A Computational Algorithm for Creating Geometric Dissection Puzzles

Yahan Zhou Rui Wang
University of Massachusetts Amherst

Abstract

Geometric dissection is a popular way to create puzzles. Given two input figures of equal area, a dissection seeks to partition one figure into pieces which can be reassembled to construct the other figure. While mathematically it is well-known that a dissection always exists between two 2D polygons, the challenge is to find a solution with as few pieces as possible. In this paper, we present a computational method for creating geometric dissection puzzles. Our method starts by representing the input figures onto a discrete grid, such as a square or triangular lattice. Our goal is then to partition both figures into the smallest number of clusters (pieces) such that there is a *one-to-one* and *congruent* matching between the two sets. Directly solving this combinatorial optimization problem is intractable with a brute-force approach. We propose a novel hierarchical clustering method that can efficiently find an optimal solution by iteratively minimizing an objective function. In addition, we can modify the objective function to include an area-based term, which directs the solution towards pieces with a more balanced size. Finally, we show extensions of our algorithm for dissecting multiple 2D figures, and for dissecting 3D shapes.

Keywords: Geometric puzzles, dissection, optimization.

1 Introduction

Geometric dissection is a mathematical problem that seeks to cut one geometric figure into pieces which can be reassembled to construct other figures. For a long time, geometric dissections have enjoyed great popularity in recreational math and puzzles [Lindgren 1972; Frederickson 1997]. One of the ancient examples of a dissection puzzle was a graphical depiction of the Pythagorean theorem. Today, a popular dissection game is the Tangram puzzle [Slocum 2003], which uses 7 geometric pieces cut from a square to construct thousands of distinct shapes. Geometric dissection is also closely related to tiling and tessellation, both of which have numerous applications in computer graphics and computational geometry.

In its basic form, the geometric dissection asks whether any two shapes of the same area are equi-decomposable, that is, if they can be cut into a finite number of congruent polygonal pieces [Frederickson 1997]. Mathematically, it is long known that a dissection always exists for 2D polygons, due to the Bolyai-Gerwien theorem [Lowry 1814; Bolyai 1832; Gerwien 1833]. Although the theorem provided a general solution to find the dissection solution, the upper bound on the number of pieces is quite high. In practice, many dissections can be achieved with far fewer pieces. Figure 2 gives a simple example. Therefore much recent work has focused on the challenge of finding the optimal dissections using as few pieces as possible, and this has inspired extensive research in the mathematics and computation geometry literature [Lindgren 1972; Cohn 1975; Frederickson 1997; Czyzowicz et al. 1999; Kranakis et al. 2000; Akiyama et al. 2003]. While many ingenious analytic solutions have been discovered for shapes such as equilateral triangles, squares and other regular polygons, finding the optimal solution for general shapes remains a difficult open research problem.

In this work, our goal is to seek an efficient computational algorithm for the geometric dissection problem, and we use our solver to facilitate the creation of dissection puzzles. To do so, we employ

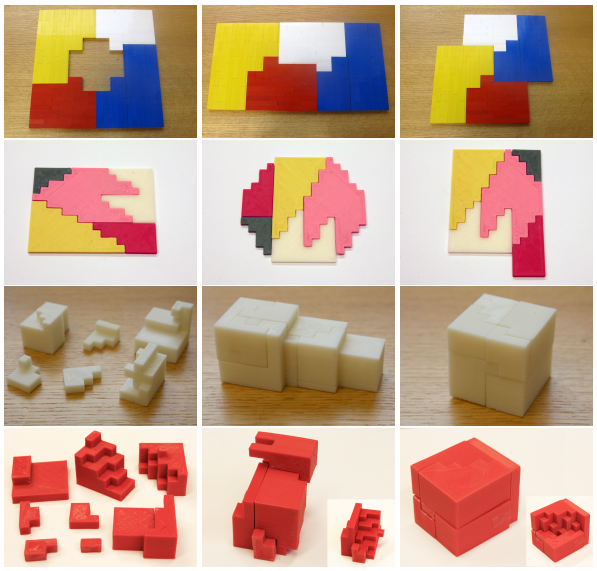


Figure 1: *Four example sets of dissection puzzles created using our algorithm. The top two rows show 2D dissections – the first is a 4-piece dissection, and the second is a 5-piece dissection of a rectangle with a rasterized octagon. The bottom two rows show 3D dissections – the first is a 6-piece dissection illustrating* $3^3+4^3+5^3 = 6^3$*, and the second is an 8-piece dissection between a polycube bunny model and a cuboid. The inlets show partial constructions. For each set we show the 3D pieces on the left, and the two shapes constructed from them on the right.*

an optimization approach that operates in a discrete solution space. Our method assumes that the input figures can be represented onto a discrete grid such as a square lattice. We rasterize each input figure into the lattice, and provide a simple editing interface to modify the rasterized figure or create one from scratch.

Following this step, we can reformulate the dissection into a cluster optimization problem. Specifically, our goal is to partition each figure into the smallest number of clusters (pieces) such that there is a *one-to-one* and *congruent* matching between the two sets. We consider two pieces congruent if they match exactly under isometric transformations, including translation, rotation, and flipping. As this is a combinatorial optimization problem, a brute-force solution is intractable even for small-scale problems. Therefore we propose a hierarchical clustering method that can efficiently find an optimal solution by iteratively minimizing an objective function. Our main idea is to start with two clusters in each figure, search for the partitioning that gives the best matching score, then progressively insert more clusters at each subsequent level until a dissection is found. The matching score is defined using a distance metric that penalizes mismatches. During optimization, we prioritize the search towards directions that are more likely to reach a dissection. Our algorithm can efficiently converge to a solution with a small number of pieces. Furthermore, we have found our solutions to be optimal for all test cases that we can verify optimality (see Section 4).

With the computational approach, we can extended the creation of puzzles in several ways. First, we can replace the square lattice with a triangular lattice which can account for 45° angled edges in

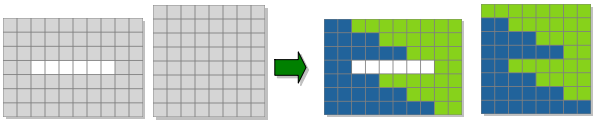


Figure 2: *An example of dissecting two shapes (a 7×10 rectangle with a center hole and a 8×8 square) using only two pieces.*

the input figures. Other regular grids, such as the hexagonal lattice, are also possible. Second, we can modify the objective function to include an area-based term, which favors pieces with a more balanced size. This can help avoid solutions where some pieces are significantly larger than other pieces, which can reduce the playability of the puzzles. Third, we show an extension of our algorithm to dissecting multiple input figures. We propose a global refinement to simultaneously optimize all input figures, instead of a trivial approach that simply overlays the pairwise dissections. Finally, we have also extended our algorithm to dissecting 3D shapes, thus creating 3D geometric puzzles. Figure 1 shows several examples produced using our method.

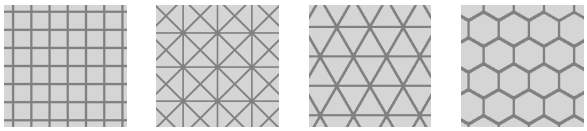
It should be noted that as we require the input to be discretized, our method is not meant to substitute the analytic approaches to many general dissection problems. Rather, our aim is to find an efficient computational solution, which provides a convenient tool for users to create a variety of different dissection puzzles.

