Tools for Physical Graphic Design

Liane Makatura* Dartmouth College
Hannes Hergeth† RWTH Aachen University
Danny M. Kaufman‡ Adobe Research
Wilmot Li§ Adobe Research
Emily Whiting¶ Dartmouth College

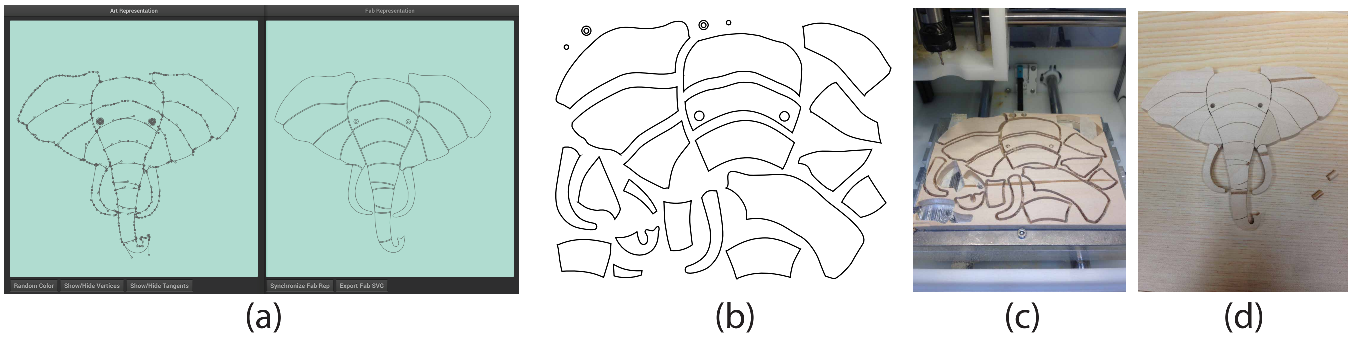


Figure 1: *An overview of our physical graphic design pipeline, which allows users to create complex physical artifacts from traditional vector graphic designs. Our current implementation (a) features side-by-side views of an editable artist representation (left), and the corresponding regions for fabrication (right). Once the user is happy with their design, it can be exported to an SVG, and (b) manually nested into a fabrication layout. Then, the user is able to (c) mill their design and (d) assemble their custom intarsia piece.*

Abstract

The expansion of rapid prototyping technologies offers unprecedented opportunity for custom manufacturing, yet viable functional design requires considerable time, effort, and domain-specific knowledge on the part of the designer. This is largely due to the fact that there is a fundamental discrepancy between the design and fabrication stages of this process: the underlying representations that allow for intuitive edits and constraint checking in the aesthetic design process are ill-suited for these challenges in the fabrication stage, and vice versa. However, this process is inherently a conversation between what is necessary for fabrication, and what is desired for design. Lack of suitable communication channels between the two often results in long, arduous feedback loops with lots of duplicated effort. We focus on bridging this gap by developing a system for the intuitive design and iteration of fabricable vector graphic illustrations. While there are many artistic endeavors that fall under this domain of physical graphic design (vector graphics meant for physical realization), this project focuses on developing a fabrication-aware regime for Intarsia, a traditional wooden mosaicing technique. We present a system which exploits the dual nature of fabrication-aware design to allow the artist to work in a traditional "art" workflow, while simultaneously viewing the effect of their edits in a synchronized fabrication representation.

1 Introduction

Vector graphics are very popular in traditional graphic design because of their flexibility in a number of different scales and contexts, such as digital displays or print media. However, these vector graphics can also be used to realize more complicated physical artifacts, such as stencils, laser-cut engravings, and even custom textiles or embroidery. The latter pipeline – using a traditional vector

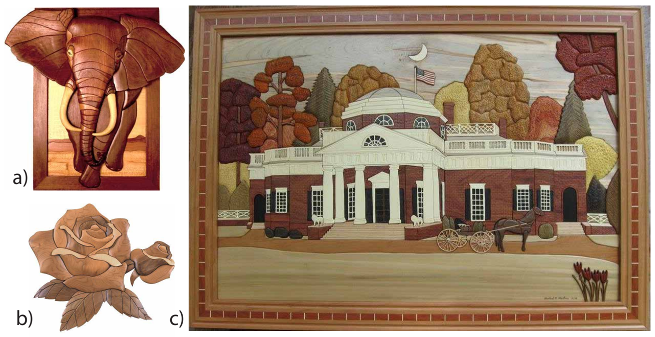


Figure 2: *Samples of professional intarsia pieces, featuring (a) "African Elephant" by Steve Bundred, (b) a standard beginner "Rose" pattern, and (c) "Monticello" by Mike Mathieu.*

graphic in order to realize a complex physical object – is what we refer to as *physical graphic design*. Currently, however, it is very difficult to translate arbitrary digital designs into successfully fabricated objects that resemble the original design. This is because the artistic process neglects to consider any design constraints that are imposed by the fabrication method itself. In this project, we attempt to define and improve upon the current state of the art in computational fabrication for one specific example of physical graphic design: intarsia.

Intarsia is a traditional woodworking technique that uses varied shapes, sizes, and species of wood fitted together to create a mosaic-like picture with an illusion of depth. After selecting the desired design and materials, an artist carefully arranges the individual pieces on wooden boards, seeking to minimize waste while exploiting natural properties such as color variation or grain patterns. Each piece is cut out with a scroll saw, before being sanded, shaped, and fitted to the remaining pieces. Intarsia is both labor- and time-intensive: "Monticello" (Figure 2) consists of over 800 individual pieces, crafted from 17 species of wood. A professional artist devoted over 150 hours to this piece over the span of two months.

Intarsia often requires the experience of a skilled craftsman, as the

*Liane.E.Makatura.17@Dartmouth.edu
†hannes.hergeth@rwth-aachen.de
‡kaufman@adobe.com
§wilmotli@adobe.com
¶emily@cs.dartmouth.edu

process is highly error prone. Each piece is manually fabricated independently of the others, yet they must fit together snugly in order to create the desired effect. This makes the art even more elusive for beginners, as even tiny deviations can compound in a big way in the final design.

However, intarsia designs are typically composed of a set of simple, closed regions. This simplicity holds even in the case of modest shaping along the top face, which is often meant to remove sharp edges, texture flat surfaces, and/or create a sense of depth within the piece. These surface details could be represented as a height field on top of the original boundary vectors. Thus, these patterns are well within the capabilities of rapid prototyping technologies like CNC milling. The use of such technologies could make the art accessible to novice users by reducing production errors, and decreasing the high time requirement of manual construction. For this reason, Intarsia is a great candidate for computational design and realization.

As previously discussed, this goal of realizing physically valid intarsia from traditional graphic designs presents a problem which is not yet understood. In particular, we must define a robust, intuitive process for translating between an artist's design and some representation which will be suitable for a computerized fabrication process. This latter stage requires a fundamentally different representation of the design, which is not readily accessible in the artist's original rendering.

To solve this problem, we conducted a thorough exploration of the design space, both theoretically and practically. Our theoretical exploration defines the dual representation between art and fabrication, seeking to understand the merits and limitations of each depiction. We then identify a practical approach to our problem, constructing a pipeline of existing technologies – such as Adobe Illustrator, Otherplan, and the desktop milling machine Othermill – in order to manually realize the process that we hope to compute. Finally, we present an interactive system which exploits our findings in order to expedite the process of iterative fabrication-aware design.

