Surface Mosaic Synthesis with Irregular Tiles

Wenchao Hu, Zhonggui Chen, Hao Pan, Yizhou Yu, Eitan Grinspun, and Wenping Wang, *Member, IEEE*

Abstract—Mosaics are widely used for surface decoration to produce appealing visual effects. We present a method for synthesizing digital surface mosaics with irregularly shaped tiles, which are a type of tiles often used for mosaics design. Our method employs both continuous optimization and combinatorial optimization to improve tile arrangement. In the continuous optimization step, we iteratively partition the base surface into approximate Voronoi regions of the tiles and optimize the positions and orientations of the tiles to achieve a tight fit. Combination optimization performs tile permutation and replacement to further increase surface coverage and diversify tile selection. The alternative applications of these two optimization steps lead to rich combination of tiles and high surface coverage. We demonstrate the effectiveness of our solution with extensive experiments and comparisons.

Index Terms—Simulated mosaics, irregular packing, polygon containment, surface tessellation

1 Introduction

Mosaics represent a popular method for decorating surfaces and can often be seen in churches, parks, or on sidewalks. Because of their appealing visual effects, mosaic patterns are regarded as a popular form of folk art. Since tiles, called *tesserae*, used in mosaics are often broken pieces of crockery, ceramics, or glass, they may come with a wide range of shapes and colors. There are many different ways to make mosaic patterns, such as *Opus Regulatum, Opus Tessellatum, Opus Vermiculatum, and Opus Palladianum* [1]. We are particularly interested in *Opus Palladianum*, also known as "crazy paving", which makes use of irregularly shaped tiles to form a pattern. Manually generating aesthetic mosaic patterns in *Opus Palladianum*, especially in large scale, is a demanding task. A rule of thumb used by artists is that adjacent tiles should have complementary shapes which fit each other.

We shall propose an effective method for synthesizing mosaics with irregularly shaped tiles, with the aim of decorating virtual objects and scenes for special visual effects in the entertainment industry. There are two important goals. First, the developed technique should be able to work well with any set of input tiles. Second, the synthetically generated mosaics should be visually pleasant, meanwhile the synthetic approach should be more efficient than manual work when similar results are obtained. Irregularly shaped tiles, especially concave ones, make it harder to achieve these goals. Given a collection of input tiles, a mosaic synthesis algorithm needs to determine a subset of tiles that not only fit each other well in shape but are also diverse enough to avoid undesirable repetitive layout of similar tiles. Furthermore, an aesthetic mosaic pattern should have a tight coverage of the base surface, without any overlap among the tiles.

Our solution is optimization-based. It alternates between continuous tile configuration optimization and combinatorial tile selection optimization. Here, the *title configuration* refers to the position and orientation of a tile in a mosaic. More specifically, the configurations of individual tiles are optimized continuously, and the mosaic tiles are selected with combinatorial optimization. Because a large number of variables are used to represent the configurations of the tiles, we need to solve an optimization problem with a large number of unknowns. To make the problem tractable, we decouple interactions among different tiles by partitioning the base surface into approximate Voronoi regions of the tiles. This partition is dynamically computed while the tile configurations are updated. With such a partition, increasing surface coverage by the tiles becomes the problem of increasing the coverage of individual regions by their corresponding tiles. Finally, combinatorial tile selection optimization is used to further reduce the uncovered area of the surface by tile permutation and tile replacements so that the new tiles better fit their neighboring tiles in shape to reduce the gaps between neighboring tiles.

2 Related Work

2.1 Irregular Packing

The problem of packing irregularly shaped objects into a given region or volume (called a *container*) is widely studied in industrial applications, including garment manufacturing, furniture making, and metal sheet cutting. The problem is to find the tightest non-overlapping spatial arrangement of a set of irregularly shaped objects in the container. The packing problem is NP-hard [2]. Hence, heuristics are usually used to obtain suboptimal solutions with reasonable time complexity. A comprehensive review of the work on 2D irregular packing can be found in [3], [4]. Although many effective methods have been proposed for 2D or 3D packing of objects of regular shapes, such as rectangles, the

- *W. Hu, H. Pan, Y. Yu, W. Wang are with the Department of Computer Science, The University of Hong Kong, Hong Kong. E-mail: {wchu, hpan, yzyu, wenping}@cs.hku.hk.*
- *Z. Chen is with the Department of Computer Science, Xiamen University, Xiamen 361005, China. E-mail: chenzhonggui@xmu.edu.cn.*
- *E. Grinspun is with the Department of Computer Science, Columbia University, New York, NY 10027. E-mail: eitan@cs.columbia.edu.*

Manuscript received 15 Jan. 2015; revised 15 Sept. 2015; accepted 28 Sept. 2015. Date of publication 5 Nov. 2015; date of current version 3 Feb. 2016. Recommended for acceptance by A. Tal.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2015.2498620

geometric complexity of irregular shapes makes the packing problem more challenging. To our knowledge, there has been no prior work on the packing problem on curved surfaces in the field of operational research.

2.2 Polygon Containment

Polygon containment, which is a variant of the packing problem, is to determine whether and how one or more nonoverlapping polygons can fit within a given region. Single polygon containment with translation only has been investigated by Baker et al [5], and rotations are considered in [6], [7], [8], [9]. The cases of two and three polygons are studied in [10], [11]. Translational and rotational polygon containment problems with even more multiple polygons is investigated by Milenkovic [12], [13]. Grinde and Cavalier [8] use a mathematical programming formulation/model to solve the following containment problem: can a (not necessarily convex) polygon P be translated and/or rotated such that P fits within a *convex* polygon Q?

2.3 Computer Simulated Mosaics

The problem of synthesizing 2D mosaics has extensively been studied. A relatively recent survey of mosaics on images and planar surfaces can be found in [14] and [15]. Much research has focused on adding mosaic-like effects to a source image. Here we only focus on the method categorized as ancient mosaics by [14] which simulates mosaics by arranging tile images in a container image. Hausner [16] creates mosaics with rectangular tiles using centroidal Voronoi diagrams based on the Manhattan distance. Liu et al. [17] improves Hausner's method by proposing a fully automatic approach that optimizes an objective function accounting for the desired mosaic properties. Kim and Pellacini [18] pack image tiles into an image container by matching the color of the image tiles to colors in the image container. Tight packing has been achieved by energy minimization and permitting tiles to undergo small deformation. Battiato et al. [19] combine gradient vector field in [20] with tile cutting strategies. Compared with using only rectangular tiles, the results can better capture the features of the underlying images and reach higher area coverage. At the same time, it provides tunable parameters to control the complexity of the resulting tiles. Please note that since tile cutting changes tile shapes, it is not applicable to our problem here. Blasi and Gallo [21] propose a method which exploits extracted directional guidelines and tile cutting, for simulating the workflow of mosaic artists. In this method, the positioning and orientation of tiles can capture image features and the tile shapes are kept close to those seen in real-life mosaic. And as an extension to [21], the work in [22] focuses on the generation of classical Opus Vermiculatum, which treats mosaic tiles in the foreground and background differently and adds snaking tiles around the foreground, in order to better highlight the content in the foreground. In [23], [24], [25], Lloyd-type relaxation is exploited to evenly distribute tiles within a planar region. When tiles have irregular shapes, these methods become less effective, and often leave large gaps between the tiles. Reinert et al. [26] propose a method for arranging given shapes into an artistic layout by inferring from user's interactions.