2 Related Work

Geometric Dissections. Geometric dissection problems have a rich history, originating from the explorations of geometry by the ancient Greeks [Allman 1889]. One of the earliest examples is a visual proof of the Pythagorean theorem by using dissections to demonstrate the equivalence of area. In Arabic-Islamic mathematics and art, dissection figures are frequently used to construct intriguing patterns ornamenting architectural moments [Özdural 2000]. Dissection figures also provide a popular way to create puzzles and games. The Tangram [Slocum 2003], which is a dissection puzzle invented in ancient China, consists of 7 pieces cut from a square and then rearranged to form a repertoire of other shapes.

In mathematics, an early significant result was the proof that any 2D polygon can be dissected using a finite number of pieces to other polygons of equal area [Lowry 1814; Wallace 1831; Bolyai 1832; Gerwien 1833] (although the same conclusion does not hold for 3D shapes [Dehn 1900]). This has commonly been referred to as the Bolyai-Gerwien theorem. Since then, attention has focused on the more challenging problem of finding optimal dissections that use the fewest number of pieces. For example, Cohen [1975] studied economical triangle to square dissections; Kranakis et al. [2000] studied the asymptotic number of pieces to dissect a regular m-gon into a regular n-gon; Akiyama et al. [2003] studied the optimality of a dissection method for turning a square into n smaller squares; Czyzowicz et al. studied the number of pieces to dissect a rational rectangle into a square [1999], and under the additional constraints of glass cuts [2007]. In addition, the popularity of such problems is culminated in seminal books such as [Lindgren 1972; Frederickson 1997]. Despite extensive research, finding the minimum dissection solution has so far only been possible for a few special cases, while the general cases remain an open research problem. Our work is first to present a computational algorithm to solve a general dissection problem in discrete domain.

Another research area is dissections with special properties, such as hinged dissections where all pieces are hinged together at vertices, and remain connected as they are rearranged. An early example was demonstrated by [Dudeney 1902] that turns an equilateral triangle to square. Such an intriguing construction inspired a number of



(a) square *(b) right tri.* *(c) equilateral* *(d) hexgon*

Figure 3: *Examples of several discrete lattice grids.*

studies including a well-known book by Frederickson [2002]. Recently, Abbott et al. [2008] proved that any two polygons of equal area have a hinged dissection, resolving a long-standing open problem. Other types of hinges have also been studied, including twisted hinges [Frederickson 2007] and piano hinges [Frederickson 2006].

Tiling. A closely related subject to geometric dissections is tiling [Grünbaum and Shephard 1986], the basic form of which is to seek a collection of figures that can fill the plane infinitely with no overlaps or gaps. The use of tiling is ubiquitous in the design of patterns for architectural ornaments, mosaics, fabrics, carpets, and wallpapers. It is also seen throughout the history of art, especially in the drawings of M.C. Escher. In computer graphics, Kaplan and Salesin [2000] presented a technique called 'escherization', which can approximate any closed figure on the plane into a tileable shape, simulating Escher-style drawings. A number of well-known tiling patterns, such as Penrose tiling, polyomino tiling, Wang tiles have also been cleverly applied in graphics, especially for blue noise sampling [Ostromoukhov et al. 2004; Ostromoukhov 2007] and texture synthesis [Cohen et al. 2003; Fu and Leung 2005; Lagae and Dutré 2006]. An excellent introduction and survey of tile-based methods in computer graphics can be found in [Lagae et al. 2008].

Tiling can also be used to create puzzles. Lagae and Dutré [2007] have shown that the tile packing results can be used to create interesting jigsaw puzzles. Another relevant work is a method for creating 3D polyomino puzzles presented by [Lo et al. 2009]. Their method aims to find a set of polyomino pieces that can tile a given parameterized surface, and they designed clever interlocks to make the puzzles physically realizable. Generally, tile-based puzzles study how to use a predefined set of pieces to cover a given shape; in contrast, geometric dissection puzzles study how to solve for a set of pieces that can simultaneously construct two or more shapes. Thus their solution methods are considerably different.

Recreational Math and Art. Our work relates to a number of topics in computer graphics that are targeted towards recreational math and art, such as 3D Burr puzzles [Xin et al. 2011], ASCII art [Xu et al. 2010], paper popup [Li et al. 2011; Li et al. 2010], camouflage images [Chu et al. 2010], shadow art [Mitra and Pauly 2009], 3D polyomino puzzles [Lo et al. 2009], maze construction [Xu and Kaplan 2007], papercraft models [Mitani and Suzuki 2004], jigsaw image mosaics [Kim and Pellacini 2002]. Solutions to many of them involve solving a complex optimization problem. For example, Chu et al. [Chu et al. 2010] used a multi-label graph cut algorithm to solve an pixel labeling problem. In general, our formulation for geometric dissections can be viewed as a label assignment problem (the label being the index of a piece). However, we haven't found any existing solution that can directly benefit our case. This is mainly because unlike in image domains, our objective function cannot be defined using a local coherence metric, thus an algorithm such as graph-cuts is not applicable.

3 Algorithms and Implementation

3.1 Assumptions and Overview

Given two input figures $\mathcal{A}$ and $\mathcal{B}$ of equal area, our goal is to find the minimum set of pieces to dissect $\mathcal{A}$ and $\mathcal{B}$. To formulate it as

an optimization problem, we require both input figures to be represented onto a discrete grid. The simplest choice is a square lattice as shown in Figure 3(a), which is naturally suitable for representing rectilinear polygons. For other shapes, such as discs, we rasterize them into the grid, resulting in approximated shapes. Note that for the purpose of creating puzzles, exact representation of the input is not necessary. At sufficient grid resolution, the discritization typically produces acceptable shape approximations. Note that after discretization, the area (number of pixels) covered by each figure must remain the same. This can be ensured either by the design of the input figures, or by using a graphical interface (see Section 3.7) to touch up the rasterized figures. In the following, we use symbols $\mathcal{A}$ and $\mathcal{B}$ to denote the two rasterized figures of equal area.

Given the input, we formulate the dissection into a cluster optimization problem. Specifically, our goal is to partition each figure into the smallest number of clusters (each cluster being a connected piece) such that there is a one-to-one and congruent matching between the two sets of clusters. Here congruency refers to two pieces that match exactly under isometric transformations, including translation, rotation, and flipping. Since the solution space is discrete, the possible transformations are also discrete. For example, on a square lattice with the grid size 1, all translations must be of integer values, and there are only 4 possible rotations: 0°, 90°, 180°, and 270°. Thus excluding translation, two congruent pieces must match under the 8 different combinations of rotation and flipping.

Generally, solving such a clustering problem requires combinatorial search, which would impose a very large solution space. As the dissection requires the solution pieces to fit exactly with each other in both input figures, leaving no holes or overlaps, standard fitting or clustering algorithms are unlikely to lead to valid results. To efficiently solve the problem, we introduce a hierarchical clustering algorithm that progressively minimizes an objective function until a solution is found. We start the search from a random initial condition, and apply refinement steps to iteratively reduce the objective function value. We use random exploration to keep the algorithm from getting stuck in local minima. Below we will first describe our algorithm for dissecting two input 2D figures defined on a square lattice, then describe its extensions to the triangular lattice, the dissection of multiple figures, and finally the dissection of 3D shapes. Figure 4 provides a graphical overview of the algorithm.

