Creating Personalized Jigsaw Puzzles

Cheryl Lau Yuliy Schwartzburg Appu Shaji Zahra Sadeghipoor Sabine Süsstrunk
École Polytechnique Fédérale de Lausanne (EPFL)

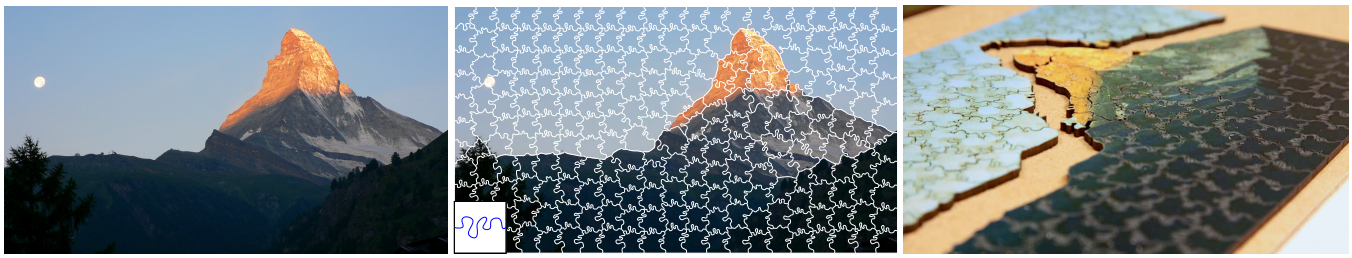


Figure 1: *Our tool creates custom jigsaw puzzles that are aesthetically interesting and challenging to assemble. Taking as input a user-defined curve that describes the general shape of a piece edge, our method optimizes for puzzle cuts that follow the main color lines of the image, subject to interlocking, intersection, and minimum width constraints. The resulting puzzle is physically realizable; it can be fabricated by current laser cutters and assembled and disassembled multiple times.* Original image courtesy of Flickr user outdoorPDK.

Abstract

Designing aesthetically pleasing and challenging jigsaw puzzles is considered an art that requires considerable skill and expertise. We propose a tool that allows novice users to create customized jigsaw puzzles based on the image content and a user-defined curve. A popular design choice among puzzle makers, called *color line cutting*, is to cut the puzzle along the main contours in an image, making the puzzle both aesthetically interesting and challenging to solve. At the same time, the puzzle maker has to make sure that puzzle pieces interlock so that they do not disassemble easily.

Our method automatically optimizes for puzzle cuts that follow the main contours in the image and match the user-defined curve. We handle the tradeoff between color line cutting and interlocking, and we introduce a linear formulation for the interlocking constraint. We propose a novel method for eliminating self-intersections and ensuring a minimum width in our output curves. Our method satisfies these necessary fabrication constraints in order to make valid puzzles that can be easily realized with present-day laser cutters.

CR Categories: I.3.3 [Picture/Image Generation]: Line and curve generation—; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

Keywords: jigsaw puzzles, aesthetics, fabrication, interlocking constraint, self intersection elimination in planar curves, minimum width enforcement

1 Introduction

We propose a tool that, according to the user's design choices, automatically creates a jigsaw puzzle based on the image content. By cutting the puzzle pieces along the image's high contrast edges, or color lines, we generate aesthetically pleasing and challenging puzzles. This technique, called *color line cutting*, is a common practice among professional scroll saw puzzle makers as it makes the puzzle more interesting and increases the value of the puzzle [Armstrong 1997]. As an aesthetic choice, the designer can cut along important color contours to highlight salient objects in the resulting puzzle pattern. However, existing techniques are manual while our optimization framework aligns the cuts to the colors lines automatically. Additionally, cutting along color boundaries eliminates local contrast making it harder for the assembler to find neighboring pieces; thus, it adds a level of difficulty to the puzzle. Figure 1 shows a jigsaw puzzle created by our algorithm whose pieces align with the contours of the mountain to create an aesthetically interesting composition.

Jigsaw puzzles have been played with since 1760 when John Spilsbury created the first jigsaw puzzle out of a map of Europe for the purpose of teaching children geography [McAdam 2014] (Figure 2(a)). Since then, jigsaw puzzles have become popular for recreation, and a market for custom jigsaw puzzles now exists. Taking advantage of the recreational allure of puzzles, some companies make customized puzzles as corporate gifts to advertise their business. People often want to create custom jigsaw puzzles from personal photos to create personalized memorabilia or to commemorate a special occasion such as a wedding. Custom puzzles are often designed independent of the image content or are artist-designed by hand. Ravensburger, a major manufacturer, uses the die-cutting method to make their standard jigsaw puzzle shapes. According to hand-drawn templates, experts bend metal sheets which are then used to punch out puzzles like a cookie cutter [Ravensburger AG 2008]. Scroll saw puzzle makers rely on their expertise to manually carve out individual pieces by twisting and turning the wood around a thin blade. Figure 2(b) shows an example of a puzzle hand-cut with a scroll saw. Some puzzle makers use laser cutters or water jet cutters to cut their puzzles but still rely on artistic skills to design the cuts. We create puzzles based on the underlying image without the need of an expert's skills.

