile placements by shape registrations, times to find the first feasible solution, i.e., a full tiling, and times to obtain the shown solutions. Solutions that were computed sub-optimally by a time limit are marked and the closeness to the optimal solution is provided.*

Journal of the Belgian, French and Italian Operations Research Societies 2, 2, 161–174.

GUROBI OPTIMIZATION, I., 2014. Gurobi optimizer reference manual.

HAUSNER, A. 2001. Simulating decorative mosaics. In *Proceedings of SIGGRAPH 2000*, 573–580.

HUANG, H., ZHANG, L., AND ZHANG, H.-C. 2011. Arcimboldo-like collage using internet images. *ACM Trans. Graph. 30*, 6 (Dec.), 155:1–155:8.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. 31*, 4, 55.

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

KAPLAN, C. S. 2009. *Introductory Tiling Theory for Computer Graphics*. Morgan-Claypool Publishers.

KIM, J., AND PELLACINI, F. 2002. Jigsaw image mosaics. *ACM Trans. Graph. 21*, 3 (July), 657–664.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph.*, 509–518.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3, 362–371.

LIN, J., COHEN-OR, D., ZHANG, H., LIANG, C., SHARF, A., DEUSSEN, O., AND CHEN, B. 2011. Structure-preserving retargeting of irregular 3D architecture. *ACM Trans. Graph. 30*, 6, 183:1–183:10.

LIU, Y., XU, W., WANG, J., ZHU, L., GUO, B., CHEN, F., AND WANG, G. 2011. General planar quadrilateral mesh design using conjugate direction field. *ACM Trans. Graph. 30*, 6 (Dec.), 140:1–140:10.

MAJEROWICZ, L., SHAMIR, A., SHEFFER, A., AND HOOS, H. H. 2014. Filling your shelves: Synthesizing diverse style-preserving artifact arrangements. *IEEE Transactions on Visualization and Computer Graphics*.

MERRELL, P., SCHKUFZA, E., AND KOLTUN, V. 2010. Computer-generated residential building layouts. *ACM Trans. Graph. 29*, 6 (Dec.), 181:1–181:12.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph. 30*, 4 (July), 87:1–87:10.

PARK, C., NOH, J.-S., JANG, I.-S., AND KANG, J. M. 2007. A new automated scheme of quadrilateral mesh generation for randomly distributed line constraints. *Computer Aided Design 39*, 4 (Apr.), 258–267.

PENG, C.-H., BARTON, M., JIANG, C., AND WONKA, P. 2014. Exploring quadrangulations. *ACM Trans. Graph. 33*, 1 (Feb.), 12:1–12:13.

PROKOPYEV, O., AND KARADEMIR, S. 2012. Irregular polyomino tiling via integer programming with application in phased array antenna design.

REINERT, B., RITSCHEL, T., AND SEIDEL, H.-P. 2013. Interactive by-example design of artistic packing layouts. *ACM Trans. Graph. 32*, 6 (Nov.), 218:1–218:7.

VANEGAS, C. A., KELLY, T., WEBER, B., HALATSCH, J., ALIAGA, D., AND MÜLLER, P. 2012. Procedural generation of parcels in urban modeling. *Comput. Graph. Forum 31*, 2.

YANG, Y.-L., WANG, J., VOUGA, E., AND WONKA, P. 2013. Urban pattern: Layout design by hierarchical domain splitting. *ACM Trans. Graph. 32*, 6 (Nov.), 181:1–181:12.

YEH, Y.-T., YANG, L., WATSON, M., GOODMAN, N. D., AND HANRAHAN, P. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Trans. Graph. 31*, 4.

YEH, Y.-T., BREEDEN, K., YANG, L., FISHER, M., AND HANRAHAN, P. 2013. Synthesis of tiled patterns using factor graphs. *ACM Trans. Graph. 32*, 1 (Feb.), 3:1–3:13.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: Automatic optimization of furniture arrangement. *ACM Trans. Graph. 30*, 4 (July), 86:1–86:12.