3.2 Dissecting Two Figures on a Square Lattice

Distance metric. Given two pieces on each figure: $\mathbf{a} \subset \mathcal{A}$, $\mathbf{b} \subset \mathcal{B}$, we define a distance metric D that measures the bidirectional mismatches between them under the best possible alignment:

$$D(\mathbf{a},\mathbf{b}) = \min_{T_{\mathbf{a},\mathbf{b}}} \Big(\big\| \{\, p \,|\, p \in \mathbf{a} \text{ and } (T_{\mathbf{a},\mathbf{b}} \times p) \notin \mathbf{b} \} \big\| + \big\| \{\, p \,|\, p \in \mathbf{b} \text{ and } (T^{-1}_{\mathbf{a},\mathbf{b}} \times p) \notin \mathbf{a} \} \big\| \Big) \quad (1)$$

where $T_{\mathbf{a},\mathbf{b}}$ is an isometric transformation from piece $\mathbf{a}$ to $\mathbf{b}$, $T^{-1}_{\mathbf{a},\mathbf{b}}$ is the reverse transformation, p counts the number of pixels that in one piece but not the other (i.e. it measures bidirectional mismatches). As D measures the minimum mismatches under all possible $T_{\mathbf{a},\mathbf{b}}$, it will be 0 if the two pieces are congruent.

To simplify the calculation of D, we first set the translation to align the centers of $\mathbf{a}$ and $\mathbf{b}$ together, then simply search among the 8 combinations of rotation and flipping to obtain D. While this does not consider other possible translations, we found it to work well in practice, and it preserves the crucial property that congruent pieces must result in zero distance. Note that if the center of a piece does not lie exactly on a grid point, we need to align it to the 4 nearby grid points and calculate D for each; the smallest among them is returned as the distance value.

Matching. Next, assume the two figures $\mathcal{A}$ and $\mathcal{B}$ have both been partitioned into k clusters $\{\mathbf{a}_i\}$ and $\{\mathbf{b}_j\}$, we need to match the elements in $\{\mathbf{a}_i\}$ to those in $\{\mathbf{b}_j\}$ such that the sum of distance between every matched pair is minimized. We call this a *matching* between the two sets, denoted as M. Mathematically,

$$M = \operatorname*{arg\,min}_{m \in \{\{\mathbf{a}_i\} \to \{\mathbf{b}_j\}\}} \sum_{(\mathbf{a}_i,\mathbf{b}_j) \in m} D(\mathbf{a}_i, \mathbf{b}_j) \quad (2)$$

where $\{\mathbf{a}_i\} \to \{\mathbf{b}_j\}$ denotes a bijection from $\{\mathbf{a}_i\}$ to $\{\mathbf{b}_j\}$. Basically we are seeking among all possible bijections the one that gives rise to the minimum total distance. This is known as the assignment problem in graph theory, which is well-studied and can be solved by a maximum weighted bipartite matching. Specifically, we create a weighted bipartite graph between the two sets $\{\mathbf{a}_i\}$ and $\{\mathbf{b}_j\}$: every element $\mathbf{a}_i$ in $\{\mathbf{a}_i\}$ is connected to every element $\mathbf{b}_j$ in $\{\mathbf{b}_j\}$ by an edge, whose weight is equal to the distance D between the two elements. The goal is to find a bijection whose total edge weight is minimal. A common solution is based on a modified shortest path search, for which we use an optimized Bellman-Ford algorithm [West 2000]. It guarantees to yield the best matching in $O(k^3)$ time, where k is the number of clusters.

We call the total pair distance under M the *matching score*, denoted as E_M. In other words, $E_M = \sum_{(\mathbf{a}_i,\mathbf{b}_j) \in M} D(\mathbf{a}_i, \mathbf{b}_j)$. Note that $E_M = 0$ if M is a dissection solution. Thus the smaller E_M is, the closer we are to reach a dissection solution.

Objective function. Since the minimum number of pieces to achieve a dissection is unknown in advance, we propose a hierarchical approach that solves the problem in multiple levels. Each level ℓ partitions the two input figures into $\ell + 1$ clusters, and outputs a set of best candidates at that level. The basic definition of such an objective function is simply the matching score E_M. Specifically, let's denote with $C_k = (\{\mathbf{a}_i\}_k, \{\mathbf{b}_j\}_k)$ a candidate solution where $\{\mathbf{a}_i\}_k$ and $\{\mathbf{b}_j\}_k$ are two given k-piece clusterings of $\mathcal{A}$ and $\mathcal{B}$ respectively; then the objective function $E_k(C)$ is:

$$E(C_k) = E_M(\{\mathbf{a}_i\}_k, \{\mathbf{b}_j\}_k) \quad (3)$$

At the end of each level ℓ, we select a set (N_b) of the best candidate solutions $\{S_\ell\}$ which give the smallest values according to Eq 3, and use the set for initialization in the next level. The algorithm will terminate when a solution is found such that $E(S_\ell) = 0$.

In the following we will describe our algorithms for the first and each subsequent level. Refer to Figure 4 for a graphical illustration.

3.2.1 Level 1 Optimization

Seeding. At the first level, our goal is to compute the best set of 2-piece clusterings to approximate the dissection. To begin, we split $\mathcal{A}$ into two random clusters $\mathbf{a}_1$ and $\mathbf{a}_2$. This is done by starting from two random seeds, and growing each into a cluster using flood-fill. While we could also use other methods to grow the clusters, the flood-fill guarantees that each cluster is a connected component. We do the same for $\mathcal{B}$, resulting in two random clusters $\mathbf{b}_1$ and $\mathbf{b}_2$.

Compute matching. Now we have the initial sets of clusters $\{\mathbf{a}_i\}_2$ and $\{\mathbf{b}_j\}_2$, we can invoke Eq 2 to compute the matching M between them. In Figure 4 we use the same color to indicate a matched pair. Note that there is no particular ordering of the clusters, so the colors may flip depending on the output of the matching algorithm.

Forward copy-paste. Our next step is to refine the clusters. As the solution space is very large, randomly modifying each cluster by itself is unlikely to result in a better matching score. Therefore we introduce a more explicit approach that copies and pastes a cluster $\mathbf{a}_i$ to its matched cluster $\mathbf{b}_j$, in order to force their shapes to become