To create a valid puzzle that is physically realizable and playable, we ensure that the puzzle cuts satisfy the fabrication constraints of current laser cutters and that the pieces are robust enough to handle multiple assembly and disassembly. To this end, we formulate an interlocking constraint, and we present a method for eliminating self-intersections and ensuring a minimum width (to prevent narrow

(a) first puzzle (b) scroll saw cut puzzle

Figure 2: ***The first jigsaw and a scroll saw cut puzzle.*** *John Spilsbury is credited with making the first jigsaw puzzle (a) out of a map of Europe. Puzzle maker Charles Hamm hand-cut this puzzle (b) with a scroll saw.* Image (a) courtesy of © British Library Board. Maps 188.v.12. Image (b) courtesy of Charles Hamm.

sections). Interlocking pieces do not separate easily, so interlocking is a desirable quality in making sure the completed puzzle holds together as a single unit. However, color line cutting often sacrifices the interlocking ability of the pieces as image contours are not often interlocking. To handle the tradeoff between color line cutting and interlocking, we let some pieces in the puzzle be interlocking while letting others follow color lines as long as those that follow color lines are indirectly interlocked to the rest of the puzzle. Additionally, we devise a linear formulation of the interlocking constraint which we can easily enforce during the puzzle cut optimization. After optimization, we apply a post-processing step to enforce the self-intersection and minimum width constraints. In order to keep as much of the optimized curve as possible, we only change the curve locally as necessary to remove self-intersections or widen a narrow part. These constraints allow us to design puzzles that we can fabricate with a laser cutter.

Contributions. We summarize our contributions as follows:

- Our method produces puzzles that follow color lines to produce aesthetically interesting puzzles based on image content.
- We propose a tool for generating and rendering images as custom jigsaw puzzles which are realizable through fabrication and can handle multiple assembly and disassembly.
- Our method handles the tradeoff between color line cutting and interlocking. We propose a novel interlocking constraint and introduce a linear formulation of the constraint to easily incorporate it into the optimization.
- We present a robust method for eliminating self-intersections and ensuring a minimum width in our output pieces' curves.

2 Related Work

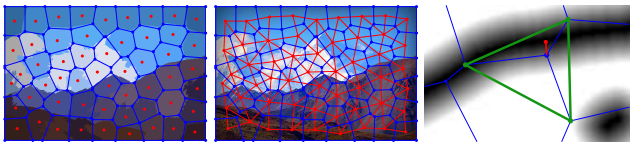
Design and fabrication tools. Do-it-yourself design and fabrication tools are becoming more and more popular. There are tools for designing your own plank-based furniture [Umetani et al. 2012], garments [Umetani et al. 2011], shadow art sculptures [Mitra and Pauly 2009], plush toys [Mori and Igarashi 2007], beadwork models [Igarashi et al. 2012], and paper pop-ups [Li et al. 2011; Iizuka et al. 2011]. With these tools, the user can design their desired object, and the system will figure out how to satisfy all the appropriate constraints. All of these methods aim to achieve user goals while handling necessary constraints. These systems need to satisfy fabrication constraints in order to produce stable and durable furniture or pop-up books that can close flat. Along the same lines, we present a tool for users to design jigsaw puzzles, and our method ensures that interlocking, intersection, and minimum size constraints are met.

Puzzle creation. Several design tools for creating puzzles exist. Methods for creating different types of 3D puzzles include those of Lo et al. [2009] who create 3D polyomino puzzles, Xin et al. [2011] who create Burr puzzles, and Song et al. [2012] who create recursive interlocking puzzles. Related to jigsaw puzzle creation is the creation of image mosaics [Kim and Pellacini 2002] where the goal is to pack a set of arbitrarily-shaped image tiles into an arbitrarily-shaped container image. Instead of arranging image tiles to line up with the container image boundaries, we want to arrange puzzle pieces to align with the image contours. Also similar to our problem is image guided fracture [Mould 2005] which can generate a crack pattern that follows text from a binary image. Although their approach aligns cell boundaries to the image content, their modified Voronoi construction does not produce convex cells necessary for enforcing our puzzle constraints. Companies offering custom puzzles services and online custom puzzle making websites exist, but these services and tools either only provide a set of limited puzzle patterns or require an artist's manual labor. Even Mathematica has a post about how to create jigsaw puzzles [Mathematica StackExchange 2013], and Photoshop has a jigsaw puzzle texture file that can be layered on top of an image, but these approaches do not take into account the image. Companies such as Liberty Puzzles, Wentworth Wooden Puzzles, and Artifact Puzzles sell laser-cut wooden puzzles, but these companies have artists help with the design. A company called Nervous System creates jigsaw puzzles automatically using a crystal growth simulation method yielding pieces with a characteristically wiggly shape. The extent to which puzzle creating tools are available in the marketplace enforces the already existing demand for tools such as ours. In contrast to these existing tools, with our method we determine the puzzle cuts based on the underlying image.

Interlocking and self intersection constraints. Finding interlocking solutions and eliminating self intersections in planar curves are problems that have been studied previously. Schwartzburg and Pauly [2013] generate 3D models from interlocking planar pieces. The 3D puzzles of Xin et al. [2011] and Song et al. [2012] also create interlocking assemblies. For jigsaw puzzles, we use a 2D notion of interlocking. Pekerman et al. [2008] propose a method for detection and eliminating self intersections in curves and surfaces. Our problem only requires self intersection removal for planar curves, and we take inspiration from Pekerman et al.'s swapping algorithm. Instead of solving for a new curve after each self intersection removal as they do, we adopt a simple reversal method based on reversing the order of points along the curve. This allows us to fix the intersection locally and keep as much of the original curve as possible.

3 Graphical User Interface and User Curve

We present a graphical user interface in which the user can specify the number of pieces in the puzzle, draw a sample user curve, and choose the amount of interlocking in the puzzle. The user curve will define the general shape of the puzzle curves. We obtain the user's sample curve by fitting a cubic B-spline (with a uniform knot vector) to sample points along the path drawn by the user. We assume the user curve is an open curve that does not self-intersect. Users can also choose the amount of interlocking in the puzzle by specifying the minimum number of neighbors with which a piece should be interlocking. The user specifies the number of pieces, draws a sample curve, and chooses the degree of interlocking, and the algorithm designs a customized puzzle based on the user's selected image. The tool outputs a vector graphic of the puzzle cuts which can be sent to a laser cutter.