2 Related Work

Inspired by the growth of rapid prototyping technologies, researchers have poured significant energy into the creation of intuitive, fabrication-aware design paradigms. Several systems automatically optimize 3D models with respect to specific post-fabrication metrics like stability or adherence to a prescribed motion [Prévost et al. 2013; Bächer et al. 2014; Stava et al. 2012]. Several others seek constraint satisfaction through interactive design tools, which guide the user toward a physically plausible design while preserving the user's ownership of the design [Umetani et al. 2012; Yao et al. 2017]. All of these methods seek to ensure that the final product performs as desired, but they must also address (even if implicitly) the interesting design constraints posed by the fabrication process itself.

Such fabrication-aware approaches have also been applied to series of stacked or intersecting planes. In theory, this lower-dimensional space is easier for humans to decompose and reason about. However, research in this area has exposed many interesting problems that are not accessible without computational assistance. Examples include the intuitive modeling of planar structures [McCrae et al. 2014], the design of assemblable planar structures with non-orthogonal joints [Schwartzburg and Pauly 2013], and the interactive design and optimization of free-formed planar-piece model airplanes [Umetani et al. 2014].

Even in the case of 2-dimensional artifacts, the constraints imposed by fabrication can make it difficult to create valid designs by hand. One work considers the 2-dimensional craft of realizable stencil-design, where each color layer must form a single connected component while deviating as little as possible from the desired image [Bronson et al. 2008; Jain et al. 2015]. Another work creates custom image-based jigsaw puzzles, where piece contours mimic a user-defined profile, while also ensuring that the pieces are sufficiently interlocked [Lau et al. 2014].

In this body of work, one common challenge is the need to reconcile the fundamental discrepancy between the formats that are intuitive for object design or editing, and those which are necessary for object realization. A system for wire-wrapped jewelry automatically decomposes pixel or vector designs into a minimal set of continuous, low-curvature wire paths for fabrication [Iarussi et al. 2015]. We have also seen systems for garment design [Umetani et al. 2011], that enable synchronized, bidirectional editing between the traditional art and fabrication representations. This system exploits the complementary strengths of each representation, which provide intuitive solutions for complementary subsets of edit types. This emphasis on intuitive edits pervades in semantic shape editing [Yumer et al. 2015] and Lillicon [Bernstein and Li 2015], which enables users to directly edit the negative space created by explicitly represented objects. This is very similar to the goal of this project, as we consider simple regions enclosed by a series of open Bézier curves.

Our implementation is also largely enabled by the mathematical groundwork in planar mapping, a topic derived from graph theory. Since their introduction as a natural dual of explicit line representations [Baudelaire and Gangnet 1989], planar maps have been used extensively in computer graphics. Planar map representations typically destroy the original vectors defining the art, making them an undesirably ”one-way” operation, which does not allow for intuitive edits to the design after its conversion. The current state of the art in ”dynamic” planar map illustration, introduced in 2007, provides several heuristics which allow the user to alter the planar-mapped design with their original strokes [Asente et al. 2007]. An implementation of this concept can be found in Adobe Illustrator, under the name *Live Paint*. We make extensive use of this implementation, both in our initial problem exploration and our final comparison with existing methods.

3 Dual Design Representation

We consider the direct realization of an existing commercial pattern, with no edits. Vendors commonly distribute these patterns in the form of a PDF, so it is straightforward to extract an SVG of the design. Traditional hard copy patterns can also be processed into a vector graphic, by scanning the image then using standard techniques such as Canny edge detection [Canny 1986] to obtain the contours of each piece. However, in each of these cases, the patterns require additional processing to arrive at a fabricable design. In particular, we note that the patterns are often specified in an *artistic* view.

This ”art” representation follows the traditional paradigm for vector graphic design, where the image is comprised of many independent vectors that trace out semantically meaningful contours. They are constructed in such a way that it is easy for an artist to work with them: (a) the pieces are configured in their ”assembled” form, as they will appear in the final design, and (b) the contours are specified in a manner that caters to the designer's intent. The ”Art” representation is a very familiar workflow for artists, so there are obvious benefits for its use in our application.

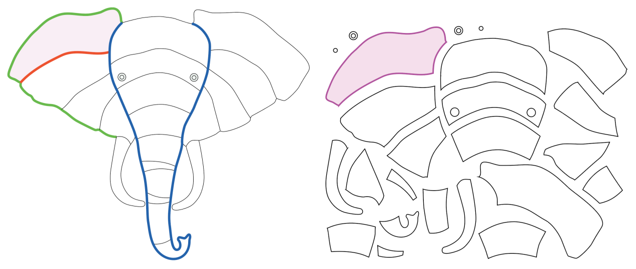


Figure 3: ***Left:*** *The "art" representation; note that the pink region is defined by the negative space enclosed by 3 separate vectors.* ***Right:*** *The "fab" representation; here, the pink region is explicitly represented by a single closed curve that can serve as a toolpath.*

It is also particularly adept for enforcing aesthetic constraints, or allowing for aesthetic edits such as:

- modifying (smoothing, simplifying, adding detail to) a boundary,
- multi-scale editing (i.e. moving an internal boundary but preserving the overall silhouette),
- maintaining perceived continuity across multiple pieces, and
- ensuring proper offsets of the boundaries, so pieces fit together with desired tolerance.

If we assumed that the lines between individual pieces were thick enough for a bit to pass through, we would be able to cut out the pattern precisely in this assembled form. However, this approach would leave large gaps between adjacent pieces, while prohibiting the use of multiple materials or artistic grain orientation. Since this method would impose far too many undesirable properties, we deem the "art" representation is unsuitable for direct computer-controlled fabrication.

Instead, we consider a representation that mimics the artisan interpretation: we prioritize each individual face of the design, explicitly encoding the closed regions themselves, rather than the boundaries between them. These closed regions can then be milled and assembled to form the final Intarsia piece. This "fab" representation is not intuitive for the kinds of aesthetic constraints mentioned above, but it is quite advantageous given our end goal of physical realization. Its similarity to the final fabricated output also makes it easier to detect and address issues with fabrication related edits and constraints, like the following:

- reducing high curvature regions that are too extreme for the endmill to cut,
- increasing space between pieces in the final fabrication layout, and
- dividing pieces that are too large for the given material.

These constraints are instrumental to the physical plausibility of the design, but they are woefully absent in the artistic representation. This forces the artist to follow all the way through with a design (often to the stage of fabrication) before discovering that it is infeasible for one of the reasons listed above.

The first strength of our approach is one that has been sought before: intentionally including, validating, and flagging violated fabrication constraints in the design process, so issues can be addressed before fabricating. This saves time, materials, and frustration on the part of the user. Moreover, intentionally utilizing both halves of this dual representation accounts for the fact that different edits and constraints are more easily executed or validated in different stages of the pipeline.

To fully harness the orthogonal strengths of these duals, we imagine a single editing paradigm that allows for direct, bi-directional communication between two presently incompatible stages of the design process. This will enable users to consistently work in the domain ("art" or "fab") that is most suited to a particular issue, entrusting our tool to propagate those changes back to the other representation for continued synchronicity.