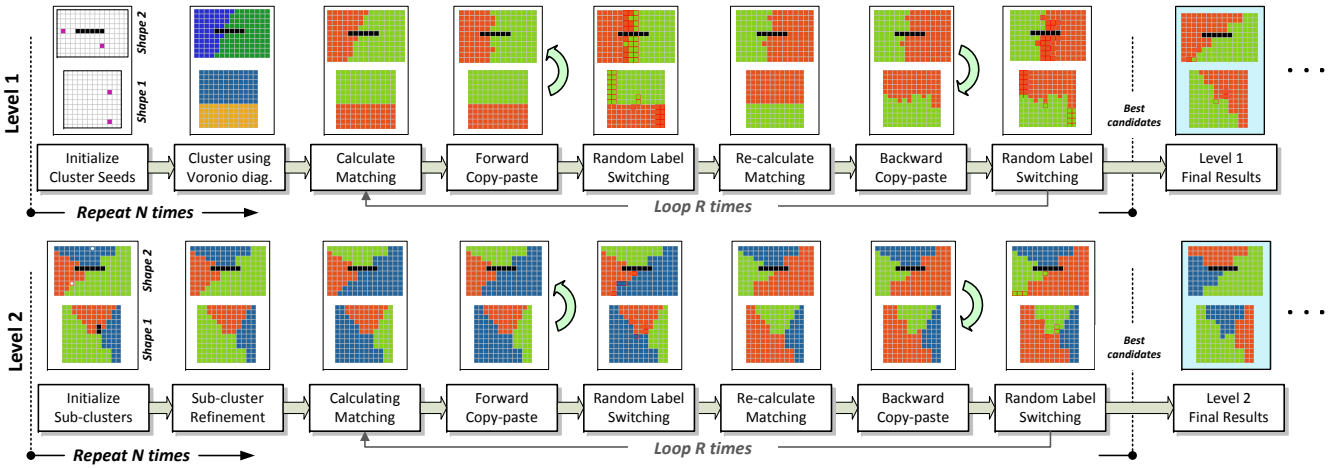


Figure 4: *An overview of our hierarchical optimization algorithm for dissecting two input figures: one is a 15×10 rectangle with an off-center hole, and the other is a 12×12 square. At each level, we show the steps being performed, and visualize the changes in one of the candidate solutions after each step. This example requires 3 pieces to dissect, which was found by our algorithm at the end of level 2.*

similar. This is called a *forward copy-paste*. To do so, we apply the transformation which yields the distance between $\mathbf{a}_i$ and $\mathbf{b}_j$ (Eq. 1) to $\mathbf{a}_i$, and pastes the result to $\mathcal{B}$. Note that if the two matched clusters are not congruent yet, the paste may overwrite neighbor pixels that belong to other clusters. This is allowed, but we randomize the ordering of clusters for copy-paste in order to avoid bias. Pixels pasted outside the boundary of a figure are ignored.

Following the above step, some pixels in $\mathcal{B}$ may have received no pasted pixels from $\mathcal{A}$, thus they become holes. We use a random flood-fill to eliminate the holes. Specifically, we randomly select already pasted pixels and grow them outward to fill the hole.

Random label switching. As mentioned above, during copy-paste, some clusters may overlap with each other, resulting in conflicts. Therefore our next step is to reduce such conflicts by modifying the cluster assignments for some pixels at the boundary of two clusters. To do so, we first recompute the matching between the current two sets of clusters, then simulate a copy-paste in the backward direction, i.e. from $\mathcal{B}$ to $\mathcal{A}$. During this process we record the pixels that would have overlapped after pasting. For each such pixel x, we randomly relabel it to the cluster of one of its four neighboring pixels. This is called *random label switching*. Note that if x is surrounded completely by pixels of its own cluster, its label will remain the same. Thus only pixels on the boundary of a cluster can potentially be switched to a different label.

Intuitively, the motivation of the forward copy-paste is to encourage the clusters in $\mathcal{B}$ to be shaped similarly to $\mathcal{A}$, and the motivation of the random label switching is to modify the cluster boundaries in $\mathcal{B}$ to reduce cluster conflicts/overlaps. The two steps combined is called a **forward refinement** step.

Backward refinement. The backward refinement performs exactly the same steps as the forward refinement, except in the reverse direction (i.e. a copy-paste from $\mathcal{B}$ to $\mathcal{A}$, followed by a random labeling switching in $\mathcal{A}$). At this point, we have completed one iteration of back-and-forth refinement.

Convergence. We repeat the back-and-forth refinement iteration for R times (the default value of R is 100). This typically reaches convergence very quickly, upon which we obtain a candidate solution C_2, whose associated objective function value is $E(C_2)$.

Random seed exploration. The refinement process can be seen as a way to find local minimum from the initial seeds. Thus small changes to the initial seeds do not significantly affect the converged result. In order to seek global minimum, we apply random exploration, where we re-compute the the candidate solution N times (the default value of N is 400), each time with a different set of initial seeds. After random exploration, the best $N_b = 30$ candidate solutions (i.e. those with the smallest objective function values) are selected and output as the level 1 final results, denoted as $\{S_1\}$.

At this point, if there exists a candidate solution whose matching score is 0, we have reached a perfect dissection. Otherwise we continue with subsequent levels. The top portion of Figure 4 illustrates all steps in level 1. Note how the candidate solution refines following each step. The red outlines on some pixels indicate unmatched pixels between a pair of clusters which are not congruent yet.

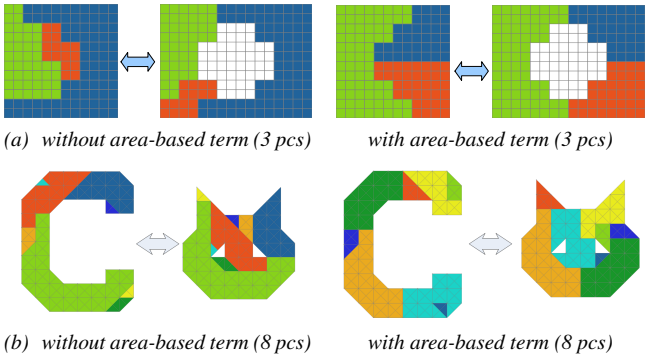
3.2.2 Level ℓ Optimization

In each subsequent level ℓ, we start from one of the best candidate solutions S_ℓ from the last level. Our goal is to insert a new cluster to S_ℓ, and then search for the best $(\ell+1)$-piece approximation using the same back-and-forth refinement process as in level 1. Intuitively, as the output of the previous level are some of the closest ℓ-piece approximations to the dissection, they serve as excellent starting points for the new level.

The main difference between level ℓ and level 1 is in creating the initial clusters. Note that we cannot use completely random seed initialization as in level 1, because doing so will completely abandon the results discovered in previous levels, and hence will not reduce the problem complexity. Instead, we introduce two heuristics to create initial clusters by exploiting the previous results, and we consider them both during the random exploration step.

Splitting an existing cluster. In the first heuristic, we select a pair of pieces $\{\mathbf{a}_i, \mathbf{b}_j\}$ from S_ℓ that has the largest (worst) matching score, and split each into two sub-clusters. We refer to the pair as the parent clusters. The splitting introduces an additional cluster for each figure; the remaining clusters which were not split remain unchanged for now. Next we need to decide how to perform the split. A straightforward way is random split, but as the parent clusters are not well matched, a random split can create difficulties for convergence in subsequent steps. Therefore we need to optimize the splitting to create better matched sub-clusters to begin with.

It turns out that we can optimize the splitting by using the same



(a) without area-based term (3 pcs) *with area-based term (3 pcs)*

(b) without area-based term (8 pcs) *with area-based term (8 pcs)*

Figure 5: *Comparing solutions computed with and without area-based term. In both examples (a) and (b), two solutions are shown which achieve the dissection with equal number of pieces. Note how the area-based term leads to results where the size of each piece is more balanced, which is usually more preferrable.*

approach as level 1 optimization. To do so, we treat the parent clusters $\mathbf{a}_i$ and $\mathbf{b}_j$ as two input figures, and apply level 1 optimization to obtain the best 2-piece dissection between them. We have found this approach to work well in practice, creating sub-clusters that are matched as well as it can. Experiments show that this can significantly improve the quality of the subsequent refinement results.