There is relatively little work on creating mosaics on surfaces in 3D. The work presented in [27] considers placing square tiles with equal size on a surface. Dos Passos and Walter [28] extends Lai et al.'s technique to surface mosaics with nonuniform rectangular tiles. In [29], they simulate mosaics over a sculpture surface using the Voronoi tiles of a set of seed points on the surface. As a consequence, the tiles in their results always have the recognizable convex shapes of Voronoi cells. In contrast, we tackle a more challenging and practical problem of creating surface mosaics using a set of irregularly shaped tiles that the user provides.

2.4 Reassembly of Fragments

Automatic reassembly of broken objects from a large collection of irregular fragments is of great interest in archaeology. Computer-aided technologies make it possible to digitize detailed geometry and texture of each fragment and reassemble digitized fragments using a computer algorithm. A common approach to this problem, based on automated cluster agglomeration, first identifies pairwise matches among the fragments according to shape [30], [31], texture and normal [32] cues, and then assembles them into larger clusters through a connectivity graph. One important assumption made in these reassembly solutions is that the fragments are broken pieces of the same object so that one may expect to find perfect matches from the fragments. This assumption is not applicable in our setting, since the tiles used in a mosaic can have arbitrary shapes and are not necessarily the pieces of the same original object.

3 Problem Description

The input to our mosaic synthesis problem consists of a base surface $\mathcal{M}$, which is usually represented as a triangle mesh surface, and a set of irregularly shaped meta objects represented as planar polygons, denoted $L = \{S_i\}_{i=1}^{m}$. The goal of optimizing tile selection and configuration is to have sufficiently large coverage of the base surface. Every tile lies on a 2D plane that is tangent to the base surface at the centroid $\mathbf{c_i}$ of P_i. Our task is to seek an optimal arrangement of tiles, $\{(P_j, \Theta_j)\}_{j=1}^{n}$, where the tile P_j is a duplicate of an object in $L = \{S_i\}_{i=1}^{m}$, and Θ_j denotes the set of the parameters describing the configuration of P_j, i.e., its position coordinates and orientation angle on the base surface $\mathcal{M}$. The tile arrangement needs to meet the constraint that the tiles do not overlap with each other. Note that the number of tiles to be used in a mosaic, denoted n, is not predefined, and is itself a variable. Surface coverage is defined as the ratio of the sum of the areas of the chosen tiles to the total area of the base surface.

In our problem formulation, we allow each tile to be a scaled version of the meta object that it is associated with, and the scaling factor of each tile is a continuous variable to be optimized. This flexibility in scaling is used to accommodate the fact that the tile size should change according to the local curvature variation of the base surface to provide a tight fit on it; specifically, smaller tiles should be placed at regions of high surface curvature. We note that this variation in tile size does not compromise much the visual effect of the mosaic, which is mainly determined by the shape and layout pattern of the tiles. Furthermore, these scaling factors are used in our algorithm for growing a set of small "seed

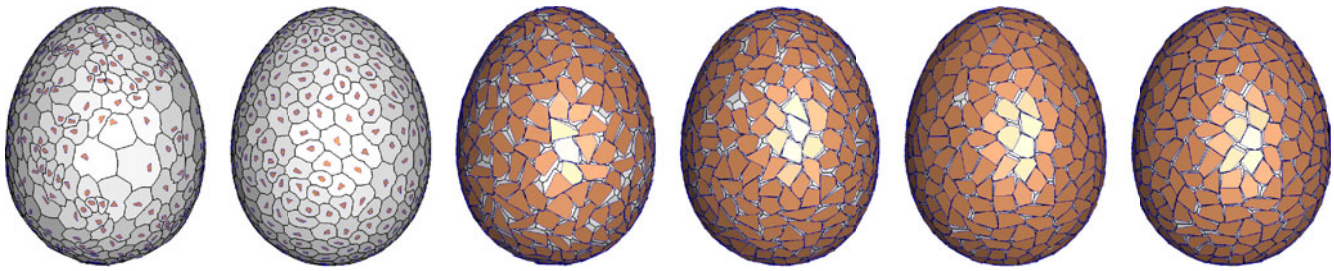


Fig. 1. Algorithm overview. (a) Tile initialization and associated domain partition; (b) Relaxation result before reaching the first scaling factor barrier; (c) Relaxation result by Algorithm 2; (d) Result by tile permutation (Algorithm 3.1) and relaxation; (e) Result by tile replacement (Algorithm 3.2) and relaxation; (f) Result by hole filling (Algorithm 3.3).

tiles" into a mosaic of tiles. Note that allowing the final scale factors not to strictly be 1.0 makes our problem formulation different from a packing problem requiring that the tile size remain the same as specified in the input.

4 Overview of Method

Our hybrid optimization scheme for the mosaicing problem defined in the previous section consists of continuous configuration optimization and discrete combinatorial optimization. The pseudo-code of the overall algorithm is given in Algorithm 1. The intermediate results of the algorithm are shown in Fig. 1.

Algorithm 1. Synthesizing Mosaic by Hybrid Optimization

Input: $\mathcal{M}$, a supporting surface
α, a small threshold for determining if the increase of coverage rate stops
$L = \{S_1, S_2, \ldots, S_m\}$, a set of meta objects
Output: A compact layout of tiles over $\mathcal{M}$
1: Generate initial tiles by randomly sampling L with repetition
2: Shrink the tiles and distribute them over $\mathcal{M}$ without any overlap
3: Perform tile configuration optimization (Algorithm 2)
4: **repeat**
5: Swap the tiles (Algorithm 3.1)
6: Shrink the tiles and optimize (Algorithm 2)
7: **until** the increase of surface coverage $< \alpha$
8: **repeat**
9: Replace the tiles (Algorithm 3.2)
10: Shrink the tiles and optimize (Algorithm 2)
11: **until** the increase of surface coverage $< \alpha$
12: Fill the holes (Algorithm 3.3)

The continuous configuration optimization is based on an iterative relaxation scheme, which iteratively adjusts tile configuration, including position, orientation and scaling, with the goal of increasing surface coverage. It starts with an initial arrangement of tiles $\{P_i\}_{i=1}^n$ of sufficiently small size over the surface $\mathcal{M}$ (Section 5.1). As shown in Algorithm 2, in each iteration, the base surface is partitioned into a set of nonoverlapping regions using the tiles as the centers of the regions (Section 5.2). To increase surface coverage, the configuration of every tile is adjusted using constrained nonlinear optimization (Section 5.3) so that the tile can cover more area within its own region. More details about the algorithm will be explained in Section 5.3.

Algorithm 2. Tile Configuration Optimization

Input: $\mathcal{M}$, a supporting surface
$\{P_1, P_2, \ldots, P_n\}$, an initial layout of tiles over $\mathcal{M}$
$\{b_1, b_2, \ldots, b_l | b_j < b_{j+1}\}$, scaling factor barriers
itn_{max}, the maximum iteration number for each phase
Output: A compact layout of tiles over $\mathcal{M}$
1: **for** each scaling barrier b_j **do**
2: $itn_{curr} \leftarrow 0$
3: **while** $itn_{curr} < itn_{max}$ **do**
4: Construct an approximate chordal axis transform and project the resulting regions to the planes of $\{P_i\}_{i=1}^n$
5: **for** each tile P_i **do**
6: $r_i \leftarrow$ scaling factor between P_i and its corresponding meta object
7: Find the largest copy of P_i (with a scaling factor s_i, a rotation angle θ_i, and a translation $\mathbf{t_i}$) that lies completely inside V_i
8: **if** $s_i r_i < b_j$ **then**
9: Apply the transformation $T(s_i, \theta_i, \mathbf{t_i})$ to P_i
10: **else**
11: Apply the transformation $T(\frac{b_j}{r_i}, \theta_i, \mathbf{t_i})$ to P_i
12: **end if**
13: **end for**
14: Project the centroids of the updated tiles to the supporting surface, and align the tiles with the tangent planes
15: $itn_{curr} \leftarrow itn_{curr} + 1$
16: **end while**
17: **end for**

When the tiles grow during tile optimization, some tiles may reach their prescribed sizes earlier than the others, before their attaining optimal position and orientation. That may hinder other tiles from reaching their optimal configuration. To avoid such suboptimal scenarios, we control the pace of scaling factor optimization with a multi-phase synchronization strategy. That is, all the tiles are expected to almost simultaneously reach their prescribed sizes. By adopting this synchronization scheme, tile sizes can be optimized in a more coordinated manner and better tile configurations can be reached globally.