As a step toward this goal, we present a system which dramatically improves the forward pass of this algorithm by presenting the two views side by side, and automatically propagating any edits made to the art view. This feedback can seamlessly inspire further edits in the art view, guiding the user toward a realizable design while preserving their full artistic control. Even in the absence of bidirectional editability, our system provides communication that makes the pipeline for fabricable design more intuitive and productive, shortening the iteration loop while moving one step closer to predictably fabricable graphic designs.

4 Implementation

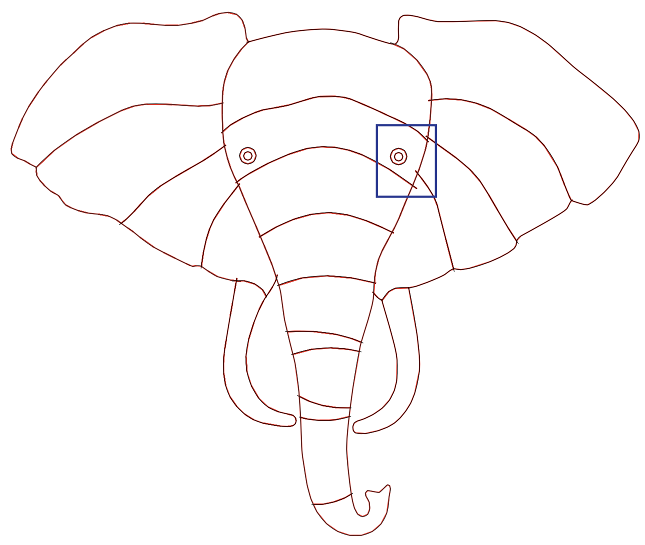
Our current implementation relies on real-time, continuous feedback to inform the user about the impact of their artistic edits in the fabrication domain. In order to realize this vision, we needed to implement both an intuitive user interface and a number of geometry processing operations which operate behind the scenes. In this section, we discuss our final implementation of these elements.

4.1 Geometric Operations

At each iteration of our algorithm, we must transform a set of Bèzier splines into a set of closed regions suitable for fabrication. To do this, the input SVG must be *(a)* flattened into a set of polylines, *(b)* mapped into a planar graph, and *(c)* offset along the normals of the boundaries. Each of these steps are described in detail below, accompanied by a motivation for their inclusion.

Bèzier Flattening Immediately after parsing the user-specified SVG, we approximate the design with a set of polylines. A polyline is a curve with C^0 continuity: that is, a curve composed of one or more line segments that are sequentially connected at their endpoints, but that need not share their tangent lines. To maintain analytic correspondences between the polyline endpoints and our original curve controls, we compute this polyline representation from the provided Bèzier curves using de Casteljau's recursive subdivision algorithm. Our subdivision terminated after the curve segments being approximated passed a simple flatness criterion [Fischer 2000].

There has also been further research on efficient polyline approximation for Bèzier curves, which improve upon the algorithm's stopping criterion to reduce the number of segments that are divided unnecessarily [Hain et al. 2005]. Optimizing our approximation could be an interesting extension in order to ensure that the final designs are as compact as possible for their specified accuracy. Fewer segments could boost the performance of several operations down the line, including tolerance offsets, validity checks, or future-endeavors such as 2D nesting (optimal layout) for the final fabrication configuration. However, we found that our implementation was generally sufficient for our needs.



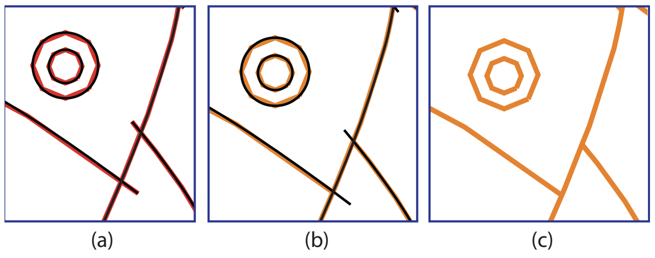


Figure 4: *Comparison between the provided bèzier curves (black) and our flattened polyline approximation (red). The three small regions display the effect of our sanitization scheme to remove the small overhangs from each intersection: (a) unsanitized output, where red precisely follows black, with overhangs included; (b) sanitized output, where the orange lines follow the black curves in all locations except in the overhangs; and (c) final sanitized output, showing show clean graph structure.*

The polyline conversion was motivated by our desire to build a fabrication-aware design tool. In order to accurately represent and analyze fabricable designs, it is necessary to consider the final output that will be passed to the machine. The relevant scheme for milling is g-code, a popular numerical control (NC) language that will transform the artist-drawn design into a series of discrete motions realizable by a machine. Permissible motions include those along line segments and circular arcs. Since this conversion would eventually affect the final result, we decided to work with a segment representation from the outset of our pipeline. This conversion also offered a number of other benefits which will be discussed throughout the paper. Thus, it was well worth the additional pre-processing time, and the slight deviation from the user-specified parameterization.

Planar Mapping We recall that our fabrication method requires a set of closed regions that can serve as the basis for a tool path. The polylines computed above are simply approximations to the original bèzier curves, so they remain firmly in the art view. To compute the fabrication view from this set of polyline curves, we employ planar maps.

Planar maps derive from the concept of planar graphs in graph theory. A *planar graph* is a graph for which there exists a 2-dimensional embedding of said graph such that no edge intersects with any other edge. A *planar map* is a function which computes a planar graph G' from a given set of edges and vertices, $G = (V, E)$. In our context, the input graph G is the set of polyline curves, which trivially has a fixed embedding in the 2D plane. Since edges can (and frequently do) cross one another in this given configuration, G need not be planar. Let us denote with P_{cross} the set of points at which two edges of G cross one another. Given our fixed 2D-embedding, it is then easy to see that the graph $G' = (V \cup P_{cross}, E')$ is planar, where E' reflects the edges of E that have been broken into smaller pieces to accommodate new vertices.

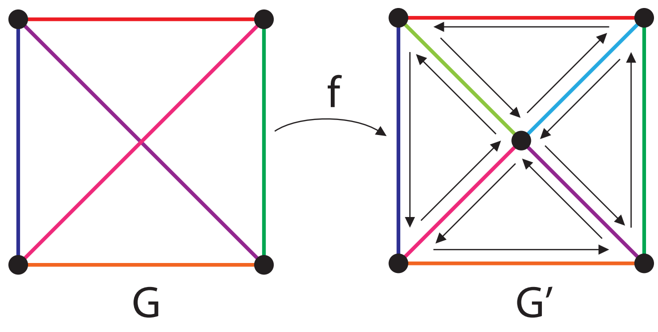


Figure 5: *We use planar maps to convert the artistic vectors to closed regions. A planar graph is a graph that can be embedded in a 2D space such that no edges overlap. The two diagonals in G violate this constraint, so G is not a planar graph. By applying the planar map function f, we can convert G into its corresponding planar graph, G'. We can then traverse directed cycles in the graph to obtain the counterclockwise-boundary (ccb) of each face.*

Our planar map algorithm does precisely this, adding a vertex into G' for every edge intersection point p which violates the planar graph condition in G. Since G' has a vertex defined at every intersection, it is now possible to relabel the graph edges in such a way that each face is defined by a simple, directed cycle. This cycle, called the *counter-clockwise boundary (ccb)*, forms the basis of our fabrication representation.

While they are conceptually simple, robust planar mapping algorithms are difficult to perfect: aside from Illustrator's proprietary *Live Paint* tool, there are very few tried and tested implementations. We compute planar maps using the open source Arrangement2 implementation found in the Computational Geometry Algorithms Library (CGAL) [Wein et al. 2017].