Creating new clusters from mismatched pixels. Our second heuristic is to create a new cluster from the currently mismatched pixels. For example, assume $\{\mathbf{a}_i, \mathbf{b}_j\}$ are a matched pair but not yet congruent, then transforming $\mathbf{a}_i$ to $\mathbf{b}_j$ will result in some pixels that are not contained in $\mathbf{b}_j$. These pixels will be marked in $\mathbf{a}_i$ as mismatched pixels. In Figure 4 the mismatched pixels are indicated with a red outline. After marking all mismatched pixels in $\mathcal{A}$ and $\mathcal{B}$, we randomly select a seed from them and perform a flood fill to grow the seed into a cluster, which then becomes a new cluster to be inserted to the current level.

Comparing the two heuristics. The rationale behind the first heuristic is that priority should be given to splitting the worst matched pair, as this is most likely to result in reduced matching score. The rationale behind the second heuristic is that when a candidate solution is very close to reaching a dissection, priority should be given to the few pixels that remain unmatched. In practice, we account for both of them during our random exploration: among the N random tries, 75% will use the first heuristic to initialize the sub-clusters, and 25% will use the second heristic. This way we can combine the advantages of both.

Global refinement and random exploration. Once the sub-clusters are created, we perform the same back-and-forth refinement process as in level 1. Now all clusters will participate in the refinement, therefore we call this step global refinement. Upon convergence, we obtain a candidate solution $C_{\ell+1}$.

In addition, we perform random exploration for N times similarly to level 1, the goal of which is to seek global minimum. Each exploration starts from a randomly selected best candidate S_ℓ from the previous level, applies one of the two heuristics to insert a new cluster, and computes refinement. Again, after random exploration, the best N_b candidate solutions are output as the final results $\{S_\ell\}$ of level ℓ. Figure 4 shows an example of level 2 optimization. For this example, our algorithm discovered a perfect dissection at the end of level 2, thus the program terminates with a 3-piece dissection.

3.3 Area-Based Term

So far we have described a computational algorithm for finding the minimum dissection of two figures. However, there is no constraint on the size or shape of the resulting pieces. Thus a solution may contain pieces that are significantly larger than others. This is often undesirable, both for aesthetic reasons and for reducing the difficulty of the puzzles (since large pieces are easier to identify and place on a target figure). For better control of the solution, we introduce an area-based term into our objective function in order to favor a solution where the size of each piece is more balanced. To do so, we modify Eq 3 to include the area-based term E_α:

$$E = E_M(\{\mathbf{a}_i\}_k, \{\mathbf{b}_j\}_k) + \lambda \cdot [E_\alpha(\{\mathbf{a}_i\}_k) + E_\alpha(\{\mathbf{b}_j\}_k)] \quad (4)$$

where λ is a weight adjusting the relative importance of the two terms. Here E_α is the total area penalty. It is defined by summing up the area penalty $\alpha(\mathbf{a}_i)$ of each piece, which is calculated as:

$$\alpha(\mathbf{a}_i) = \begin{cases} A(\mathbf{a}_i)/\bar{A} - 1 & \text{if } A(\mathbf{a}_i) > 2\,\bar{A} \\ \bar{A}/A(\mathbf{a}_i) - 1 & \text{if } A(\mathbf{a}_i) < \bar{A}/2 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

In the above equation, A denotes the area of a piece, and $\bar{A}$ denotes the average area (i.e. the total area divided by the number of pieces). Essentially $\alpha(\mathbf{a}_i)$ penalizes a piece if it is either more than twice the average area, or less than half of the average area; otherwise we consider a piece to be within the normal range of size variations and assign a zero penalty.

Figure 5 shows an example comparing solutions computed with and without applying the area-based term. Note that while both solutions achieved equal number of pieces, enabling the area penalty leads to pieces of a more balanced size, which is often preferable. In addition, more uniformly sized pieces also tend to be symmetric with each other, which is a desirable property.

Note that the preference towards balanced area and the preference towards smaller number of pieces are often conflicting goals. For example, if the area weight factor λ is set too large, the solution will be heavily biased towards area uniformity, and will deviate from the goal of seeking the smallest number of pieces. To address this issue, we gradually decrease λ as the level ℓ increases. This will reduce the effect of area penalty over levels, encouraging the solver to focus more on finding the minimum solution as the level increases. Our current implementation sets $\lambda = \frac{1}{2}\,0.8^{\ell-1}$.

Avoiding split pieces. Another improvement we made to the objective function is to include a term that penalizes split pieces. A split piece is one that contains disconnected components. While these components transform together in the same way, they are not physically implementable. Thus we simply add a large penalty to such pieces in order to eliminate them during best candidate selection. Note that we do not actively prevent them because there are cases where split pieces are temporarily unavoidable, such as during the first several levels of processing when the input figures themselves are fragmented (see Figure 11 (c)-(f)).

3.4 Extension to the Triangular Lattice

Besides using a square lattice, our method can also be extended to other lattices including the ones shown in Figure 3 (b,c,d). Currently we have implemented the right triangular lattice shown in (b), which is constructed by splitting each grid on a square lattice to four isosceles triangles along the diagonals. Using this lattice, we can represent input figures with both rectilinear edges as well as 45° angled edges. This usually makes the discrete representation more expressive. Figure 9 shows several examples.

With the triangular lattice, our algorithms remain almost the same, because the possible transformations, including translation, rotation, and flipping, remain the same with a square lattice. The main difference is that a triangle pixel has three neighbors (those connected to it along the three edges) while a square pixel has four.

3.5 Extension to 3D Shape Dissection

We can also extend our algorithm to dissecting 3D shapes that are represented onto a cubic voxel grid. In this case, each voxel has six neighbors, and the transformation of each piece considers 24 different 3D rotations. However, unlike 2D, a piece is not allowed to be mirrored (which is analogous to flipping in 2D), because in general mirroring is not physically plausible in 3D. The area-based term is correspondingly modified to a volume-based term. Several examples of 3D dissection are shown in Figure 1, 6 and 7. Note that our implementation currently does not consider how the pieces can be locked together to form a stable 3D structure. If a piece has no sufficient support underneath it, the structure will not be physically stable. Although the examples shown in this paper have not encountered such issue, it remains a direction for future research.

3.6 Dissecting Multiple Figures

Finally, we present an extension of our algorithm to simultaneously dissecting multiple figures. Note that a trivial approach is to simply overlay the pairwise dissection solutions, and output the intersections of all pieces. Unfortunately this will produce a large number of pieces that are overly fragmented – the upper bound is exponential with respect to the pairwise dissection results.

Here we achieve multi-figure dissection by adapting our optimization based algorithm. From Figure 4 we can see that the primary steps at each level of the algorithm consist of 1) cluster initialization and 2) cluster refinement. Below we discuss how these two steps are modified for a multi-figure setting respectively. The matching M is still computed from a bipartite graph between a pair of figures. It is possible to redefine M based on the complete k-partite graph among all k figures, but computing the maximum weighted matching for such a graph is known to be an NP-hard problem.

Multi-figure cluster initialization. At level 1, the initial two clusters for each figure are created in the same way as before, i.e. each figure independently creates two random clusters using a flood-fill on random seeds. At each subsequent level, the initial clusters are computed using pairwise sub-cluster refinement. Specifically, we first pick a random figure as the pivot figure. Without loss of generality, let's assume the pivot figure is $\mathcal{A}$. Next, we select a piece from $\mathcal{A}$ that has the worst total matching scores with all other figures, and split it as well as its matched pieces in all other figures into two sub-clusters. These sub-clusters then need to be refined, for which we use the same back-and-force process as before, except that we now perform one iteration of refinement at a time, between the pivot figure and other figures in a round-robin fashion.