Starting from a random initial placement of tiles, configuration optimization alone usually cannot produce satisfactory results. Therefore, we further apply combinatorial

optimization for tile permutation, tile replacement and hole filling, as illustrated in Algorithm 3. For every tile in an existing layout, we temporarily remove it and extract the boundary of the vacancy formed by its surrounding tiles. The tile permutation step checks whether there exists a subset of tiles such that a permutation of the tiles in this subset will yield a tighter fit inside the tiles' regions than their current occupants. If yes, the performed tile permutation will be accepted; otherwise, the permutation is not accepted and we back track to the configuration before the permutation. See details in Section 6.1. Since there may exist a number of different meta objects that could be used for every unoccupied region, the replacement step tries to replace a tile with a meta object that better fills the region (Section 6.2). Finally, we detect areas on the supporting surface that have not been covered by any tiles, and fill them with additional tiles to obtain the final mosaic pattern.

Algorithm 3. Tile Combination Optimization

Input: $\mathcal{M}$, a supporting surface
$L = \{S_1, S_2, \ldots, S_m\}$, a set of meta objects
$\{P_1, P_2, \ldots, P_n\}$, an initial layout of tiles over $\mathcal{M}$
Output: A layout of tiles with improved shape compatibility

Algorithm 3.1 Tile Permutation
1: choose a subset of the tiles in the current layout
2: compute the best permutation of the tiles in the subset

Algorithm 3.2 Tile Replacement
3: **for** the tile surrounded by large uncovered region **do**
4: Remove it from the layout
5: Extract the hole boundary formed by surrounding tiles
6: Fill the hole with the best matching meta object
7: **end for**

Algorithm 3.3 Hole Filling
8: Detect the vacant space (holes)
9: Fill the holes with the best matching meta objects

We perform tile configuration optimization and tile combination optimization alternately in multiple passes. These operations are complementary to each other and together they lead to an optimized choice of tiles and their configuration. After each pass of tile combination optimization, as a relaxation technique, we reduce the size of all tiles by a certain percentage before optimizing their configuration again. Surface coverage is usually improved significantly after a few rounds of such combinatorial tile optimization and relaxation.

In the following, we discuss the details of our algorithm.

5 Tile Configuration Optimization

5.1 Initial Tile Placement

The initial number of tiles, denoted n, is determined by dividing the total area of the supporting surface by the average area of the meta objects. After determining the number, the tiles are initially distributed over input surface. This distribution is achieved by a curvature based strategy, such that initial tile density varies with curvature at different regions of a surface. In particular, the tile density is positively correlated with the curvature, i.e., more tiles with small sizes at high curved regions and vice versa, so as to make tiles closely capture surface features. In our implementation, we use the curvature to define a function on surface to control the desired tile size at a surface point p, denoted as $DTS(p)$. Then n points are uniformly sampled on the surface according to the density function derived from $DTS(p)$ by the blue noise sampling method in [33]. Tiles are chosen one by one from input in a round-robin way. If n is larger than the number of tiles, the procedure aforementioned is repeated until n tiles are distributed over the surface. Then the tiles are placed on the tangent planes at the selected points over the surface. In addition, all tiles are shrunk to be 10 percent of their desired sizes to remove potential overlaps between any tiles. Note that subsequent relaxation steps will make the tiles larger to reach their prescribed sizes.

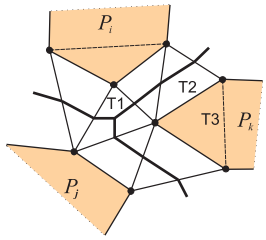


Fig. 2. Approximate construction of Chordal Axis Transform. T1, T2, and T3 are a junction triangle, an external triangle, and an internal triangle, respectively.

5.2 Approximate Chordal Axis Transform

Given a set of n disjoint tiles $\{P_1, \ldots, P_n\}$ with their centroids on the supporting surface, we would like to partition the supporting surface into a set of nonoverlapping regions, one surrounding each tile, so that we can optimize the configuration of each tile individually to improve the coverage of its containing region. We use the chordal axis transform (CAT) [34] for partitioning the surface.

In practice, we approximate a tile with a set of sufficiently dense and uniformly spaced sample points along its boundary. Furthermore, a Delaunay triangulation of these sample points over the supporting surface is constructed as in [35]. Then we compute the chordal axis [34] from the Delaunay mesh. We first classify the edges of the Delaunay mesh into two categories. Edges connecting vertices from two tiles are called external edges while the other edges are called internal edges. Triangles can also be classified into three categories: triangles with three external edges (junction triangle), triangles with two external edges(external triangle), and triangles without external edges (internal triangle), as shown in Fig. 2. The chordal axis is obtained by connecting the midpoints of the external edges of every external triangle and also connecting the midpoints of all edges of every junction triangle to the centroid of that triangle.

All edges created in the previous step form n connected 3D polygons, each of which surrounds one of the tiles, as shown in Fig. 1. We call these polygons CAT regions. In the optimization detailed in the next section, a tile will be restricted to lie on a tangent plane in every iteration. We project every CAT region onto the tangent plane containing the corresponding tile, resulting in a 2D projected CAT region for each tile. The projected CAT region of tile P_i is denoted V_i.

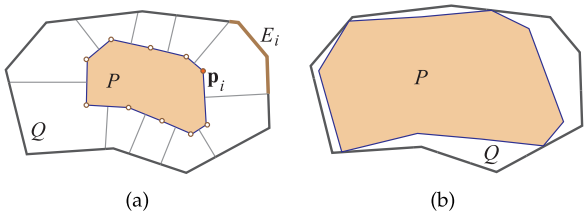


Fig. 3. Local constraints for extremal polygon optimization: (a) local constraints for vertex $\mathbf{p_i}$; (b) the extremal polygon obtained by our method.

5.3 Open Boundaries and Sharp Features

We treat open boundaries and sharp features on a supporting surface as follows. We sample a sufficient number of points on a mesh boundary and use them as barriers to stop tiles from moving beyond the boundary. The projected CAT regions associated with these points are sufficiently small when tiles are packed tightly near the boundary. In other words, these extra points do not occupy much surface area so that there is little gap between the tiles and the mesh boundary. In practice, this method makes tiles align well with the boundary (see Figs. 8 and 10). In addition, a sharp feature curve on the mesh surface is handled as a two-sided boundary in our implementation.

5.4 Tile Containment Optimization

In the relaxation process as described in Algorithm 2, we need to optimize the position, size, and orientation of a tile P_i within its projected CAT region V_i. We solve this problem by finding the largest copy of P_i inside V_i, since our goal is to maximize surface coverage. Because the projected CAT region V_i is actually a planar polygonal region, the problem is equivalent to solving extremal polygon containment problem [36] with two polygons. There has been much work on this problem as reviewed in Section 2. However, we cannot use the existing methods directly because those methods have restrictive assumptions on fixed rotations and convex polygons, while both the tile polygon and the container polygon may be concave in our setting.