It is worth noting that the Arrangement2 library imposes some limitations on the project that directly affect the end user. In contrast to *Live Paint*, CGAL lacks a gap or tolerance parameter that allows the algorithm to infer intersections between curves that are within some small radius of one another. This means that the user is responsible for ensuring the explicit existence of all intersections intended in their design. While this is not a terrible imposition on the artist, it is counter to their normal mode of operation, in which aesthetics take priority. With this metric, junctions need only be *visually perceived*, as opposed to *analytically present*. These two metrics yield considerably different results, so the need to artificially extend each intersecting line segment conflicts slightly with our desire to preserve the artist's traditional design flow.

As discussed further in Section 7, the use of this library presented several additional challenges. However, in general, it suited the needs of our project well, and many of the inconveniences that remain can be addressed with clever heuristics in the pre- and post-processing stages.

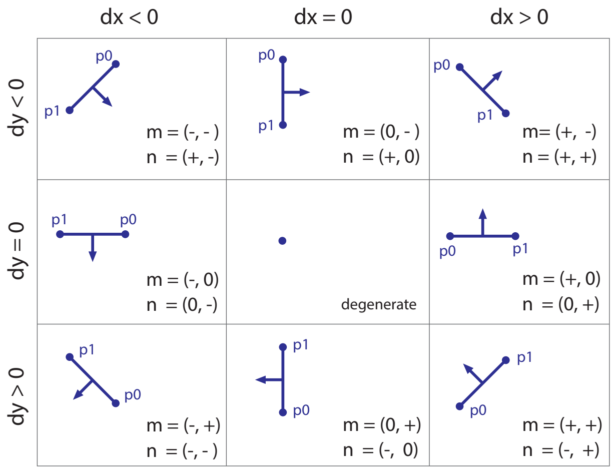


Figure 6: *To determine the normal along which to offset, we imagine walking along each segment (from $p0$ to $p1$) as part of a counter-clockwise trajectory. Here, we denote the slope m of each line, along with our desired normal, n. By comparing them, we observe that the desired normal direction is always $(-dy, dx)$, where dy is the signed vertical difference between the two endpoints, and dx is the signed horizontal difference.*

Boundary Offsets To ensure that the pieces will fit together properly once fabricated, we need to impose a slight boundary offset that shrinks each region by a given tolerance δ. This amount can be specified by the user, and it is relative to the absolute scale of the final constructed artifact.

To compute this offset boundary, we first consider naive rescaling of the piece. Purely convex regions can be shrunken effectively in this way, but this requirement is too restrictive for complex intarsia pieces. Instead, for each face we offset the segments of the counter-clockwise boundary (ccb) by shifting them δ units along their normals. Our desired normals will point toward the interior of the face. To determine the normal sign, we imagine walking along the segment's trajectory: since we are traversing the segments counterclockwise, we always want the normal pointing "left." As shown in Figure 6, this means our desired normal is always oriented along the direction $(-dy, dx)$, where dy is the signed vertical difference between the endpoints $p0$ and $p1$, while dx is the signed horizontal difference.

Once we determine the line $(y = mx + b)$ along which our offset segment should be placed, we need to determine the locations of the vertices. We cannot simply offset the existing vertices because the displacement along the normals may cause neighboring segments to intersect with, or become disconnected from, one another. To solve this, we compare adjacent segments' offset lines, and analytically solve for their intersection point, p. We place the intersection point between the two segments at p, extending or shortening the edge on either side accordingly, to ensure that our offset polylines maintain C^0 continuity.

4.2 Work Flow

In conjunction with these operations, we developed an interactive user interface that presents the dual representations side by side. The traditional art view is on the left, and the fabrication view is on the right. Once the user has populated the art view with their chosen SVG, our system automatically computes and displays the corresponding fabrication representation. As the user continues to edit their design in the art view, we provide real time feedback about the implications of their edits on the fabrication representation. Below, we present an overview of the system.

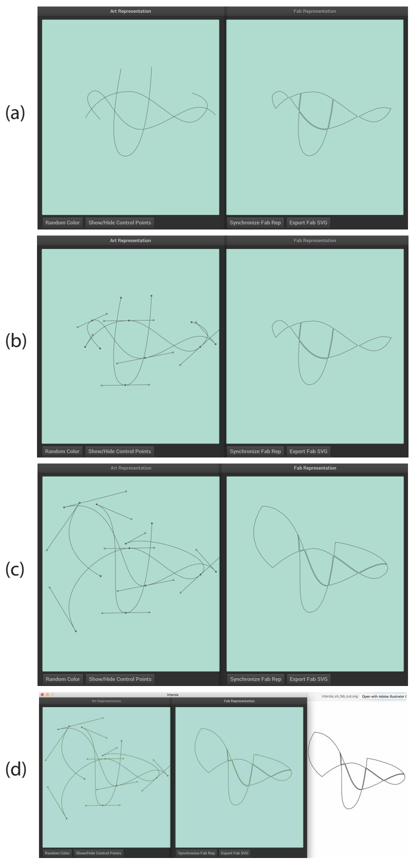


Figure 7: *Typical user flow for our system: (a) load in an SVG and immediately view the corresponding fabrication representation; (b) enable editing features by toggling the buttons at the bottom; (c) edit the art view (left), while receiving continuous synchronized feedback about the implications of those edits in the fabrication view (right). Finally (d) export an SVG, and verify that it matches the system's computed design.*

Data Input/Output Given the striking similarity between artists' intarsia patterns, machinable contours, and traditional vector graphic design, we chose to support SVG files for input and output. The CGAL Arrangement2 library does not natively support file input or output, so we built several parsers to interface between SVGs and several different types of arrangements, curves, and polylines. Our system leverages the open-source libraries NanoSVG and simple-svg for SVG reading and writing, respectively. The use of standard libraries ensures that our format is widely compatible with other software such as Adobe Illustrator for design, or Otherplan for CNC toolpath generation. This enables artists to easily shift among their favorite tools without any added hassle of file conversion.

Using the Tool Users primarily interact with our intuitive user interface, which was implemented using the NanoGUI library. Our application displays the two representations side by side, along with some additional functionality specific to each.

In the art canvas on the left, our system displays the original bèzier curves that were specified in the input file. We deliberately use this representation as opposed to its polyline approximation because the splines allow for easy, intuitive editing that conforms to the artist's

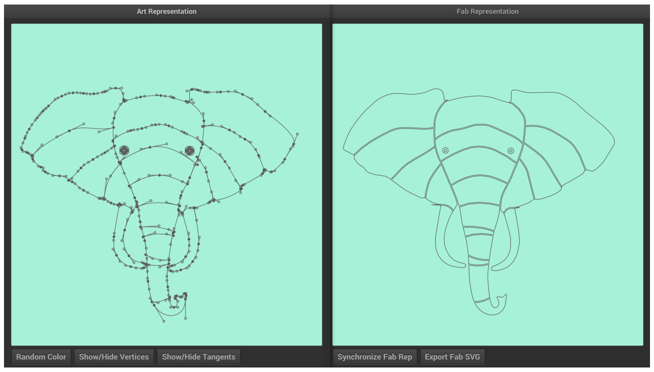


Figure 8: *Another UI example with more complicated input. Even with large designs, the system is efficient enough to handle edit propagation at interactive rates.*