Multi-figure cluster refinement. As before, during global refinement, our goal is to modify the clusters in order to achieve improved matching result. To do so, we again select one figure as the pivot figure, then perform the matching and copy-paste from the pivot figure to all other figures. Next, we loop over all figure to perform random label switching. Here the candidates for label switching in a given figure is the set of all pixels that have at least one mismatch with any other figure. Once this is done, we proceed to the next figure as the pivot. Thus the original forward vs. backward refinement in the two-figure setting is now generalized to the multi-figure setting, where each figure will be used as the pivot once to perform a forward refinement with other figures.

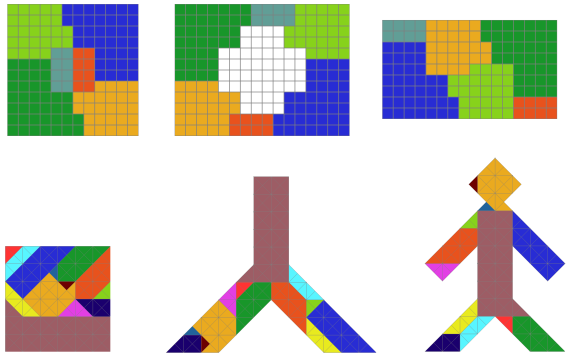


Figure 8: *Examples of three-figure dissections. The top row shows a 6-piece dissection of a square, a rectangle with a cross hole, and a solid rectangle. The bottom row shows a 13-piece dissection of a square, the Chinese character for 'person', and a figure of a person.*

Figure 8 shows two examples of three-figure dissection results. Note that these results achieve perfect dissections between all three figures. Since the computation for multi-figure dissection is more expensive, the running time is considerably longer than before.

3.7 Implementation Details

Algorithm implementation. A 2D figure is loaded from a binary image and stored as a 2D array. We represent a piece using the STL's `set` data structure. The matching M between two clusters needs to be evaluated frequently, and we employ an optimized Bellman-Ford algorithm to quickly compute it. It is stored as a bidirectional list together with the transformations defined for each pair of pieces. We store the triangular lattice using a 2D array as well, where an array element stores the four triangle pixels, each at a different orientation. A 3D shape is loaded from a binary image that represents each slice of the shape, and is stored a list of 2D arrays. Finally, we parallelize the random exploration step using multiple threads, since each exploration is independently computed. This allows us to achieve linear speedup using a multicore CPU.

Figure editing interface. As our method requires the two input figures to contain equal number of pixels (or equal number of voxels in 3D), we implemented a simple user interface to assist the editing of input figures if necessary. For a user-provided figure, we first rasterize it onto the lattice grid, then allow the user to directly edit each pixel individually. Alternatively, the user can create a figure from scratch in the interface, similar to editing a standard binary image. The program reports the total number of pixels covered by each figure to facilitate pixel counting.

Physical implementations. There are several ways to manufacture the dissection puzzles we created. If a square lattice is used, we can build the resulting pieces using Lego bricks, which are easy to construct. For triangular lattice or 3D dissections, we produce the puzzles using 3D printing. Figure 1 shows several examples of physically produced puzzles.

4 Results and Discussions

Optimality. To examine the optimality of our algorithm, we compared our solutions for several representative dissection problems with the reference solutions described in Frederickson's book entitled *Dissections: Plane and Fancy* [1997]. These examples are demonstrated in Figure 11. As shown in the left column, all input figures are rectilinear polygons of integer coordinates, which can be exactly represented using a square lattice. Thus they provide a direct evaluation of our method. The optimal solutions for these

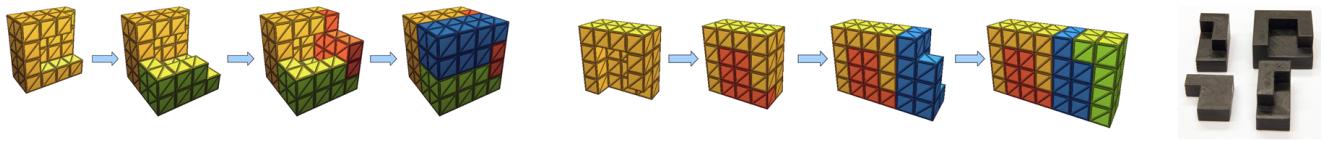


Figure 6: *An example of 3D shape dissection. The first input is a 4^3 cube with a 2^3 cavity at the center, and the second is a 7x4x2 cuboid. The solution contains 4 pieces shown on the right. The two sets of images on the left show the assembly of the pieces into each input shape.*

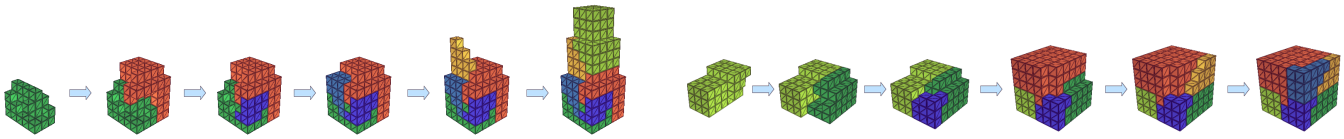


Figure 7: *A 3D shape dissection that illustrates $3^3 + 4^3 + 5^3 = 6^3$. The solution contains 6 pieces that are shown in Figure 1 example three.*

examples are known and are listed in the right column of the figure. The middle column shows our solutions. Several inputs contain disconnected components, for which our algorithm can handle successfully. For all examples we achieve the same number of pieces with the reference. Note also that for many of them, our solutions are different from the reference (i.e. in the shape of the resulting pieces). The examples in Figure 2 and 4 were also produced using our algorithm, and the results are known to be the optimal.

Performance. Our results were obtained on an Intel Core i7 2.66 GHz CPU with 6 GB RAM and 8 hyperthreads. For relatively simple shapes, such as the 2D examples in Figure 1 and Figure 11, the total computation time is within 20 minutes. The figures in these examples generally contain 50∼160 pixels. Higher resolution input will result in increased computation time, but we have found that the cost is more dependent upon the number of levels (hence the number of pieces) required to solve a dissection and less dependent on the number of pixels. This is mainly because each higher level needs to process more clusters. In addition, multi-figure dissections generally take much longer time to run. For example, our longest computation time is 5 hours for the three-figure Chinese character dissection in Figure 8, which produced 13 pieces. All other 2D examples were computed within half an hour. For 3D dissection, the example in Figure 6 took 7 minutes to run, and the one in Figure 7 took about an hour.

Two-figure dissections. Figures 2, 4, 5, 11 all demonstrate examples of two-figure dissections using a square lattice. In Figure 1, the first row shows a dissection between a rectangle and a square with a cross hole, and the second row shows a dissection between a rectangle and a rasterized octagon. Figure 9 shows two-figure dissections computed using a triangular lattice.

Area-based term. In Figure 5 we have shown that enabling the area-based term often leads to results where the size of each piece is more balanced. In Figure 10 we shown an additional example where our algorithm has found multiple solutions at the final level. The best solution can be selected as the one that gives rise to the smallest area variance of the pieces. Other criteria can also be used to define the best solution.