(a) superpixels & graph (b) graph & weak dual (c) graph adjustment

Figure 3: ***Graph creation.*** *We create a Voronoi graph (blue) over the image using superpixel centers (red in (a)) as the Voronoi seeds. The superpixels result (a) yields regions whose boundaries follow color lines in the image. The Voronoi graph represents the puzzle cuts while its weak dual (red in (b)) represents the puzzle pieces. We adjust the graph in (c) by moving the blue vertex to its new location (red) on its closest color line (black). Restricting the vertex to the polygon (green) created by its neighbors preserves cell convexity.* Original image courtesy of Flickr user blmiers2.

(a) superpixels (b) color lines (c) distance field

Figure 4: ***Color lines.*** *We extract important color lines (black) from superpixels (a). Color lines separate regions with a color difference above a threshold. We use the distance field (c) thresholded to a maximum distance of 20 pixels during our optimization to pull curves towards color lines.* Original image courtesy of Flickr user blmiers2.

4 Method

Our method consists of three steps. First, we roughly place puzzle pieces according to color regions in the image. We represent the puzzle with a graph whose edges correspond to the puzzle cuts. Then, we solve for the puzzle cuts along each graph edge to determine the puzzle piece shape. For each edge, we optimize for a curve that matches the user's sample curve and follows color lines while satisfying intersection, minimum piece size, and interlocking constraints. Finally, we remove any self-intersections and modify the curve to obey minimum width constraints during a post-processing step. The result is a puzzle design that is built upon the user's design choices and that satisfies the necessary constraints for making a valid puzzle.

4.1 Puzzle Piece Placement

We roughly position the n puzzle pieces desired by the user according to the color regions in the image with the goal of aligning the pieces to color boundaries. First, we segment the image into n color regions using Achanta et al.'s [2012] superpixels method (with a compactness of 30). Superpixels group pixels into Voronoi-like regions that align with color boundaries as in Figure 3(a). Using superpixels allows us to find separations between color regions in the image even when no specific edge exists. For example, the Van Gogh and Seurat paintings in Figure 14 do not have explicit edges separating the stars from the sky or the man from the background, respectively. A standard edge detector may give us too many edges, such as those surrounding each paint stroke, whereas superpixels give us the edges between color regions at a slightly higher semantic level. Then, using the superpixel centers as seed points, we create a Voronoi graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose cells approximately follow the color edges in the image. Each Voronoi cell represents a puzzle piece, and the edges $\mathcal{E}$ of the graph represent the puzzle cuts that separate pieces. The weak dual, or Delaunay triangulation constrained to the image borders, of this graph will then represent the puzzle pieces and their neighbors as shown in red in Figure 3(b).

4.2 Color Lines

Color lines are high contrast edges in the image that we want our puzzle cuts to follow. Since superpixel boundaries align with color boundaries, we also use superpixels to determine the color lines in the image. However, since the desired number of puzzle pieces can vary, we apply the superpixels algorithm again but this time with a fixed number of 200 superpixels in order to make sure there is enough oversegmentation to obtain a precise boundary. Although many neighboring superpixels will have neighbors of the same color, between pairs of dissimilar colors, the superpixel boundaries will align with all the high contrast edges in the image. Therefore, we extract a binary color line map from superpixel boundaries between neighboring superpixels with a difference in average color above a threshold of 25 in CIE LAB space. Figure 4(b) shows the color lines extracted from the superpixel boundaries in 4(a). To produce a guidance image for the snake-like optimization, we compute a distance field to the color lines which will encourage curves to follow the color lines. The distance field is thresholded at a maximum distance d_c so that only curves that are already close to a color line will try to follow it. Figure 4(c) shows the clipped distance field $\mathbf{I}$. For the examples in the paper with approximate resolutions of 1000×700, we set the number of superpixels to 200 and $d_c = 20$ pixels, but for significantly higher resolution, these parameters should be increased.

Some Voronoi graph vertices may not exactly line up with color lines but may be close. Since puzzle cuts go through the graph vertices, we adjust those vertices to get puzzle cuts that follow the color lines more closely. For each vertex in $\mathcal{V}$ that is less than d_c away from a color line, we project that vertex to its nearest color line pixel as shown in Figure 3(c). Restricting the vertex to the hull of its neighboring vertices preserves the convexity of the original Voronoi cells, a property we rely on when establishing our linear intersection constraints.

4.3 Puzzle Curve Optimization

Once the puzzle pieces are approximately placed to follow color lines, we solve for the puzzle cuts that determine the pieces' shapes. For each edge in the graph $\mathcal{G}$, we solve for a curve that closely matches the user curve, follows color lines, and satisfies constraints. The intersection, minimum piece size, and minimum width (between two curves) constraints are represented as linear bounds in the optimization while the self intersection and minimum width (between points on a single curve) constraints are enforced during a post-processing step.

We model the optimized curve for each edge of a puzzle piece as a cubic B-spline curve of the form $\mathbf{Bx}$, where the columns of $\mathbf{B} \in \mathbb{R}^{2n_s \times 2n_c}$ contain the cubic B-spline basis functions for the n_c control points, sampled at n_s uniform parameter values along the curve, and $\mathbf{x} \in \mathbb{R}^{2n_c}$ is a vector which specifies the locations of the control points. Similarly, n_s sample points along the B-spline for the user curve is $\mathbf{Bx}_u$, where $\mathbf{x}_u$ contains the control points of the user curve.