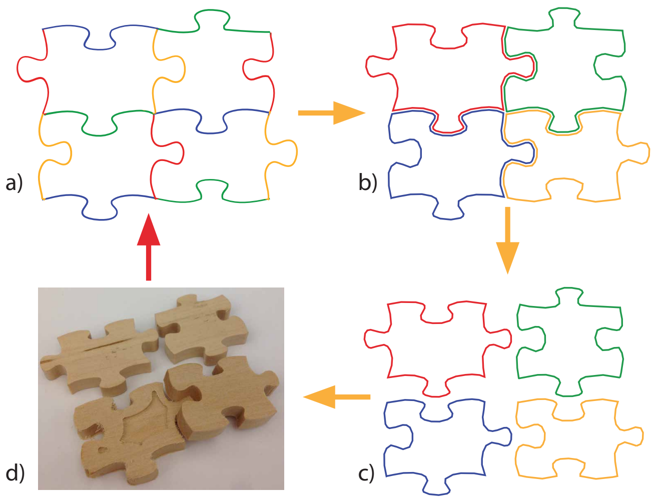


Figure 9: *General design pipeline for transforming arbitrary vector illustrations into fabricable designs. We start with (a) artistic splines that are semantically meaningful, but which fail to explicitly represent the boundary of the closed region we wish to fabricate. We then convert these splines to (b) explict boundaries of the closed regions we wish to fabricate, with the boundaries slightly offset to account for fabrication tolerances. Finally, we (c) shift the piece into a final fabrication layout, then (d) test our final result. If the design does not work (d), the user must return to (a) the original representation to fix the issue, then complete the process again. It often took many time-consuming iterations to reach a functional design.*

typical workflow. The art canvas provides the ability to independently toggle displays for the curve endpoints and control points. The former are displayed as solid-filled circles along the curve, whereas the latter are visualized as tangent lines with open circles. When one or both of these visualizations are displayed, the user can interactively edit any of the points by clicking and dragging, as in any other graphics software. Dragging a curve endpoint will also update its associated control points, to maintain whatever tangent continuity was previously specified. Motion of the control points is independent, so only the control point(s) under the cursor will be affected by any such edits.

Meanwhile, the fabrication canvas shows a polyline representation of the closed fabrication regions, after they have been computed using the algorithm outlined in Section 4.1. The updates to this fabrication view are relayed in real time, so the user is always able to visualize the impact of any art-view edits they may have made. Once the user is happy with their final design, the fabrication canvas provides an opportunity to export an SVG depicting the fabrication representation shown in the canvas panel.

Nesting Though it would be perfectly valid to load the exported file directly into a milling software such as Otherplan, this would be nearly akin to passing in the ”art” representation directly. This is due to the fact that our current implementation does not support automatic layout, or nesting. In this stage, the closed regions should ideally be shuffled from their ”assembled” configuration into a layout that is more suitable for fabrication. This layout must satisfy a number of physical constraints, such as (a) fitting all pieces within the material boundaries, (b) wasting as little material as possible, and (c) maintaining enough space between pieces for the bit to pass through. Presently, this layout optimization must be done by hand in Illustrator or something similar. In the future, this is something we would like to integrate into our system.

Fabrication Now, the file is ready for fabrication. The system could be tuned to any number of prototyping technologies, including CNC mills, laser cutters, or – with more drastic modifications – 3D printers. However, in our experiments, we focused on an implementation for the Othermill desktop milling machine. We generated our tool paths by loading the nested SVG file into Otherplan, and following the dialogs in order to properly set up the materials, tools, and toolpaths.

Though we used a single inexpensive wood for the experiments presented throughout this paper, it would be possible to spread the design's pieces out across multiple species to generate more interesting designs, akin to those in Figure 2. This idea, and other similar improvements, give way to a host of future directions that will be discussed shortly. Before this discussion, we provide a closer look at the results generated by our current implementation.

5 Comparison to Existing Technologies

Before designing our system, we explored several technologies and approaches that are currently available to artists in this domain. We found that clever use of tools like Adobe Illustrator, Otherplan, and the Othermill enabled the outcome we were hoping for. However, the process was tedious and non-intuitive: even simple cases (e.g. puzzle pieces) required tens of iterations over the course of several hours to arrive at a fabricable design. This experimental process is outlined below, as it was critical for our understanding of the problem statement, and the design of our technical approach as discussed in Section 4.

Much like our automated implementation, we begin the existing technology pipeline with a set of artistic vectors that must be transformed into closed, fabricable regions. As previously mentioned, we compute this planar map using Adobe Illustrators *Live Paint* tool. This process generates results similar to those of our system, providing explicit boundaries around each closed face, which could then be manually rearranged into a fabrication layout.

However, this direct approach neglects the fact that the boundaries must be offset slightly in the inward direction, ”shrinking” the faces to ensure that they fit together smoothly in the final construction. The two pipelines differ most in the way they address this issue of boundary offsetting. In our system, the boundary offsets are performed *after* the planar map has been computed. This is an intuitive approach, that mimics the manual (i.e., professional artisan) approach to intarsia: you first define the explicit boundaries of the

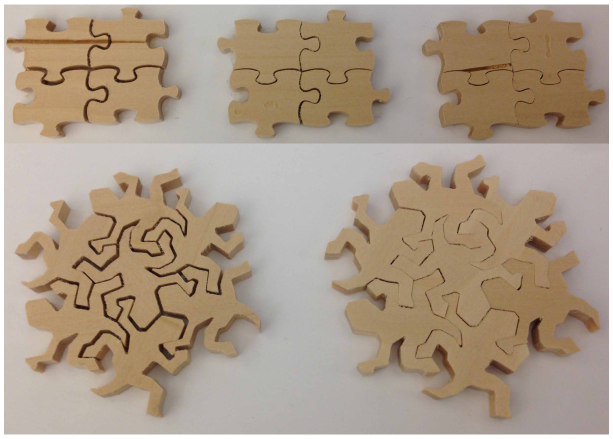


Figure 10: *CNC milled intarsia, with pattern design via the existing technology pipeline described in Section 5. Each pattern took several iterations to obtain an acceptably small tolerance between the pieces, while still ensuring that they would fit together. In this case, the rightmost designs are preferable, as their tight fit mirrors the aim of professional intarsia: consistent tolerances, with negative space no thicker than a business card in between. We used cyclic patterns for validation because they require high precision in order for the pieces to align properly with one another. This verified that the pieces were fitting together as expected, mitigating the chance that we were propagating any non-negligible errors throughout the design.*

regions you would like to fabricate, then you alter them slightly in order to ensure that they are perfectly aligned with the neighboring piece(s).

After computing the *Live Paint* regions, the pipeline of existing technologies fails to offer an intuitive way to complete this offset operation. While it is possible to offset the boundaries at this point in the process, it would either require clever application of existing procedures, or a considerable amount of manual manipulation. Instead, we rearranged our processing steps to make use of a common artist's hack. While still in the art view, we set the "stroke width" attribute of all vectors equal to 2δ, so that they were exactly twice as wide as the offset we desired. Then, we applied the outline stroke tool to the image, which generates explicit parameterizations for the profile of each vector's stroke. At any point along a visible spline, we can now imagine that there are three curves: (a) the "median" curve, which is the original vector parameterization that runs through the center of the stroke; and (b) the two "offset" curves, which trace along the extent of the displayed stroke width, forming a pair of "railroad tracks" that are consistently some distance δ away from the median, and 2δ away from one another. We then discard the original splines, and compute the *Live Paint* planar mapping on the offset curves themselves.