We formulate extremal polygon containment as an optimization problem with nonlinear constraints as follows. Let P denote the inner tile polygon be P and Q the container polygon. Recall that the inner polygon P is approximated with a sequence of sample points, denoted $\{\mathbf{p_1}, \ldots, \mathbf{p_l}\}$. Our strategy is to define a feasible region delimited by a subset of edges of Q for each sample $\mathbf{p_i}$. We compute a Voronoi diagram for the set of sample points on P. Each edge of polygon Q must intersect with at least one of the Voronoi regions, as shown in Fig. 3. Let $\{\mathbf{e_1}, \ldots, \mathbf{e_m}\}$ be the set of edges of Q. We collect the subset of edges of Q that intersect with the Voronoi region of each sample $\mathbf{p_i}$, and denote it as E_i. In our formulation, sample $\mathbf{p_i}$ is constrained to be on the positive side of the edges in E_i. Note that the set of edges in E_i changes as the optimization proceeds. Therefore, we need to dynamically update E_i during the optimization.

Our formulation relies on the local coordinate frame associated with the polygon P. Let the translation of P be $\mathbf{t} = (t_1, t_2)$, the angle of the rotation around the origin be θ, and scaling, if allowed, be s. As P moves away from its initial position, the samples on P can be represented as functions of (s, θ, t_1, t_2) as follows,

$$\mathbf{p_i}' = \mathbf{p_i}(s, \theta, \mathbf{t}) = s \cdot R_\theta(\mathbf{p_i}) + \mathbf{t}, \quad 1 \le i \le n,$$

where R_θ is the 2D rotation matrix specified by the angle θ.

In the above context, we formulate the extremal polygon containment problem as a 2D constrained nonlinear optimization as follows.

$$\begin{aligned} \text{Maximize} \quad & f(s, \theta, \mathbf{t}) = s \\ \text{subject to} \quad & \frac{(\mathbf{p_i}' - \mathbf{v_j^0}) \times \mathbf{e_j}}{|\mathbf{e_j}|} \ge 0 \quad \text{for} \quad 1 \le i \le l, j \in E_i \\ & -\frac{\pi}{6} \le \theta \le \frac{\pi}{6}, \end{aligned} \tag{1}$$

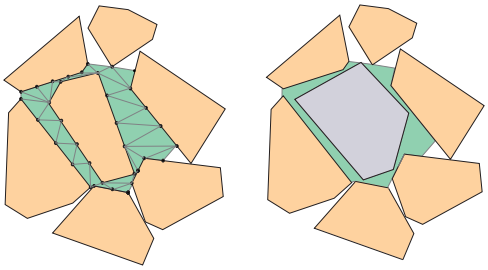
where $\mathbf{v_j^0}$ is the starting point of the directed edge $\mathbf{e_j}$. The objective function of the optimization is simply the scaling factor s, which directly controls the area of P. The amount of incremental rotation is constrained to a small range every time the set of constraining edges is updated, because the local containment constraints for P may not continue to hold if there is large step of rotation. An empirical range for the incremental rotation is $[-\pi/6, \pi/6]$, which always leads to stable results in our experiments.

Specifically, we introduce a set of scaling factor barriers $\{b_1, b_2, \ldots, b_l = 1.0 | b_i < b_{i+1}\}$, which is a monotonic sequence of numbers within the valid range for scaling factors. As shown in Algorithm 2, we increase an active barrier step by step from the smallest number to the largest in this sequence. At each step, we perform a number of relaxation iterations. Those tiles which scale fast would reach the active barrier first and have to wait for the others. Once the size of every tile has reached the active barrier, the active barrier is increased to the next level and tiles can be further scaled again.

We use the interior-point algorithm provided by the software library KNITRO [37] to solve the above optimization problem. It replaces a nonlinear programming problem with a series of barrier subproblems, and follows an interior path to the solution. The initial values for (s, θ, t_1, t_2) are set to $(1, 0, 0, 0)$. A local maximizer of the above optimization problem with this particular starting point is usually satisfactory for our application. In the presence of an active barrier for the purpose of synchronization among tiles, as discussed in Section 4, we simply make the scaling factor equal to the active barrier when the final scaling factor from the above optimization exceeds the active barrier.

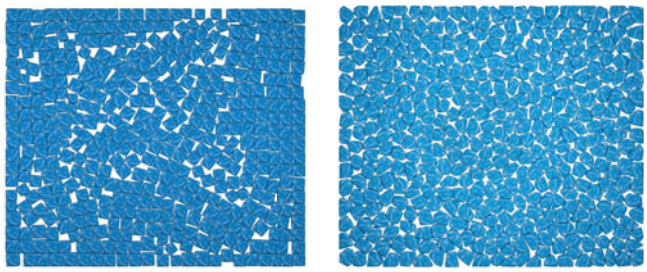
6 Tile Combination Optimization

Tiling results obtained from the relaxation process are often not visually satisfactory enough. There may be areas of uncovered space left within projected CAT regions due to shape mismatch between a tile and its surrounding tiles. We resort to tile combination optimization to further reduce the amount of uncovered space. Generally, for every tile, we extract the "hole" delimited by its surrounding tiles, and try to locate a different meta object that, after appropriate scaling, can better fit into the hole than its current occupant.



(a) (b)

Fig. 4. Tile combination optimization. (a) Region extraction; (b) Replaced tile with better shape compatibility with surrounding tiles.



(a) (b)

Fig. 6. An example of the comparison between uniform and variant shapes, both with 400 tiles. The coverage rate: (a) 78.3 percent (b) 86.4 percent.

6.1 Tile Permutation

Since we maintain a Delaunay mesh of all the sample points along the edges of tiles, the extraction of the uncovered hole region around a tile can be performed through a region growing scheme, as shown in Fig. 4a. Starting with the set of boundary edges of a tile that participates in a permutation, the region grows by including triangles incident to its boundary edges. In practise, the growth of the region stops if all the boundary edges either reach other tiles or are shorter than 10 percent of average edge length. Experimentally, if the short edges below this are included in a hole region, the shape of the region will become too complicated to be matched. Afterwards, to find another tile that can better fit into the extracted region, we exploit a shape matching method (Section 6.3).

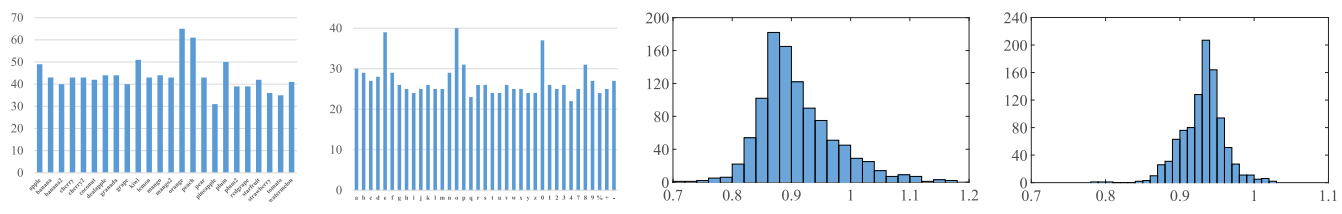
Given two shapes, the shape matching method returns a matching transformation and a similarity score between them. For each tile, we compute shape similarity scores between its hole region and all other tiles. We compute a new permutation of the tiles used in the layout as follows. Firstly, we build a bipartite graph, where each edge between the ith region and jth tile is associated with a weight defined by their similarity score. Our task is then finding the matching with the minimum accumulated weight in the bipartite graph. Such a matching corresponds to the best permutation of the tiles and is found by solving the assignment problem using the Hungarian algorithm [38]. Note that the hole regions of the tiles used in shape matching give an overlapping decomposition of the supporting surface. As a result, the new shapes of adjacent tiles after permutation may be overlapping with each other. To avoid this problem, we divide the tiles in the current layout into subsets such that the tiles in the same subset are not adjacent to each other. Two tiles are considered to be adjacent if there is at least one Delaunay edge connecting them.