Three-figure dissections. Figure 8 shows two examples of three-figure dissections. The first is a 6-piece dissection of a 12×12 square, a 16×12 rectangle with a cross hole in the center, and a 16×9 solid rectangle. The second is a three-figure dissection of a square, a Chinese character meaning 'person', and a simple figure of a person. Our algorithm found a 13-piece dissection of this example. Many Chinese or Japanese characters are hieroglyphic, thus they are suitable for creating dissection puzzles as the character look similar to the figure it represents.

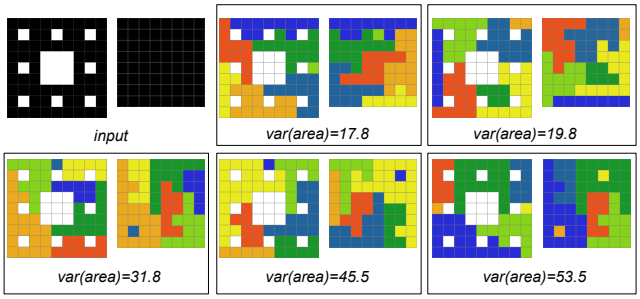
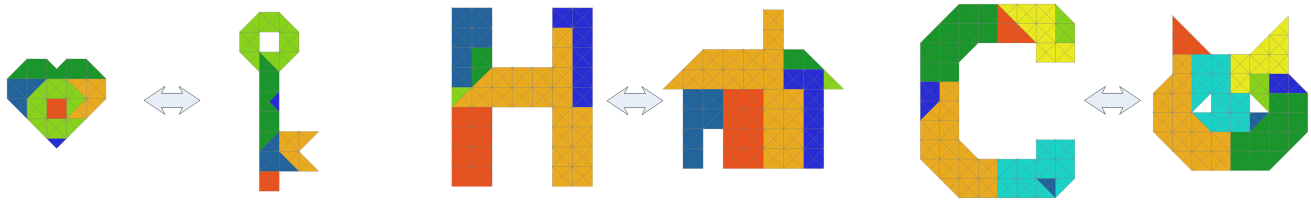


Figure 10: *An example where the algorithm found multiple solutions with equal number of pieces. The input are a 9×9 Serpenski carpet and a 8×8 square. Five selected solutions are shown, all of which are 7-piece solutions. We calculate the area variance for each. Smaller variance corresponds to a more uniform/balanced size, which is generally preferable for aesthetic reasons.*

3D dissections. Figures 1, 6 and 7 demonstrate 3D puzzles created using our algorithm. In particular, the third row in Figure 1 is inspired by the 2D Pythagorean triples and demonstrates $3^3 + 4^3 + 5^3 = 6^3$; and the fourth row is the dissection of a polycube Bunny model and a $6 \times 6 \times 7$ cuboid. We have found these puzzles to be quite enjoyable and challenging to play with. Some of them look deceptively simple, but can take a considerable amount of time to solve.

Limitations. One of the main limitations of our method is that due to discretization, many input figures cannot be exactly represented onto a discrete lattice grid. They have to be rasterized, resulting in approximate shapes. Therefore our method is not meant to substitute analytic approaches to many dissection problems, especially those involving regular polygons. Nonetheless, for the purpose of generating puzzles, we have found the approximate shapes are sufficient in many cases. Furthermore, our results may provide insights and useful initial solutions for discovering an analytic dissection.

Another limitation is that the user is currently given little control over the algorithm, other than adjusting the area-based term. Thus it's difficult to constrain the solution to have certain desirable properties. One example is the symmetry of the pieces, which is often desirable from an aesthetic point of view. We have not considered such properties during the solution process. However, as a dissection problem often has multiple solutions, such as shown in Figure 10 and 11, it's possible to account for these properties when selecting the final best solution. An alternative way is to include a symmetry-based term in the objective function in order to actively enforce such constraints.



(a) Heart to Key (6 pcs) *(b) H to House (6 pcs)* *(c) C to Cat (8 pcs)*

Figure 9: *Three examples of two-figure dissections using the triangular lattice. The examples in (b) and (c) dissect an English letter with an object figure whose name starts with that letter.*

5 Conclusions and Future Work

In summary, we have presented an efficient computational algorithm to compute geometric dissections. We extended our algorithm to incorporating area-based weight, to triangular lattice, to dissecting multiple figures, and finally to dissecting 3D shapes. We believe our algorithm and extensions provide a convenient tool for users to design a variety of different geometric puzzles.

In terms of applications, the ability to create dissection puzzles itself presents an interesting application for educational and entertainment purposes. There are other practical applications. For example, the 3D extension of our algorithm may be used to solve manufacturing problems, such as decomposing a furniture into as few pieces as possible to fit in a specific packaging box. Another example is to design furniture that can transform between different shapes to provide multiple functions.

In future work, besides addressing some of the limitations discussed in Section 4, we plan to explore a few additional directions. First, we plan to investigate how to design 3D puzzles that can be interlocked with each other, providing a stable physical structure. Second, we plan to incorporate user-specified constraints into the design. For example, we can allow the user to specify certain parts of the input that must remain integral pieces, thus preventing them from splitting. It is also possible to include a symmetry-based term, similar to our area-based term, in order to favor solutions with more symmetric pieces. Finally, by implementing the algorithm on modern GPUs, we hope to gain significant performance speedup towards interactive design of puzzles.

References

ABBOTT, T. G., ABEL, Z., CHARLTON, D., DEMAINE, E. D., DEMAINE, M. L., AND KOMINERS, S. D. 2008. Hinged dissections exist. In *Proc. of SCG*, 110–119.

AKIYAMA, J., NAKAMURA, G., NOZAKI, A., OZAWA, K., AND SAKAI, T. 2003. The optimality of a certain purely recursive dissection for a sequentially n-divisible square. *Comput. Geom. Theory Appl. 24*, 1, 27–39.

ALLMAN, G. J. 1889. *Geometry from Thales to Euclid*. Hodges, Figgis, Dublin.

BOLYAI, F. 1832. *Tentamen juventutem. Typis Collegii Reformatorum per Josephum et Simeonem Kali*. Maros Vasarhelyini.

CHU, H.-K., HSU, W.-H., MITRA, N. J., COHEN-OR, D., WONG, T.-T., AND LEE, T.-Y. 2010. Camouflage images. *ACM Trans. Graph. 29*, 4, 51:1–51:8.

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.

COHN, M. J. 1975. Economical triangle-square dissection. *Geometriae Dedicata 3*, 4, 447–467.

CZYZOWICZ, J., KRANAKIS, E., AND URRUTIA, J. 1999. Dissections, cuts, and triangulations. In *Proc. of the 11th Canadian Conference on Computational Geometry*, 154–157.

CZYZOWICZ, J., KRANAKIS, E., AND URRUTIA, J. 2007. Rectilinear glass-cut dissections of rectangles to squares. *Applied Mathematical Sciences 1*, 52, 2593–2600.

DEHN, M. 1900. Über den rauminhalt. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 345–354.

DUDENEY, H. E. 1902. *Puzzles and prizes*. Weekly Dispatch, April 6–May 4.

FREDERICKSON, G. N. 1997. *Dissections: Plane and Fancy*. Cambridge University Press.

FREDERICKSON, G. N. 2002. *Hinged Dissections: Swinging and Twisting*. Cambridge University Press.

FREDERICKSON, G. N. 2006. *Piano-hinged Dissections: Time to Fold!* A K Peters.