Figures 10 and 12 offer several examples of pieces fabricated using the existing technology pipeline described above.

In practice, the approaches taken by each of these pipelines generate comparable results. However, there are two notable advantages of our method over the existing technologies approach. First, our method offers a considerable speedup in the design process, as the conversion from the art representation to the fabrication representation is instantaneous. Our formulation also explicitly embraces fabrication constraints that go beyond the offset requirement, such as the design's eventual conversion to g-code polylines. At first glance, it appears as if the existing technologies pipeline offers an advantage over our implementation, because it appears to work with the specified bèzier curves throughout the entire design process. However, the implementation of each discrete step in this process could contain any number of conversions between design parameterizations: rasterizing the stroke width, fitting "offset" curves to the stroke outlines, etc. The proprietary implementations within Illustrator make it difficult to pinpoint exactly which of these approximations may be occurring. In any case, the bèziers of the final design will have to be ported to polylines for fabrication. This makes it likely that the manually constructed fabrication view has been subjected to more approximations than the results from our system.

There are also several subtle nuances between the underlying algorithms in each pipeline. For example, stroke-width offsets in Illustrator tend to blur sharp corners as the offset increases, whereas our method preserves the sharpness of the intersection. We also note the subtle fact that changing the width of a spline can alter the perceived shape of that spline. This can result in the unexpected exposure of artifacts that were hidden in the original design. For example, consider a region in which there are several vertices near one another, arranged in a zigzag pattern. Thickening the stroke around this central path tends to hide or average out such high frequency details, while thinning the stroke might expose high frequency details that were previously (intentionally) hidden within the extent of the stroke width. Our approach avoids this issue in most situations.

Since our system relies exclusively on the underlying parameterization of the splines rather arbitrary display settings like stroke-width, it may also encourage users to think about the design in terms of the median spline itself. This approach promotes a clean, precise expression of the design – rather than one that simply relies on visual perception – as it reduces the artist's freedom to hide undesirable artifacts.

6 Results

While we managed to obtain satisfying results with existing tools, we found that the process to be unnecessarily time consuming and tedious. In general, it was difficult to assess the validity of any particular design until the pieces had been fed through the entire pipeline, then physically fabricated. Since the final fabrication representation cannot be edited in a reasonable way, any required alterations must be made in the original art view, all the way at the beginning of the pipeline again. Thus, after every iterative change, the artist was forced to alter the stroke width, repeat the live paint process, then move the new pieces into an optimized fab layout once again. This makes the design process frustratingly inefficient, as iterations are very slow, and there are several intermediate steps that prevent the artist from seeing the impact of their changes in the final representation.

The current implementation of our pipeline still suffers from several of the bottlenecks described above, including manual fabrication layout, and the inability to accurately assess the validity of a design until it has been realized. However, our system was designed with a thorough understanding of the problem and these potential extensions in mind, so we believe that future iterations of this project will be able to incorporate features that address some of these concerns directly. Moreover, our system has already offered significant improvements in the design process by (1) recognizing and exploiting the duality of the art and fabrication representations, (2) automating several tedious intermediate steps, and (3) creating an intuitive tool that uses real-time feedback to drastically reduces the tedious overhead of valid fabricable design.

These improvements have reduced the amount of time and energy necessary to convert an artistic design into valid fabricable form.

With the existing technologies pipeline, it could take up to 20 minutes to realize even small changes to simple designs, such as the puzzle. While the actions themselves were quite straightforward, there was a lot of overhead involved in order to clean up the file structures, and ensure that each step was computed in the proper order. Our automated pipeline produces the same results in a mere moment, while hiding many of the tedious details and displaying only those results that were of use to the artist. This streamlined approach compresses the iteration cycle dramatically, and empowers the artist to explore a greater portion of the nearby design space.

7 Discussion

One of the most interesting pieces of this project was the initial exploration stage, because the domain of physical graphic design is both expansive and relatively unexplored. We spent several weeks simply defining this space, gathering inspiration from previous academic works, industry endeavors, and personal artistic creations. At the end of this extensive analysis, we had familiarized ourselves with a wide range of projects, including stencils, bas relief, developable surfaces, package design, chip carving, and various forms of papercraft. For each of these domains, we tried to identify a set of constraints that would need to be satisfied in order to realize a digital design in that particular fabricated form. All told, we identified nearly 30 distinct constraints spanning four broad categories:

- **Aesthetic** - How well do we approximate the target design? Is our design readable (scale vs. detail)?
- **Material** - Are there any minimum or maximum dimensions? Should we consider any material properties such as grain (wood) or layers (3D printing), whether for strength or aesthetics?
- **Fabrication** - How many degrees of freedom does the machine have? Can the actions be executed in any order?
- **Feasibility** - Is the final design assemblable? Do we need to consider multi-piece alignment or structural stability?

As one might expect, we found a substantial amount of overlap among the constraints necessary for any individual problem. Thus, we believe that it is both possible and desirable to identify a fabrication-aware paradigm that goes beyond any single project domain listed above, in order to address the more general class of physical graphic design. We attempted to conceptualize our system within this larger context, such that it might be a start toward generalized physical graphic design. This goal is both interesting and motivating, but it is well beyond the scope of any single paper. Thus, we narrowed our interests to consider a single domain in order to approach this goal from a more tractable angle. We chose intarsia because it exhibited many of the most common constraints, while also presenting several unique challenges and potential extensions in and of itself.

Initally, we designed a full pipeline for intarsia, which went from the initial pattern creation all the way through to an optimized output for either traditional scroll-saw creation or CNC milling. This pipeline also envisioned an interactive, fabrication-aware editing stage. This latter part of the pipeline – from editing to final output – is where we chose to focus our efforts. This project was a great exercise in problem exploration and scoping, as we were surprised to find that even within this increasingly narrowing scope, there were many potentially interesting directions left unexplored.

Part of the reason for this additional curbing was the fact that we encountered several challenges throughout the course of this project, both in the form of one-time setbacks and persistent obstacles. Small but impactful hurdles included issues like library compilation and compatibility of the disparate units that we chose, as this initial setup proved to be much more time intensive than we originally expected. Our library choices also required us to spend a decent amount of time implementing wrappers and interfaces that we had previously taken for granted, such as the ability to read, write, and display SVG data, or interact with the vertices in a reasonable way.

One of the most challenging obstacles we faced was the ability to understand, interpret, and correctly interface with the CGAL Arrangement2 implementation. While this is widely understood to be the most robust open-source solution for planar mapping, the library is certainly not without its flaws. In particular, we found that the documentation was somewhat lacking. It was plentiful, in the sense that every class and function was documented, and several high level descriptions, use cases, and examples were available. However, the function descriptions were often minimal or non-existent, and the project structure was somewhat of a labyrinth, requiring several deep dives through the documentation before truly understanding what you were searching for. This made it difficult to discern the underlying implementation of the code that we were trying to leverage– including the data structures, the assumptions, and the intent of various abstractions – so progress was often much slower than anticipated. This will continue to pose an issue in future iterations of this project, as we attempt to augment the functionality for features like the ideal bi-directional editing approach discussed in section 3. This effort will propel us even further from the sample use cases which – with careful scrutiny – have guided us to this point.