We have two principal objectives for our optimization. First, the optimized curve should match the user curve as closely as possible. Since we are only concerned about matching the shape of the curve and not the position and scale of the curve, we minimize the distance

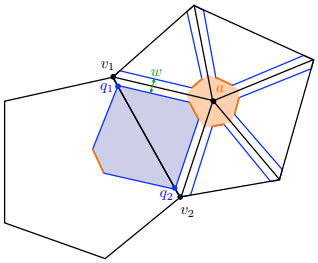


Figure 5: ***Intersection, min piece size, and minimum width (between two curves) constraints.*** *Each curve with endpoints (v_1, v_2) is restricted to the shaded blue area delineated by four linear intersection and minimum width constraints (blue) and two linear minimum piece size constraints (orange).*

between the first derivatives of the two curves at the n_s sample point locations. Varying the position and scale allows individual pieces to have different forms which facilitates assembly. The second objective is to make the optimized curve follow the color lines as closely as possible. We solve for the optimized curve's control points $\mathbf{x}$ by minimizing the following:

$$\begin{aligned} &\text{minimize} \quad \left\|\mathbf{B}'\mathbf{x} - \mathbf{B}'\mathbf{x}_u\right\|^2 + \lambda \sum_{i=1}^{n_s} \mathbf{I}(\mathbf{B}_i\mathbf{x}) \\ &\text{subject to} \quad \mathbf{A}\mathbf{x} < \mathbf{b}. \end{aligned} \tag{1}$$

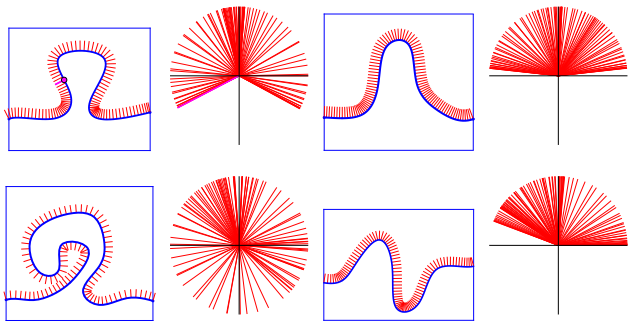
$\mathbf{B}'$ is a matrix that defines the tangent curve. The matrix $\mathbf{B}' \in \mathbb{R}^{2n_s \times 2n_c}$ is defined as $\mathbf{B}' = \frac{1}{h}\mathbf{T}\mathbf{D}$ where the columns of $\mathbf{T} \in \mathbb{R}^{2n_s \times 2(n_c-1)}$ contain the quadratic B-spline basis functions, the matrix $\mathbf{D} \in \mathbb{R}^{2(n_c-1)\times 2n_c}$ takes the differences between adjacent control points, and h is the knot spacing between the uniformly sampled parameter values. $\mathbf{I}$ is the clipped distance field to the color lines. $\mathbf{B}_i \in \mathbb{R}^{2\times 2n_c}$ contains the rows in the B-spline basis function matrix $\mathbf{B}$ corresponding to the ith sample point. λ is a weight on the color lines term. The color lines term is modeled using active contours [Kass et al. 1988], guided by $\mathbf{I}$, to pull the curve towards the color lines. Minimizing an energy function of these two objectives does not guarantee a valid curve, so we impose constraints on the curve in the form of linear constraints $\mathbf{A}\mathbf{x} < \mathbf{b}$ as described in Sections 4.3.1 and 4.3.2 and a post-processing step described in Section 4.5.

4.3.1 Intersection and Minimum Piece Size Constraint

When optimizing for a curve, we need to make sure adjoining puzzle cuts do not intersect and are separated by at least a minimum width of w units[1]. We do this by restricting each cut to lie within the blue shaded area in Figure 5. We construct this area by partitioning each convex cell by connecting each vertex to the piece center. We offset these lines inwards by $\frac{w}{2}$ to make sure a curve does not come within w of a neighboring curve. To prevent the curves from chiseling out large portions of a puzzle piece and ensuring that each piece has a minimum area of at least a units, we do not allow the curves to enter the orange similar polygon with area a centered at the cell center.

Each curve must start and end at the endpoints (q_1, q_2) and is restricted to the area (shaded in blue) formed by the six linear bounds in $\mathbf{A}\mathbf{x} < \mathbf{b}$. After the optimized curve is checked by the self-intersection and minimum width process, the curve is extended to (v_1, v_2) and smoothed. We do not optimize edges shorter than $4w$ because the curve bounds will not be valid, and we set their output polylines to (v_1, v_2). These linear constraints, while conservative, make each curve independent of each other making it possible to process them in parallel.

[1] When laser cutting our puzzles, we calibrate physical measurement (in millimeters) to pixel size, allowing us to specify widths in both units.



(a) interlocking (b) Gauss map (c) not interlocking (d) Gauss map

Figure 6: ***Interlocking curves.*** *Interlocking curves (a) have unit normals that span more than 180° in their Gauss maps (b). The magenta point on the curve prevents the bottom piece from moving in the direction of the normal and thus prevents the piece from sliding downwards. Curves that are not interlocking (c) do not have normals that span at least 180° in their Gauss maps (d).*

4.3.2 Interlocking Constraint

We define a curve as *interlocking* if the two pieces it joins are not able to slide apart once put together. Figure 6 shows examples of interlocking and non-interlocking curves.