6.2 Tile Replacement and Hole Filling

The tile replacement replaces a tile with another one (from the meta objects), which can better fill its surrounding region. However, the excessive use of this operation could lead to the loss of shape diversity in the layout, since it tends to favor some tiles than others in the resulting layout. Thus, we only replace tiles surrounded by large uncovered regions. We use the area ratio between a tile and its surrounding region to measure the uncovered space. Empirically, we sort all the area ratios in an increasing order and only replace the first 25 percent of these tiles. Figs. 5a and 5b illustrate the tile frequencies in two examples in Fig. 8 after replacement is performed three times. From the figures, it can be seen that certain tiles become more dominant than others Actually, this can be mitigated by limiting the number of times the replacement step is performed and the number of tiles to be replaced. Usually less than three rounds of replacement can lead to sufficiently high coverage, and more iterations only insignificantly improve the coverage.

Due to the limited number of shapes provided by the user, there may still be some relatively large uncovered areas on the supporting surface. We follow the same steps as in Section 6.1 to detect and extract holes. We fill every detected hole with a new tile. During hole filling, a new shape is selected through shape matching. In our experiments, we fill only those holes whose areas are larger than 50 percent of the mean area of all the tiles, since holes larger than this percentage are obviously noticeable.

Fig. 6 shows an example on combination optimization. Wherein, Fig. 6a contains a uniform tile shape only, for which the shape-based tile permutation and replacement have no effect. And Fig. 6b contains variant shapes, for which shape variety can be exploited for more compact arrangement. It can be observed that the optimization based on shape matching can improve coverage rate. And



(a) (b) (c) (d)

Fig. 5. Statistics of Fig. 8(d)&(f): (a)&(b) Tile appearance frequency after three rounds of replacement; (c)&(d) Histograms on ratio of tile size to DTS.

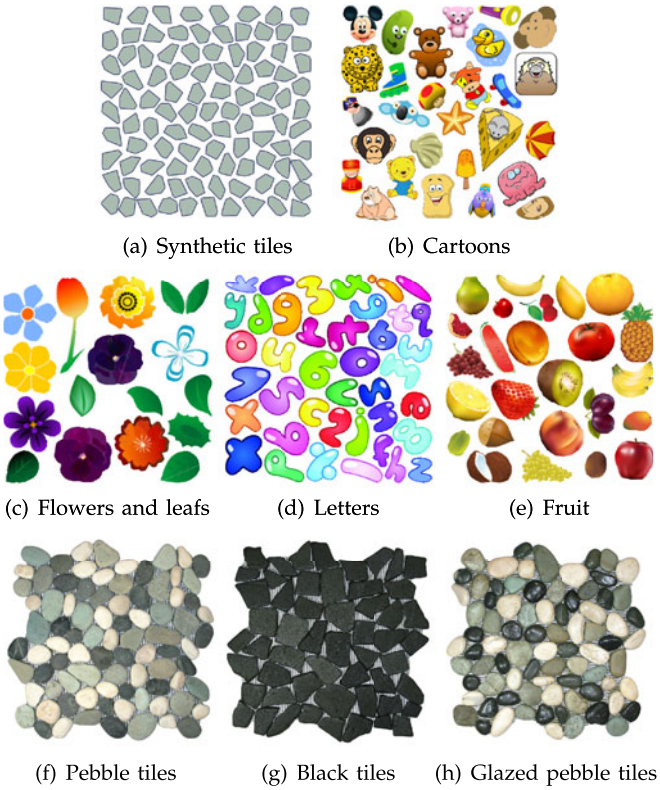


Fig. 7. Tile sets used in mosaic synthesis. The number of meta objects in tile sets (a-h) are 100, 28, 16, 40, 25, 70, 54, and 79, respectively.

Fig. 9 shows an example with very concave tiles, from which we can see that permutation and hole filling can match neighboring tiles in most local regions.

6.3 Shape Matching

There exist many shape matching techniques that work with different shape representations and transformations. We use the affine registration method in [39] to compute the matching transformation between two shapes. Only its similarity transformation part of the matching transformation is used in our solution.

Specifically, the best matching meta object for a target region is selected as follows. First, the centroid of every meta object is aligned with that of the target region. Second, an optimal 2D orientation of every meta object is computed by applying the registration method in [39]. Third, every meta object is uniformly scaled so that it has the same area as the target region. Finally, the scaled meta object with the largest common area shared between itself and the target region is chosen to be the best matching one. And the area of the symmetric difference of the two shapes is defined as the similarity metric.

Note that a meta object is not associated with a specific size during shape matching. All the tiles will be scaled smaller before the subsequent relaxation step. And again, the synchronization strategy for the scaling factor during relaxation will prevent tiles from becoming overly large or small.

7 Experiments and Results

We have implemented our algorithm in C++. All the experiments were conducted on a PC with a 2.4 GHz Intel quad-core processor and 8 GB memory. We use OpenMP to automatically parallelize the code for tile configuration optimization. The running time for each model depends mainly on the number of tiles used. Typically, our algorithm generates a high-quality synthesized surface mosaic in 10-20 minutes for a mesh covered with 2,000 tiles.

Eight sets of tiles were presented for mosaic synthesis in this paper, as shown in Fig. 7. These tile sets include synthetic shapes, like the random convex polygons in Fig. 7a, shapes cropped from clip arts as in Figs. 7b, 7c, 7d, and 7e as well as tile sets cropped from photographs of real-world tiles made from rocks and stones, as shown in Figs. 7f and 7g. Note that every tile set we use consists of irregularly shaped tiles.

In all our experiments, we use the following function to define the desired tile size at a mesh vertex:

$$DTS(v_i) = (|\kappa_{max}| + 1)^{-0.6},$$

where κ_{max} is the maximum principal curvature at the mesh vertex v_i, and the constant 1 is added to ensure that the denominator does not vanish. The desired tile size at any point of the mesh surface is then obtained by linear interpolation. Figs. 5c and 5d show histograms of the ratio between the resulting tile size and DTS computed at tile-surface touching points. From the two histograms, it can be seen that most tile sizes respect the surface curvature. Through size control, the ratio is restricted within the range $[0.7, 1.2]$. During the optimization discussed in Section 5, the size function is used to guide the initial distribution of tiles and the tile size at any point over a mesh surface. In all the rendered images, all the 2D planar tiles are extruding them off the base surface slightly so that the 3D appearance of the titles allows better appreciation of the mosaics. In particular, within highly curved surface regions, such extrusion may cause adjacent tiles to overlap. Therefore, over the regions with negative curvature, we perform collision detection after extrusion. As long as penetration occurs, the two colliding tiles are shrunk until they are completely separated.