The Arrangement2 library also posed its fair share of practical obstacles, which required the conception and development of several heuristics in order to ensure that we were getting the proper result. For instance, the Arrangements2 class provides functionality to traverse the counter-clockwise boundary of a given face, ensuring that the segments are returned in sequential order. We assumed that the segments would also be returned in the correct orientation, such that the endpoint of the current segment would be the starting point for the next. However, after several mysteriously failed results, we realized that despite their being stored as directed half-edges with explicit "source" and "target" members, this was not necessarily the case. Since the orientation was crucial to our algorithms, we had to systemically evaluate the pairs of points each time we traversed the edges, flipping the points if necessary and accounting for all corner cases that could occur at the beginning or end of a boundary cycle.

Logical discrepancies and misinterpretations of this sort were not uncommon while working with this library, and they were often much more challenging to debug. One of the most persistent issues we encountered was our attempt to create planar maps directly from the given artistic splines. We were able to parse these bèziers into the appropriate format, and feed them through the planar mapping code in order to arrive at what appeared to be a proper planar graph of the illustration we had inserted (Figure 11). However, each time we tried to write out the faces using the curve segments from the original design, we ended up with designs that were very close to what we expected, but always slightly off in one way or another. In some cases, this resulted in properly closed but slightly overlapping regions (Figure 11d), which is invalid because it requires multiple pieces to occupy the same physical space. In other attempts, we ended up with curve segments that had clearly over- or under-shot their correct intersection values, leaving us with regions that were decidedly not closed (Figure 11c). To the best of our knowledge, these issues stemmed from numerical imprecision within the library itself, despite the fact that we used the suggested numeric and algebraic kernels in all of our underlying template implementations. Despite several months' worth of effort, these issues remain unsolved. As previously mentioned, our current implementation circumvents the problem by converting all input curves to polylines

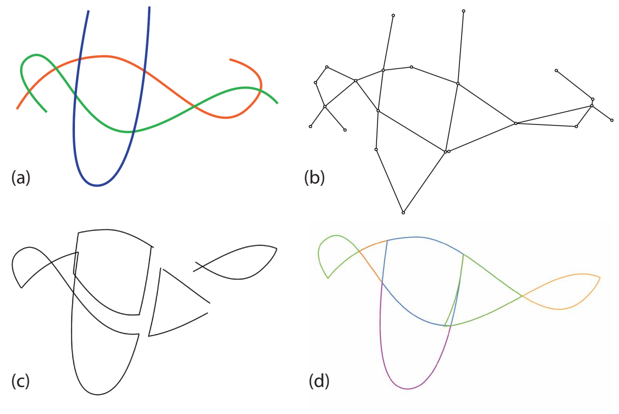


Figure 11: *Documenting issues with direct bèzier planar mapping using CGAL Arrangement2. The bèzier input (a) appears to generate a proper planar graph (b), but attempts to write out the closed regions using the underlying curve segments gave undesirable results (c,d). Note that the pieces of (c) were shifted using Adobe Illustrator in order to clearly display the errors in our output. The pieces of (d) have been falsely colored to emphasize their closed-region representation, which is correct aside from the green corner protruding into its neighboring blue piece.*

before computing the planar map. In addition to its merits as described earlier in the paper, this decision to embrace polylines was a breakthrough that enabled us to move forward with the project.

8 Limitations and Future Work

Although our system already contributes tangible gains over existing pipelines for fabricable intarsia design, the current implementation serves as a mere foundation for the possibilities yet to be explored. Throughout the paper, we have suggested a number of features, like bèzier flattening, that could be improved through optimized algorithms. There are also a number of places where we might consider methods that differ entirely from our current implementation. One example of this would be the use of signed distance fields in our offsetting algorithm in order to determine whether our offsets have caused edges of the face to cross one another, resulting in a non-simple closed region which is invalid for fabrication.

In future work, we would also like to address several small artifacts that our algorithm currently produces. One such artifact stems from the fact that CGAL's Arrangement2 implementation does not allow for a gap parameter, which currently requires the artist to artificially extend lines beyond the desired intersection point to ensure that the algorithm registers it properly. We have already implemented a sanitization algorithm which detects and removes the unwanted tails that extend into neighboring pieces due to this artificial extension. However, at intersections of three or more vectors, this artificial extension often enclosed a secondary region that was not present in the original design. These artifacts manifest themselves as tiny additional faces in the final fabrication design – three such enclosures can be found in the elephant's final fabrication view (as seen in Figure 8). While these are trivial to remove in a post process, we would like to allow the user to deal with these artifacts directly in our system. If the user alters the art view after removing these unwanted faces, we will also need to consider the interesting question of how to store and respect these fabrication edits, so that the faces do not continue to appear in future computations of the fabrication representation.

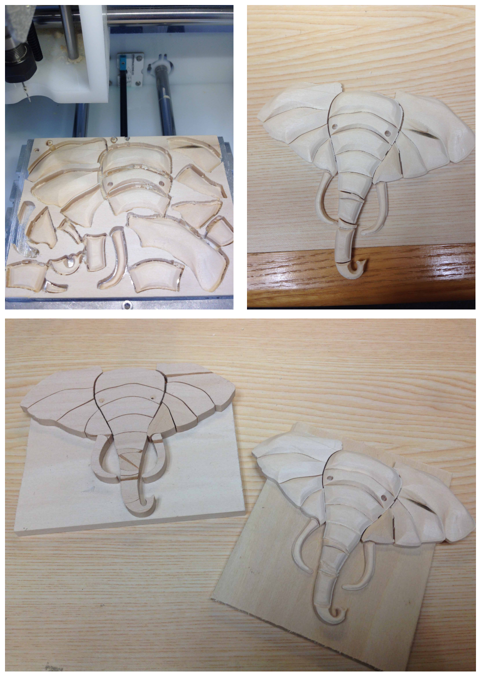


Figure 12: *Documentation of our shaping extension in the existing technologies pipeline. This could serve as an interesting future direction, as it offers more design and visualization challenges, while also producing a much richer final artifact as compared to the 2D boundary representation only (bottom).*

Beyond the features which we have already implemented, we would like to implement a validity simulation that will localize any regions of the design that are unrealizable with the fabrication mechanisms (bit diameters) specified by the user. A path might be considered unrealizable if it has high curvature regions or other openings that are too narrow for the bit to pass through. These issues can be localized to independent pieces of the design, or they can result from a global layout problem in which two pieces are simply too close to one another in the final fabrication layout. All of these issues stem from relative sizing issues between the design and the finite-width bit being used for fabrication, so a naive solution to this problem is to continually scale up the design until the size ratio allows for a permissible realization of the design. However, even if the scale of the final piece is irrelevant (which is rarely true), this method will eventually run into fundamental limits, such as material exhaustion or a maximum build volume. Thus, it would be preferable to address the issue on a more fundamental level with a system that provided intuitive editing controls and synchronized feedback that guided the user toward a valid fabricable design.