The interlocking constraint necessitates that the piece cannot move tangentially to any point on the curve. For this to be true, the angular span of unit normals in the Gauss map of interlocking curves must be strictly greater than 180° degrees, whereas the angular span of normals of non-interlocking curves is not [do Carmo 1976]. This span could be calculated by integrating curvature and removing overlapping angular spans. However, this is a highly nonlinear operation and difficult to enforce as a constraint to our optimization. Instead, we enforce that the Gauss map of the optimized curve should have a similar topology to that of the user curve. Specifically, we find the axis crossings of the first derivative of the user curve and aim to have similar axis crossings in the optimized curve. The user

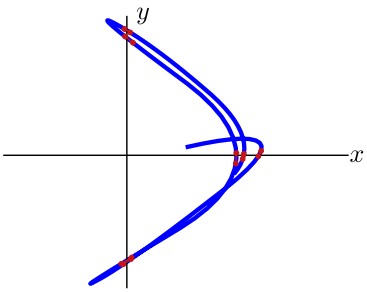
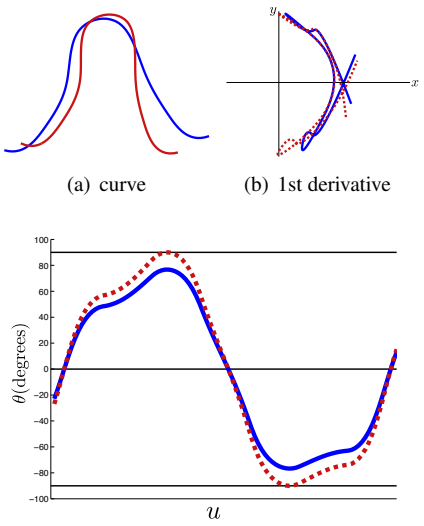


Figure 7: ***Linear interlocking constraint.*** *We find the axis crossings (red) in the first derivative of the user curve and find sample points bounding the axis crossings. We constrain the corresponding points in the optimized curve to their respective quadrants.*



(a) curve (b) 1st derivative

(c) tangential angles

Figure 8: ***Make curve interlocking.*** *We make the original non-interlocking curve (blue) interlocking (red) by stretching its tangential angles (c) so that its first derivative (b) can meet the required span of* $180^\circ + \epsilon$.

curve is rotated until at least one axis is crossed in both its positive and negative directions. In Figure 7 we find sample points bounding the axis crossings (red). We constrain the corresponding points in the optimized curve to stay within the same quadrants as the sample points, and we add these linear constraints to $\mathbf{Ax} < \mathbf{b}$. This is a reasonable assumption since matching the resultant curve to the user curve is also one of our goals.

Make User Curve Interlocking. To formulate the interlocking constraint, we assume that the user curve is already interlocking. If the user curve is not initially interlocking, we make it so by forcing its tangential angles to span $180^\circ + \epsilon$, where $\epsilon = 0.001$. From the user curve's first derivative, we obtain a continuous function of the curve's tangential angles centered on 0°, scale it to the required angular span (Figure 8(c)), recalculate the first derivative from the scaled tangential angles (Figure 8(b)), and solve for a new curve that has this first derivative (Figure 8(a)).

4.4 Indirect Interlocking

Sometimes puzzle designers sacrifice the interlocking ability of a single piece in order to follow an important color line that is not interlocking. With indirect interlocking, we can still make sure that the puzzle as a whole is interlocking. As long as every piece is connected in a single graph, then the puzzle is indirectly interlocking. We determine which edges should remain interlocking and which edges are free of this requirement by computing the minimum spanning tree $\mathcal{M}$ of the weak dual graph $\mathcal{D}$ (red in Figure 3(b)). Each edge in $\mathcal{D}$ connects neighboring puzzle pieces and corresponds to a pair of superpixels. Edges in the minimum spanning tree represent connections between pieces that need to be interlocking in order for the entire puzzle to be connected. We bias the interlocking to occur between the pieces with the lowest contrast. We assign higher edge weights to edges that align with important color lines. Edge weights are calculated as the difference in average color in CIE LAB space between two neighboring superpixel regions, normalized over all edges. The minimum spanning tree guarantees that each node has at least one neighbor, but for added stability the user can increase the minimum required degree per node, thereby increasing the amount of interlocking but possibly at the expense of following color lines (Figure 13). To create a minimum graph with a degree higher than one, we go through the vertices in $\mathcal{D}$ in order of descending importance (where importance is a vertex's highest edge weight) and add the edges necessary to fulfill the minimum degree requirement. The additional edges are added from lowest to highest weight until either the degree requirement is met or all neighbors of the piece have been added. Since it is hard to solve for an interlocking curve that also obeys the minimum width on short edges, we remove edges that are less than a small fixed percentage of the image size before we create the minimum graph.

We achieve indirect interlocking by enforcing interlocking constraints during optimization only on edges in $\mathcal{M}$. While this ensures a connected surface, the optimized curve may not obey the minimum width constraint, and when it is post-processed to satisfy the minimum width, it may then violate the interlocking constraint. If this happens, we reduce the weight λ on the color lines term in Equation 1. Normally, λ is set to the importance value of the edge. If there are still edges in $\mathcal{M}$ that cannot fulfill the interlocking constraint after λ becomes 0, we recalculate the minimum graph without these edges and re-optimize for the modified edges.

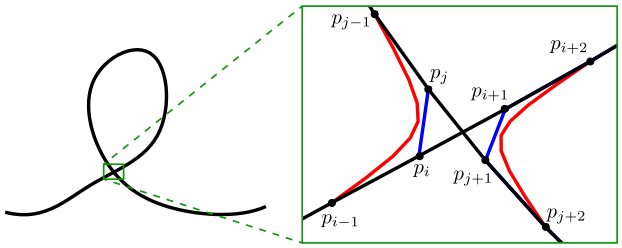


Figure 9: ***Fixing self intersections.*** *A self intersection occurs in this simple loop curve between segments* (p_i, p_{i+1}) *and* (p_j, p_{j+1}). *We fix the self intersection by replacing the intersecting segments with new segments* (p_i, p_j) *and* (p_{i+1}, p_{j+1}) *(blue). We then reverse the order of points in the loop. The modified segments (blue) are subdivided and smoothed using Laplacian smoothing to get the smooth output (red).*

4.5 Post-Processing

4.5.1 Self Intersection Constraint

As a post-processing step, we make sure that the curves generated by our algorithm do not self-intersect. However, since we have optimized the curve according to an objective function, we devise a technique to fix self-intersections that modifies the rest of the curve as little as possible.