We have tested our algorithm on various mesh models and tile sets to create a variety of surface mosaic patterns. Fig. 8 shows some of the results synthesized by our method. The coverage rate of the supporting surfaces is between 85 and 89 percent for the rock and stone tiles and between 72 and 80 percent for tiles cropped from clip arts. The tiles in these results faithfully follow the geometry of the supporting surface by varying their size according to the surface curvature. Fig. 11 shows the relationship between the number of tiles and coverage rate. And in most cases, more tiles lead to higher coverage rate.

Together with a preprocessing step of mesh segmentation, users can exploit our method on different segments using different tiles, in order to create multiple patterns on one single surface (Fig. 10). Since our method is capable of aligning tiles along open boundaries, the tiles on different segments can match well at the inter-segment borders, without leaving noticeable gaps.

7.1 Anisotropic Mosaics

It often happens that the orientations of the tiles are required to align with a given vector field, especially when the tiles have regular shapes with distinct orientations, such

Fig. 8. A gallery of synthesized mosaics on surfaces. From left to right and top to bottom: Hypersheet with the tile set in Fig. 7(f), #tiles $n = 1{,}869$, coverage rate $\alpha = 0.868$; Bunny with the tile set in Fig. 7(a), $n = 4{,}753$, $\alpha = 0.881$; Double torus with the tile set in Fig. 7(h), $n = 1{,}530$, $\alpha = 0.865$; Bowl with the fruit tile set, $n = 500$, $\alpha = 0.788$; Cloak with the flower tile set, $n = 800$, $\alpha = 0.726$; Egg with the tiles of letters, $n = 1{,}000$, $\alpha = 0.801$.

as rectangles and squares. Our method can easily be adapted for this type of anisotropic mosaic synthesis. When the tile relaxation algorithm (Algorithm 2) is applied, the rotation of a tile is not a variable subject to optimization, but determined by a vector field. That is, only tile translation and scaling are optimized. Fig. 12 shows an example where the orientations of the tiles are aligned with the principal directions of a surface.

7.2 Comparisons

We compare our method with the two surface mosaic methods in [24] and [40], which employ iterative centroidal Voronoi tessellation for planar mosaic generation. During each iteration, these methods compute a Voronoi tessellation of the underlying domain taking polygonal tiles as sites, move tiles to the centroid of their Voronoi cells, and reorient the tiles so that the principal direction of a tile matches the orientation of the closest domain boundary [24] or the principal direction of the tile's Voronoi cell [40]. The principal directions are estimated by principal component analysis. To improve the coverage rate of the results by these methods, we perform a simple scaling at the end so that every tile touches the boundary of its Voronoi cell. From Fig. 13, we see the resulting coverage rates of these

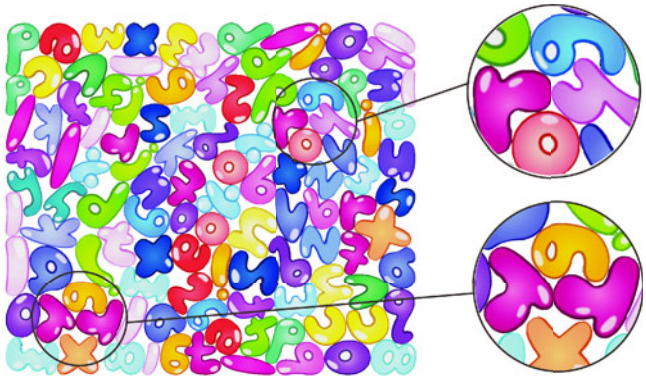


Fig. 9. 2D example with the concave alphanumeric tiles.

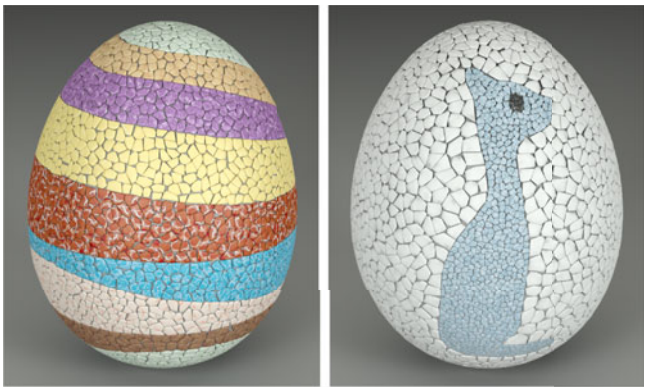


Fig. 10. Synthesized surface mosaics with user-customized segmentation and patterns.

Fig. 11. From left to right, the number of tiles: 400, 800, 1,200; coverage rate: 80.02, 83.19, 87.58 percent.

two techniques are typically at least 20 percent lower than the coverage achieved by our results.

Square tiles with the same size [27] or different sizes [28] have been used for surface mosaics. Dos Passos and Walter [29] present a simple method for visually simulating mosaics with irregular tiles over a sculpture surface. The tiles are computed using a Voronoi diagram defined with a distribution of points on the surface. We compare our method with this method in Fig. 14. With perfect edge alignment between adjacent tiles, the result obtained with the method in [29] does not resemble a man-made mosaic, where larger gaps between adjacent tiles are necessary to fill grout, and the edges of two adjacent tiles are not always perfectly aligned. In comparison, our result looks closer to a manually produced mosaic

Fig. 12. Anisotropic mosaic simulation on surface.

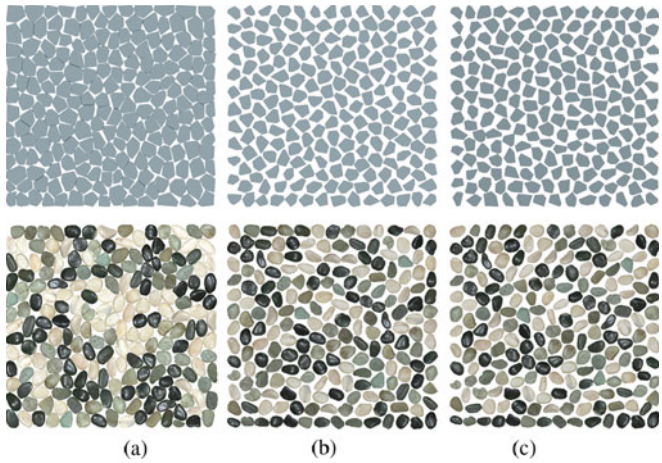


Fig. 13. A comparison with existing techniques using centroidal Voronoi tessellation. Two tile sets are used: (a) our results with coverage rates 84.7 percent (upper) and 84.3 percent (lower) ; (b) results by [24] with coverage rates 60.9 and 68.1 percent; (c) results by [40] with coverage rates 59.7 and 64.8 percent.

Fig. 14. A comparison with an existing technique for surface mosaic generation by using Voronoi polygons [29]. Left: result by [29]; middle&right: our results.

pattern. Most importantly, our method can handle any user-provided irregularly shaped tiles, while the tiles in Dos Passos and Walter's results must have the usual convex shapes generated as 2D Voronoi cells.

We have also conducted a comparison between surface mosaics generated by our method and those by manual design. We have developed a simple interface to allow a user to place tiles directly on a surface piece by piece.