We could also imagine relaxing any number of the assumptions made in this paper, such as our restriction to the 2-dimensional case, in which we ignore any overall edge or surface shaping to focus exclusively on the boundaries defining each piece. In the future, we hope to extend the system to include 2.5D components like surface shaping, which determines the height field imposed on each individual piece. We started exploring the creation of these 2.5D intarsia pieces through our existing technologies pipeline (12) by including 3D modeling softwares such as Autodesk Maya and Autodesk Fusion 360. The process was relatively painful, as it required roughly three weeks and four broken endmills to find a pipeline that worked; ultimately, though, we found the results quite satisfying. The additional shaping gives more depth to the final result, while offering additional challenges and interesting questions for us to reconcile. We might realize an extension like this by allowing the user to impose a height field on each piece, using a process like diffusion curves [Orzan et al. 2008] or any of the various processes used to generate bas relief pieces [Kerber et al. 2012].

We then hope to tackle other parts of our original pipeline, building out functionality to address things like 3D visualization, optimal grain-aware nesting, initial pattern creation, and more complicated design elements like layering. Interesting variations of problems like nesting have been studied in previous works [Koo et al. 2016], but our application places unique constraints on each of these elements so the idea transfer would still be very interesting to explore. In the end, our system would ideally guide the user through the entire design process, from an image input all the way through to visualization and a final physical piece. This comprehensive tool would further reduce the barrier to entry for intarsia design and realization, which is currently only accessible to skilled artisans.

9 Conclusion

This project effectively conceptualizes and implements a novel paradigm for the intuitive design of fabricable Intarsia. We conduct an extensive and methodical exploration of the design space, from both theoretical and practical perspectives. This inspired the dual representation that we presented above, while allowing us to identify and evaluate a pipeline of existing technologies against which we could compare our methods. Finally, we leverage our findings in an interactive design tool that propagates user edits from one fundamental representation to the other, allowing users to immediately see the impact of their edits in the fabrication domain. By recognizing, defining, and subsequently exploiting the dual representation that is necessary for design and fabrication to coexist, our system is able to make noticable improvements to the process of iterative fabrication-aware design. This project has also laid the groundwork for more extensive exploration, not only in the context of Intarsia, but the more general class of physical graphic design.

Acknowledgements

This work was completed as an undergraduate senior thesis at Dartmouth College, though it began as an intern project at Adobe's Creative Technologies Lab in Seattle. This work was supported by the Adobe Research Women-in-Technology Scholarship. We would like to thank Adobe Research, for the generous scholarship opportunity that enabled this collaboration, and for their support and courtesy in allowing our group to continue the work following the internship duration.

References

ASENTE, P., SCHUSTER, M., AND PETTIT, T. 2007. Dynamic planar map illustration. In *ACM SIGGRAPH 2007 Papers*, ACM, New York, NY, USA, SIGGRAPH '07.

BÄCHER, M., WHITING, E., BICKEL, B., AND SORKINE-HORNUNG, O. 2014. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph. 33*, 4 (July), 96:1–96:10.

BAUDELAIRE, P., AND GANGNET, M. 1989. Planar maps: An interaction paradigm for graphic design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, CHI '89, 313–318.

BERNSTEIN, G. L., AND LI, W. 2015. Lillicon: Using transient widgets to create scale variations of icons. *ACM Trans. Graph. 34*, 4 (July), 144:1–144:11.

BRONSON, J., RHEINGANS, P., AND OLANO, M. 2008. Semi-automatic stencil creation through error minimization. In *Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering*, ACM, New York, NY, USA, NPAR '08, 31–37.

CANNY, J. 1986. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8*, 6 (Nov), 679–698.

FISCHER, K., 2000. Piecewise linear approximation of bézier curves.

HAIN, T. F., AHMAD, A. L., RACHERLA, S. V. R., AND LANGAN, D. D. 2005. Fast, precise flattening of cubic bézier path and offset curves. In *Computers & Graphics*, Elsevier, 656 – 666.

IARUSSI, E., LI, W., AND BOUSSEAU, A. 2015. Wrapit: Computer-assisted crafting of wire wrapped jewelry. *ACM Trans. Graph. 34*, 6 (Oct.), 221:1–221:8.

JAIN, A., CHEN, C., THORMÄHLEN, T., METAXAS, D., AND SEIDEL, H.-P. 2015. Multi-layer stencil creation from images. *Comput. Graph. 48*, C (May), 11–22.

KERBER, J., WANG, M., CHANG, J., ZHANG, J. J., BELYAEV, A., AND SEIDEL, H.-P. 2012. Computer assisted relief generation&#x2014;a survey. *Comput. Graph. Forum 31*, 8 (Dec.), 2363–2377.

KOO, B., HERGEL, J., LEFEBVRE, S., AND MITRA, N. 2016. Towards zero-waste furniture design. *IEEE Transactions on Visualization and Computer Graphics PP*, 99, 1–1.

LAU, C., SCHWARTZBURG, Y., SHAJI, A., SADEGHIPOOR, Z., AND SÜSSTRUNK, S. 2014. Creating personalized jigsaw puzzles. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*, ACM, New York, NY, USA, NPAR '14, 31–39.

LIU, S., JACOBSON, A., AND GINGOLD, Y. 2014. Skinning cubic bézier splines and catmull-clark subdivision surfaces. *ACM Trans. Graph. 33*, 6 (Nov.), 190:1–190:9.

MCCRAE, J., UMETANI, N., AND SINGH, K. 2014. Flatfitfab: Interactive modeling with planar sections. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, ACM, New York, NY, USA, UIST '14, 13–22.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 Papers*, ACM, New York, NY, USA, SIGGRAPH '08, 92:1–92:8.

PRÉVOST, R., WHITING, E., LEFEBVRE, S., AND SORKINE-HORNUNG, O. 2013. Make it stand: Balancing shapes for 3d fabrication. *ACM Trans. Graph. 32*, 4 (July), 81:1–81:10.

SCHWARTZBURG, Y., AND PAULY, M. 2013. Fabrication-aware design with intersecting planar pieces. *Computer Graphics Forum 32*, 2pt3, 317–326.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MĚCH, R. 2012. Stress relief: Improving structural strength of 3d printable objects. *ACM Trans. Graph. 31*, 4 (July), 48:1–48:11.

UMETANI, N., KAUFMAN, D. M., IGARASHI, T., AND GRINSPUN, E. 2011. Sensitive couture for interactive garment editing and modeling. *ACM Transactions on Graphics (SIGGRAPH 2011) 30*, 4.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4 (July), 86:1–86:11.

UMETANI, N., KOYAMA, Y., SCHMIDT, R., AND IGARASHI, T. 2014. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph. 33*, 4 (July), 65:1–65:10.

WEIN, R., BERBERICH, E., FOGEL, E., HALPERIN, D., HEMMER, M., SALZMAN, O., AND ZUKERMAN, B. 2017. 2D arrangements. In *CGAL User and Reference Manual*, 4.9.1 ed. CGAL Editorial Board.

WHITING, E., OUF, N., MAKATURA, L., MOUSAS, C., SHU, Z., AND KAVAN, L. 2017. Environment-scale fabrication: Replicating outdoor climbing experiences. In *Proc. 2017 CHI Conference on Human Factors in Computing Systems*, ACM, 1794–1804.

YAO, J., KAUFMAN, D. M., GINGOLD, Y., AND AGRAWALA, M. 2017. Interactive design and stability analysis of decorative joinery for furniture. *ACM Trans. Graph. 36*, 2 (Mar.), 20:1–20:16.

YUMER, M. E., CHAUDHURI, S., HODGINS, J. K., AND KARA, L. B. 2015. Semantic shape editing using deformation handles. *ACM Trans. Graph. 34*, 4 (July), 86:1–86:12.