FREDERICKSON, G. N. 2007. Unexpected twists in geometric dissections. *Graph. Comb. 23*, 1, 245–258.

FU, C.-W., AND LEUNG, M.-K. 2005. Texture tiling on arbitrary topological surfaces using wang tiles. In *Proc. of EGSR*, 99–104.

GERWIEN, P. 1833. Zerschneidung jeder beliebigen anzahl von gleichen geradlinigen figuren in dieselben stücke. *Journal für die reine und angewandte Mathematik (Crelle's Journal) 10*, 228–234.

GRÜNBAUM, B., AND SHEPHARD, G. C. 1986. *Tilings and patterns*. W. H. Freeman & Co.

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

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

KRANAKIS, E., KRIZANC, D., AND URRUTIA, J. 2000. Efficient regular polygon dissections. *Geometriae Dedicata 80*, 1, 247–262.

LAGAE, A., AND DUTRÉ, P. 2006. An alternative for wang tiles: colored edges versus colored corners. *ACM Trans. Graph. 25*, 4, 1442–1459.

LAGAE, A., AND DUTRÉ, P. 2007. The tile packing problem. *Geombinatorics 17*, 1, 8–18.

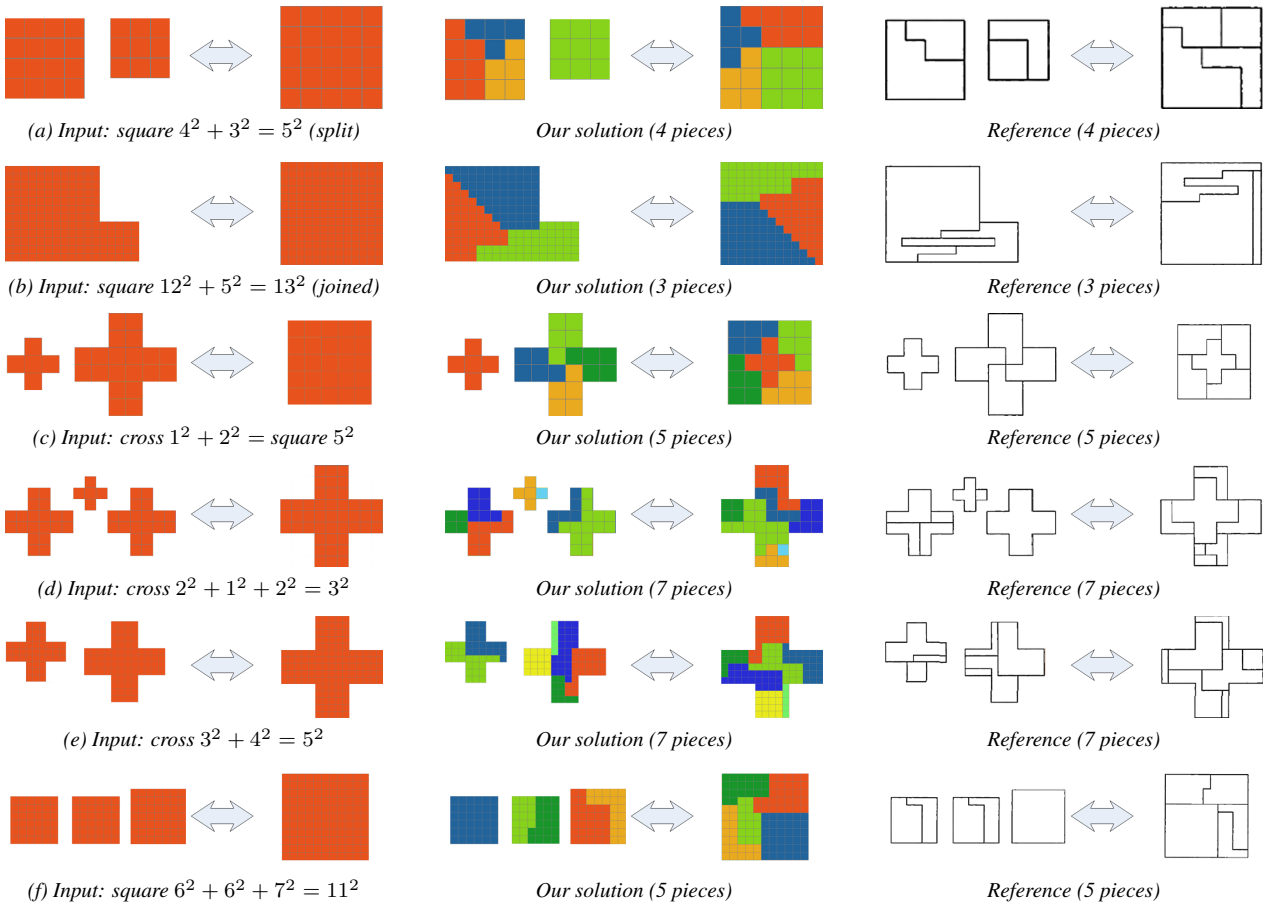


Figure 11: *A comparison of our solutions with reference solutions described in [Frederickson 1997]. Some of these examples are visualizations of the Pythagorean triple numbers. The left column shows the input, the middle shows our solution, and the right shows the reference solution. For all examples we achieve the equal number of pieces with the reference.*

LAGAE, A., KAPLAN, C. S., FU, C.-W., OSTROMOUKHOV, V., AND DEUSSEN, O. 2008. Tile-based methods for interactive applications. In *ACM SIGGRAPH 2008 classes*, 93:1–93:267.

LI, X.-Y., SHEN, C.-H., HUANG, S.-S., JU, T., AND HU, S.-M. 2010. Popup: automatic paper architectures from 3D models. *ACM Trans. Graph. 29*, 4, 111:1–111:9.

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

LINDGREN, H. 1972. *Recreational Problems in Geometric Dissections and How to Solve Them*. Dover Publications.

LO, K.-Y., FU, C.-W., AND LI, H. 2009. 3D polyomino puzzle. *ACM Trans. Graph. 28*, 5, 157:1–157:8.

LOWRY, M. 1814. Solution to question 269, [proposed] by mr. w. wallace. *Leybourn, T. (ed.) Mathematical Repository 3*, 1, 44–46.

MITANI, J., AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graph. 23*, 3, 259–263.

MITRA, N. J., AND PAULY, M. 2009. Shadow art. *ACM Trans. Graph. 28*, 5, 156:1–156:7.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. 23*, 3, 488–495.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Trans. Graph. 26*, 3.

ÖZDURAL, A. 2000. Mathematics and arts: Connections between theory and practice in the medieval islamic world. *Historia Mathematica 27*, 2, 171–201.

SLOCUM, J. 2003. *The Tangram Book*. Sterling.

WALLACE, W. 1831. *Elements of Geometry (8th ed.)*. Bell & Bradfute, Edinburgh.

WEST, D. B. 2000. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall.

XIN, S.-Q., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND DANIEL, C.-O. 2011. Making Burr puzzles from 3D models. *ACM Trans. Graph. 30*, 4, to appear.

XU, J., AND KAPLAN, C. S. 2007. Image-guided maze construction. *ACM Trans. Graph. 26*, 3.

XU, X., ZHANG, L., AND WONG, T.-T. 2010. Structure-based ascii art. *ACM Trans. Graph. 29*, 4, 52:1–52:10.