For efficiency, we first represent our curve as piecewise linear. We then detect self-intersections using line segment intersection. In Figure 9, line segment (p_i, p_{i+1}) intersects with segment (p_j, p_{j+1}). To fix the intersection, we eliminate the intersecting segments and instead join p_i to p_j and p_{i+1} to p_{j+1}. We then change the topology of the curve by reversing the sequence of points between p_j and p_{i+1}. In order to have a smooth curve, we apply Laplacian smoothing in the local region around the fixed self-intersection.

4.5.2 Minimum Width Constraint

Even if a puzzle piece boundary is not self-intersecting, if it is too thin in some areas, part of the puzzle piece can break off. Thus we introduce a minimum width constraint on the curve. This ensures

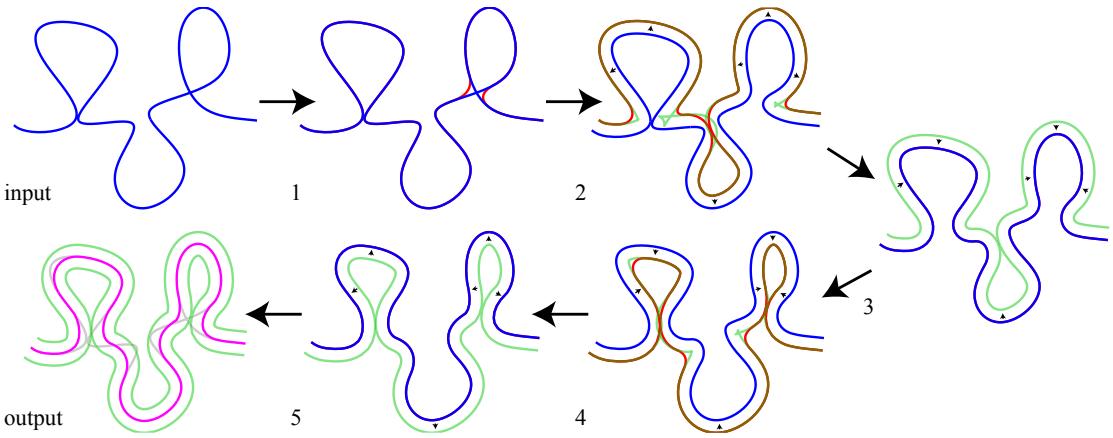


Figure 10: ***Removing self-intersections and ensuring a minimum width.*** *(1) Fix intersections in the input curve (blue). (2) Fix intersections in the positive offset (green). (3) Offset back in the negative direction, and fix intersections (none in this case). (4) Fix intersections in the negative offset (green). (5) Offset back in the positive direction, and fix intersections (none in this case) to get the output curve (pink). Our method only changes the curve locally where the constraints are violated, keeping as much of the input curve (gray) as possible.*

that any point on a curve must be at least the minimum width w away from any other point on the curve along its normal. We convert the minimum width problem into a self-intersection problem of the curve's offsets. An offset from a curve is defined as the set of points traced by the normal vector of length equivalent to the offset distance. If a curve's offsets at distances $\pm\frac{w}{2}$ away do not self-intersect, then the curve satisfies the minimum width constraint. In general, the positive and negative offsets also cannot intersect each other. This could happen if the curve swirls back onto itself and terminates in the middle of the swirl. However, this will not happen in our case because our linear bounds will not allow the curve to swirl around an endpoint. The positive and negative offsets o of curve f at offset distance of $\frac{w}{2}$ are defined as

$$o = f \pm \frac{w}{2} n \tag{2}$$

where n contains the unit normals to the curve, calculated as unit tangent vectors rotated 90°.

After optimizing each curve, we apply the following six step method for eliminating self-intersections and ensuring a minimum width in planar curves. Figure 10 shows the result of this process. For stability we apply a smoothing after each operation on the modified segments.

1. Fix self-intersections of the optimized curve according to Section 4.5.1.
2. Take the positive offset of the fixed curve. Trim local intersections. Fix self-intersections in the positive offset.
3. Offset back in the negative direction. Trim local intersections. Fix self-intersections in the offset back.
4. Take the negative offset of the curve from (3). Trim local intersections. Fix self-intersections in the negative offset.
5. Offset back. Trim local intersections. Fix self-intersections in the offset back.
6. Iterate steps (1) to (5) using the curve from (5) in place of the optimized curve until there are no intersections in the curve or its offsets.

There are two types of self-intersections in offset curves: local and global. Local intersections occur when the offset distance is larger than the radius of curvature of the original curve, while global intersections occur when two points on the curve are closer than the minimum width [Maekawa 1999]. For local intersections, we trim the curve at the point of intersection. For global intersections, we do not want to trim away the curve because that could mean trimming away a significant portion of the curve that was optimized to follow color lines and match the user curve. In this case, we fix these intersections by removing the self-intersections in the offset curve.

A self-intersection occurs when the curve C is equivalent at two different parameter values such that $C(u) = C(v)$. In order to eliminate all self-intersections in one traversal of the curve, we process the self-intersections using a priority queue ordered by the second curve parameter v. This also guarantees that the resulting curve is a single connected curve that does not split into two parts. Experimentally we found that 10 iterations were enough to fix most curves. However, for short curves particularly, it is possible that we reach no valid solution. In this case, we set that curve to the edge endpoints (v_1, v_2). Figure 11 shows the result of enforcing the self intersection and minimum width constraints on example curves. Our method preserves as much of the optimized curve as possible by only changing the curve locally where necessary.