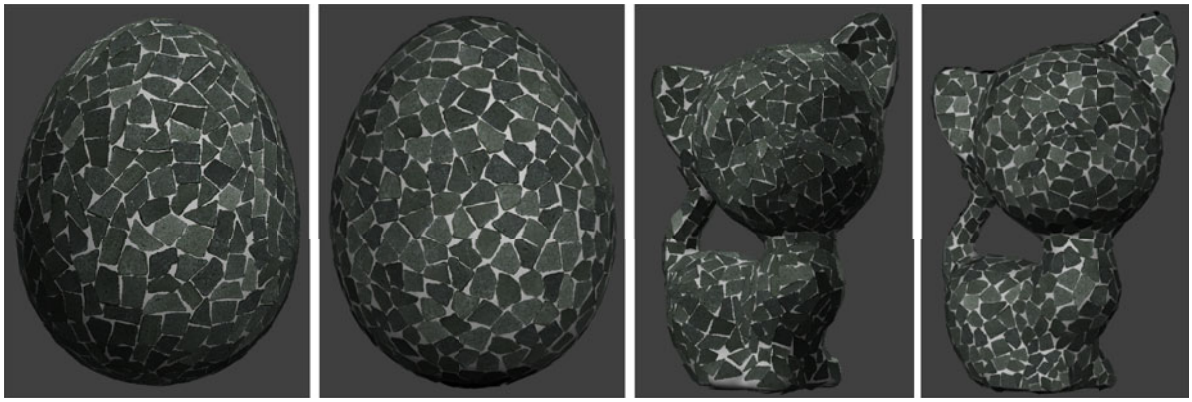


Fig. 15. A comparison with manual design. (a)&(c): results by manual design, finished in 246 minutes and 357 minutes respectively; (b)&(d) results by our method, finished in 17 minutes and 47 minutes respectively.

The user is also allowed to scale the tiles slightly to find a better fit. Fig. 15 shows results from both this manual design process and our automatic method. The manual design can make very nice surface mosaics at a large labor cost. Our automatic solution is much faster than manual work while both results are comparable in terms of aesthetic quality.

8 Conclusion

Automated synthesis of surface mosaic patterns helps make virtual worlds look more realistic. An important aspect of realism comes from the history of an artifact, or the process by which it is made. Therefore, we seek to compare our automated technique to designs produced by humans. We have presented a method for making synthetic surface mosaics with irregularly shaped tiles. It integrates continuous constrained optimization with discrete combinatorial optimization, and alternates between tile configuration optimization and tile combination optimization. Tile configuration optimization relies on a surface partition tailored for the tiles and optimizes the configuration of each tile to achieve a relatively tight fit within its own surface region. Tile combination optimization performs tile permutation and replacement to reduce uncovered surface area. These operations are complementary to each other and altogether they give rise to a near-optimal choice of tiles and their configuration. Extensive experiments and comparisons have demonstrated the effectiveness of our solution.

8.1 Limitations

To achieve better surface coverage and adapt to surface curvature variation, we apply tile scaling during the synthesis process and allow the tile sizes to be slightly different from their input sizes. As a consequence, our problem formulation is different from a packing problem and the method is not applicable to generating mosaic patterns on real physical objects with real tiles. Although our method is capable of handling generic tile shapes, its outcome is still dependent upon the shape distribution. Different input tile datasets vary primarily in terms of shape diversity and irregularity. Our method deal with tile shapes in the steps of tile permutation and replacement. Both steps give suggestions based upon the best similarity measure we can find. Hence in the case of a tile dataset with irregular shapes (e.g., the letters and numbers), for certain local regions, there may not exist a good matching tile, which comprises the coverage of those regions. From the perspective of performance, the method involves many iterations. Hence it will be quite valuable to find out a gradient-based solution. Also, we have only considered the shape characteristics of the tiles and the underlying surface. Hence, when tiles with different textures are provided, the mosaic pattern generated by our methods may look chaotic. Besides, only the coverage rate is taken into account as the objective. Therefore, as future work, the addition of other aesthetic criteria (e.g., tile distribution) into the optimization is worth exploration.

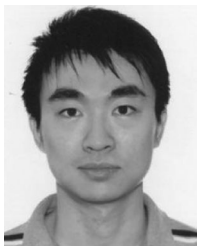
Acknowledgments

Zhonggui Chen's work is supported partially by National Natural Science Foundation of China (61472332), and the Fundamental Research Funds for the Central Universities (20720140520). Wenping Wang's research is supported partially by the GRF project (17208214) of Hong Kong Research Grant Council, NSFC projects (61272019, 61572021 and 61332015) and Shenzhen Science and Technology project (JCYJ20140903112959962). Zhonggui Chen is the corresponding author.

References

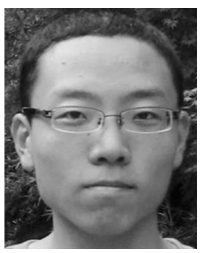
[1] S. King, *Mosaic Techniques & Traditions: Projects & Designs from Around the World*. New York, NY, USA: Sterling Publishing Co., August 28, 2006.
[2] R. Fowler, M. Paterson, and S. Tanimoto, "Optimal packing and covering in the plane are NP-complete," *Inf. Process. Lett.*, vol. 12, no. 3, pp. 133–137, 1981.
[3] K. Dowsland and W. Dowsland, "Solution approaches to irregular nesting problems," *Eur. J. Oper. Res.*, vol. 84, no. 3, pp. 506–521, 1995.
[4] J. A. Bennell and J. F. Oliveira, "A tutorial in irregular shape packing problems," *J. Oper. Res. Soc.*, vol. 60, no. s1, pp. 93–105, 2009.
[5] B. Baker, S. Fortune, and S. Mahaney, "Polygon containment under translation," *J. Algorithms*, vol. 7, no. 4, pp. 532–548, 1986.
[6] F. Avnaim and J. Boissonnat, "Polygon placement under translation and rotation," in *Proc. 5th Annu. Symp. Theoretical Aspects of Comput. Sci.*, 1988, vol. 294, pp. 322–333.
[7] R. Martin and P. Stephenson, "Putting objects into boxes," *Comput.-Aided Des.*, vol. 20, no. 9, pp. 506–514, 1988.