5 Results

We designed several jigsaw puzzles with our tool. Figure 12 shows how the user curve allows the user to customize the style of the cut influencing the overall appearance of the puzzle. The user can draw or select a curve that matches the image aesthetics. For example, the user can draw a jagged shape for the cat puzzle to emphasize the

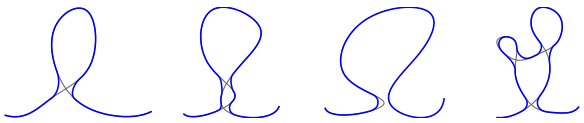


Figure 11: ***Example curves with self-intersections removed and narrow parts widened.*** *Our robust method eliminates self-intersections (first and second) and widens parts that are too narrow (third). The original curves (gray) are fixed locally in the offending regions, leaving the rest of the curve unmodified.*

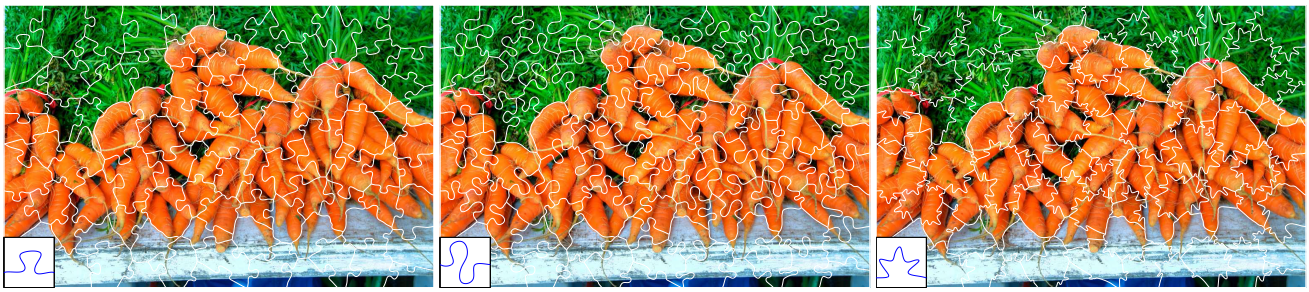
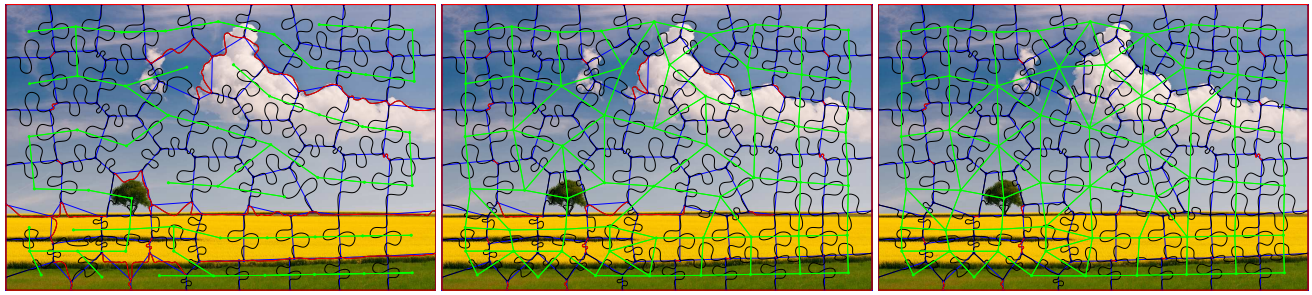


Figure 12: ***Different user curves.*** *These puzzles are created from different user curves. As a result, they have different characteristic piece shapes.* Original image courtesy of Flickr user Ed Yourdon.



(a) min degree=1 (b) min degree=4 (c) min degree=6

Figure 13: ***Different degrees of interlocking.*** *The user can control the amount of interlocking in the puzzle by setting the minimum degree for each piece. A minimum degree of one results in a puzzle with minimal interlocking connections while a degree of six creates a fully interlocking graph. The minimum graph (green) designates interlocking connections (white). Non-interlocking curves (red) are not in the minimum graph and have more freedom to follow color lines. (Note that pieces may not satisfy the degree requirement if they do not have enough neighbors with shared edges long enough to support an interlocking curve.)* Original image courtesy of Flickr user skoeber.

angular shape of the cats ears, or the user can draw a curvy shape to complement the swirling sky in Van Gogh's *Starry Night* (both in Figure 14). Figure 13 shows the tradeoff between color line cutting and the amount of interlocking in the puzzle. As the minimum degree increases from 1 to 6, edges are added to the minimum graph (green). The non-interlocking cuts (red) along the clouds in (a) are interlocking in (c) but no longer closely follow the outline of the clouds. Out of the examples in the paper, we have physically fabricated eight puzzles, and each one has been disassembled and assembled multiple times. Figure 14 shows a variety of jigsaw puzzles and the final laser cut puzzles created from different images with varying number of pieces and different user curves. Our puzzle cuts follow the contours of the arch, trace the cat's silhouette, and isolate the stars in Van Gogh's *Starry Night*. Note that our method finds the main color lines in the Van Gogh and Seurat paintings even though hard edges separating these regions are not explicit. Also note that for a given piece, it is often not clear how many neighbors it has and along which section of the pieces boundary each neighbor will connect. This affords the assembler a different kind of challenge than a uniform grid-like puzzle would.

6 Limitations and Future Work

There are more design choices in puzzle making that can, in the future, be added to our tool. First, we could add a choice of different pattern styles. The Voronoi graph controls the overall pattern of the puzzle pieces. By specially placing the sample points used to generate the Voronoi graph, we can create different pattern styles. For example, we could sample the image based on saliency to create more puzzle pieces in salient regions. Second, we could allow the user to draw or select a set of input curves instead of just one. Then, we could build a basis space for the curves to allow more variations in output curve shapes. Third, we could add *whimsies* to the puzzle by setting their boundaries as hard boundaries in the graph. Whimsies are specially shaped pieces often of objects that match the theme of the image [Liberty Puzzles 2014]. These would allow the users to further customize their puzzle.

7 Conclusion

We present a tool for designing customized jigsaw puzzles based on the image content and a user curve. Our method solves for puzzle cuts that follow color contours in the image, creating aesthetically interesting and challenging puzzles. We laser cut eight example puzzles to show that our puzzle designs are physically realizable and playable. At least 15 people have played with the puzzles, and the feedback we received was that the puzzles were interesting, very different from traditional puzzles, fun, and challenging. The users not only remarked that the quality of the puzzle pieces is on par with or superior to commercially available puzzles, but they were also very fascinated with the intricate shapes and patterns of the puzzle pieces we could generate. By making sure the puzzle is at least indirectly interlocking through regions of lowest contrast, we handle the tradeoff between color line cutting and interlocking. We enforce interlocking where necessary with our novel linear constraint. The concept of interlocking is not just applicable to puzzles; it can also be used in carpentry to ensure proper alignment of pieces that will be glued together. After optimization, we eliminate self-intersections and ensure a minimum width with a novel process that only modifies curves locally where necessary, retaining as much of

the optimized curve as possible. Our self-intersection and minimum width methods can also be applied to other curve processing domains such as any kind of laser cutting, stenciling, or cutout animation. In CAD modeling for manufacturing, the width of the physical cutting device needs to be taken into account, necessitating also a minimum width constraint. Our resulting jigsaw puzzles follow color lines, match the general shape of a user-drawn curve, and satisfy the interlocking constraint as well as the necessary fabrication constraints.

Acknowledgements

The authors would like to thank the following people for useful discussions and their help in proofreading the paper and putting together puzzles: Anne Jorstad, Stefan Lienhard, Jay Unnikrishnan, Mario Deuss, Romain Testuz, Bin Jin, Aurélien Lucchi, Bailin Deng.

This research has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 257453; COSYM.

References

ACHANTA, R., SHAJI, A., SMITH, K., LUCCHI, A., FUA, P., AND SÜSSTRUNK, S. 2012. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 11, 2274–2282.

ARMSTRONG, B. 1997. Jigsaw puzzle cutting styles: A new method of classification. *American Game Collectors Association, Game Researchers' Notes 25*.

DO CARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces*. Pearson Education Canada.

IGARASHI, Y., IGARASHI, T., AND MITANI, J. 2012. Beady: Interactive beadwork design and construction. *ACM Trans. Graph. 31*, 4, 49.

IIZUKA, S., ENDO, Y., MITANI, J., KANAMORI, Y., AND FUKUI, Y. 2011. An interactive design system for pop-up cards with a physical simulation. *The Visual Computer: Int. J. of Comput. Graph. 27*, 6–8, 605–612.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

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

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, 98.

LIBERTY PUZZLES, 2014. About us. Accessed June 2014. `http://www.libertypuzzles.com/about`.

LO, K.-Y., FU, C.-W., AND LI, H. 2009. 3d polyomino puzzle. *ACM Trans. Graph. 28*, 5, 157.

MAEKAWA, T. 1999. An overview of offset curves and surfaces. *Computer-Aided Design 31*, 3, 165–173.

MATHEMATICA STACKEXCHANGE, 2013. How can I calculate a jigsaw puzzle cut path? Accessed July 2013. `http://mathematica.stackexchange.com/questions/6706/how-can-i-calculate-a-jigsaw-puzzle-cut-path`.

MCADAM, D., 2014. History of jigsaw puzzles. American Jigsaw Puzzle Society. Accessed 2014. `http://www.jigsaw-puzzle.org/jigsaw-puzzle-history.html`.

MITRA, N., AND PAULY, M. 2009. Shadow art. *ACM Trans. Graph. 28*, 5.

MORI, Y., AND IGARASHI, T. 2007. Plushie: An interactive design system for plush toys. *ACM Trans. Graph. 23*, 3, 45.

MOULD, D. 2005. Image-guided fracture. In *Graphics Interface*, 219–226.

PEKERMAN, D., ELBER, G., AND KIM, M.-S. 2008. Self-intersection detection and elimination in freeform curves and surfaces. *Computer-Aided Design 40*, 2, 150–159.

RAVENSBURGER AG, 2008. Ravensburger values. Accessed July 2013. `http://www.ravensburger.com/us/about-ravensburger/brand-philosophy/ravensburger-values/index.html`.

SCHWARTZBURG, Y., AND PAULY, M. 2013. Fabrication-aware design with intersecting planar pieces. *Comput. Graph. Forum 32*, 2.

SONG, P., FU, C.-W., AND COHEN-OR, D. 2012. Recursive interlocking puzzles. *ACM Trans. Graph. 31*, 6, 128.

UMETANI, N., KAUFMAN, D., IGARASHI, T., AND GRINSPUN, E. 2011. Sensitive couture for interactive garment editing and modeling. *ACM Trans. Graph. 30*, 4.

UMETANI, N., IGARASHI, T., AND MITRA, N. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4, 86.

XIN, S., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3d models. *ACM Trans. Graph. 30*, 4, 97.

Figure 14: ***Puzzle cuts follow color lines.*** *Using different images and user curves, our method creates a variety of aesthetically interesting and physically realizable puzzles that follow high contrast color contours in images. The puzzle pieces resemble different user defined curves. Some of the final laser cut puzzles are also shown.* Original images courtesy of Wikipedia, Flickr users Arches National Park (Neal Herbert), blmiers2, Nattu, pelymo_05, R. J. Malfalfa, Sebastiàn Dario.