[8] R. B. Grinde and T. M. Cavalier, "Containment of a single polygon using mathematical programming," *Eur. J. Oper. Res.*, vol. 92, no. 2, pp. 368–386, 1996.
[9] V. J. Milenkovic, "Rotational polygon overlap minimization and compaction," *Comput. Geom.*, vol. 10, no. 4, pp. 305–318, 1998.
[10] F. Avnaim and J. Bsissonnat, "Simultaneous containment of several polygons," in *Proceedings 3rd Annu. Symp. Comput. Geom.*, 1987, pp. 242–247.
[11] R. Grinde and T. M. Cavalier, "A new algorithm for the two-polygon containment problem," *Comput. Oper. Res.*, vol. 24, no. 3, pp. 231–251, 1997.
[12] V. J. Milenkovic, "Multiple translational containment part ii: Exact algorithms," *Algorithmica*, vol. 19, no. 1, pp. 183–218, 1997.
[13] V. J. Milenkovic, "Rotational polygon containment and minimum enclosure using only robust 2d constructions," *Comput. Geom.*, vol. 13, no. 1, pp. 3–19, 1999.
[14] S. Battiato, G. Di Blasi, G. M. Farinella, and G. Gallo, "Digital mosaic frameworks—an overview," *Comput. Graph. Forum*, vol. 26, no. 4, pp. 794–812, 2007.
[15] G. Puglisi and S. Battiato, "Artificial mosaic generation," in *Proc. Image Video-Based Artistic Stylisation*, 2013, pp. 189–209.
[16] A. Hausner, "Simulating decorative mosaics," in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Tech.*, 2001, pp. 573–580.
[17] Y. Liu, O. Veksler, and O. Juan, "Generating classic mosaics with graph cuts," *Comput. Graph. Forum*, vol. 29, no. 8, pp. 2387–2399, 2010.
[18] J. Kim and F. Pellacini, "Jigsaw image mosaics," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 657–664, 2002.
[19] S. Battiato, A. Milone, and G. Puglisi, "Artificial mosaic generation with gradient vector flow and tile cutting," *J. Elect. Comput. Eng.*, vol. 2013, p. 8, 2013.
[20] S. Battiato, G. Di Blasi, G. Gallo, G. Guarnera, and G. Puglisi, "Artificial mosaics by gradient vector flow," in *Short Proc. EUROGRAPHICS*, pp. 581–588, 2008.
[21] G. Di Blasi and G. Gallo, "Artificial mosaics," *Visual Comput.*, vol. 21, no. 6, pp. 373–383, 2005.
[22] S. Battiato, G. Di Blasi, G. Farinella, and G. Gallo, "A novel technique for opus vermiculatum mosaic rendering," in *Proc. 14th Int. Conf. Central Eur. Comput. Graph., Vis. Comput. Vis.*, 2006, pp. 133–140.
[23] S. Hiller, H. Hellwig, and O. Deussen, "Beyond stippling—methods for distributing objects on the plane," *Comput. Graph. Forum*, vol. 22, no. 3, pp. 515–522, 2003.
[24] K. Smith, Y. Liu, and A. W. Klein, "Animosaics," in *Proc. Symp. Comput. Animation*, 2005, pp. 201–208.
[25] K. Dalal, A. Klein, Y. Liu, and K. Smith, "A spectral approach to NPR packing," in *Proc. 4th Int. Symp. Non-Photorealistic Animation Rendering*, 2006, pp. 71–78.
[26] B. Reinert, T. Ritschel, and H.-P. Seidel, "Interactive by-example design of artistic packing layouts," *ACM Trans. Graph.*, vol. 32, no. 6, p. 218, 2013.
[27] Y.-K. Lai, S.-M. Hu, and R. R. Martin, "Surface mosaics," *Visual Comput.*, vol. 22, nos. 9–11, pp. 604–611, 2006.
[28] V. A. Dos Passos and M. Walter, "3D mosaics with variable-sized tiles," *Visual Comput.*, vol. 24, nos. 7–9, pp. 617–623, 2008.
[29] V. Dos Passos and M. Walter, "3D virtual mosaics: Opus palladium and mixed styles," *Visual Comput.*, vol. 25, no. 10, pp. 939–946, 2009.
[30] H. C. da Gama Leitão and J. Stolfi, "A multiscale method for the reassembly of two-dimensional fragmented objects," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 9, pp. 1239–1251, Sep. 2002.
[31] Q.-X. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann, "Reassembling fractured objects by geometric matching," in *Proc. ACM SIGGRAPH*, 2006, pp. 569–578.
[32] C. Toler-Franklin, B. Brown, T. Weyrich, T. Funkhouser, and S. Rusinkiewicz, "Multi-feature matching of fresco fragments," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 185:1–185:12, Dec. 2010.
[33] Z. Chen, Z. Yuan, Y.-K. Choi, L. Liu, and W. Wang, "Variational blue noise sampling," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 10, pp. 1784–1796, 2012.
[34] L. Prasad, "Morphological analysis of shapes," *CNLS Newslett.*, vol. 139, pp. 1–18, 1997.
[35] R. Dyer, H. Zhang, and T. Möller, "Delaunay mesh construction," in *Proc. Symp. Geom. Process.*, 2007, pp. 273–282.
[36] S. Toledo, "Extremal polygon containment problems," in *Proc. Symp. Comput. Geom.*, 1991, pp. 176–185.
[37] R. Byrd, J. Nocedal, and R. Waltz, "Knitro: An integrated package for nonlinear optimization," in *Large-scale Nonlinear Optimization*, New York, NY, USA: Springer, 2006, pp. 35–59.
[38] J. Munkres, "Algorithms for the assignment and transportation problems," *J. Soc. Ind. Appl. Math.*, vol. 5, no. 1, pp. 32–38, 1957.
[39] J. Ho, A. Peter, A. Rangarajan, and M. Yang, "An algebraic approach to affine registration of point sets," in *Proc. 12th Int. Conf. Comput. Vis.*, 2009, pp. 1335–1340.
[40] L.-P. Fritzsche, H. Hellwig, S. Hiller, and O. Deussen, "Interactive design of authentic looking mosaics using Voronoi structures," in *Proc. 2nd Int. Symp. Voronoi Diagrams Sci. Eng.*, 2005, pp. 1–11.



Wenchao Hu received the BEng degree in computer science from the Shandong University. He is currently working toward the PhD degree in computer science at The University of Hong Kong. His research interests include computer graphics and computational geometry.

Zhonggui Chen received the BSc and PhD degrees in applied mathematics from the Zhejiang University, in 2004 and 2009, respectively. He is an associate professor at the Department of Computer Sciences, School of Information Science and Technology, Xiamen University, China. His research interests include computer graphics and computational geometry.



Hao Pan received the BEng degree in software engineering from the Shandong University, and the PhD degree in computer science from The University of Hong Kong. He is currently a researcher with the Microsoft Research Asia. His research interests include computer graphics and computational geometry.

Yizhou Yu received the PhD degree from the University of California, Berkeley, in 2000. He is currently a full professor at the Department of Computer Science, The University of Hong Kong, and an adjunct professor at the University of Illinois, Urbana-Champaign. He received the 2002 National Science Foundation CAREER Award and the Best Paper Award at 2005 and 2011 ACM SIGGRAPH/EG Symposium on Computer Animation. He is on the editorial board of Computer Graphics Forum and International Journal of Software and Informatics. He is the program chair of the Pacific Graphics 2009, Computer Animation and Social Agents 2012, and has served on the program committee of many leading international conferences, including SIGGRAPH, SIGGRAPH Asia, and International Conference on Computer Vision. His current research interests include data-driven methods for computer graphics and vision, digital geometry processing, video analytics, and biomedical data analysis.

Eitan Grinspun received the BASc degree in engineering science from the University of Toronto and the PhD degree in computer science from the California Institute of Technology, in 1997 and 2010, respectively. He is an associate professor of computer science at the Columbia University, and the director of the Columbia Computer Graphics Group. He was a professeur dUniversite Invite in Paris at the lUniversite Pierre et Marie Curie in 2009, and a research scientist at the Courant Institute of Mathematical Sciences in 2003-2004. His research seeks to discover connections between geometry, physics, and computation, typically with applications to computer graphics. He is an Alfred P. Sloan research fellow and US National Science Foundation (NSF) CAREER Award recipient, and was previously an NVIDIA fellow and a Caltech Everhart Distinguished lecturer. The technologies developed by his laboratory are used in consumer software such as Adobe Photoshop and Illustrator, at film studios such as Disney, Pixar, and Weta Digital, and in physics laboratories at institutions such as MIT and the Universite Paris VI. His work has been profiled in The New York Times, Scientific American, and Popular Science (Brilliant 10).

Wenping Wang received the PhD degree from the University of Alberta, Edmonton, Canada. He is a professor and the department head at the Department of Computer Science, The University of Hong Kong, Pokfulam. His research interests include computer graphics, visualization, and geometric computing. His current research interests include mesh generation and surface modeling for architectural design. He is a journal associate editor of *Computer Aided Geometric Design, Computers and Graphics, and IEEE Transactions on Visualization and Computer Graphics*, and the program co-chair of several international conferences, including Pacific Graphics 2003, ACM Symposium on Physical and Solid Modeling (SPM 06), Conference on Shape Modeling (SMI 09), and the conference chair of Pacific Graphics 2012 and SIGGRAPH Asia 2013. He is a member of the IEEE.