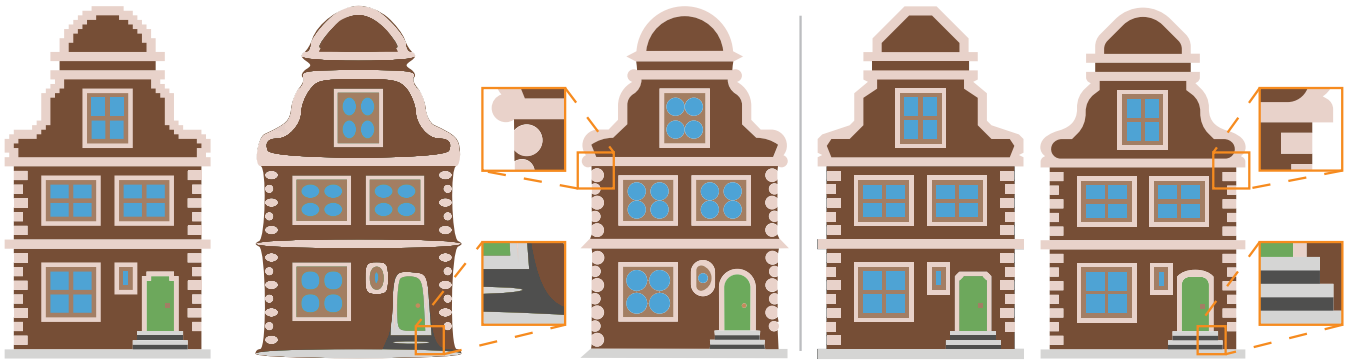
PolyFit: Perception-Aligned Vectorization of Raster Clip-Art via Intermediate Polygonal Fitting

EDOARDO ALBERTO DOMINICI, University of British Columbia
NICO SCHERTLER, University of British Columbia
JONATHAN GRIFFIN, University of British Columbia
SHAYAN HOSHYARI, Adobe Inc.
LEONID SIGAL, University of British Columbia
ALLA SHEFFER, University of British Columbia



(a) Raster Input (b) [Selinger 2003] (c) [Hoshyari et al. 2018] (d) Intermediate Polygon (e) Ours

Fig. 1. Vectorizing raster clip-art inputs (a) using existing methods, here Potrace [Selinger 2003] (b) and [Hoshyari et al. 2018] (c), results in visible artifacts (highlighted in zoomed-in insets). PolyFit outputs (e), created using an intermediate polygonal approximation step (d), are more consistent with viewer expectations than those produced by these alternatives. Please zoom in online to see details. Input image ©IconScout — www.iconscout.com

Raster clip-art images, which consist of distinctly colored regions separated by sharp boundaries typically allow for a clear mental vector interpretation. Converting these images into vector format can facilitate compact lossless storage and enable numerous processing operations. Despite recent progress, existing vectorization methods that target such data frequently produce vectorizations that fail to meet viewer expectations. We present *PolyFit*, a new clip-art vectorization method that produces vectorizations well aligned with human preferences. Since segmentation of such inputs into regions had been addressed successfully, we specifically focus on fitting piecewise smooth vector curves to the raster input region boundaries, a task prior methods are particularly prone to fail on. While perceptual studies suggest the criteria humans are likely to use during mental boundary vectorization, they provide no guidance as to the exact interaction between them; learning these interactions directly is problematic due to the large size of the solution space. To obtain the desired solution, we first approximate the raster region boundaries with coarse intermediate polygons leveraging a combination of perceptual cues with observations from studies of human preferences. We then use these intermediate polygons as auxiliary inputs for computing piecewise smooth vectorizations of raster inputs. We define a finite set of potential polygon to curve primitive maps, and learn the mapping from the polygons to their best fitting primitive configurations from human annotations, arriving at a compact set of local raster and polygon properties whose combinations reliably predict human-expected primitive choices. We use these primitives to obtain a final globally consistent spline vectorization. Extensive comparative user studies show that our method outperforms state-of-the-art approaches on a wide range of data, where our results are preferred three times as often as those of the closest competitor across multiple types of inputs with various resolutions.

CCS Concepts: • **Computing methodologies → Image manipulation**.

Additional Key Words and Phrases: clip-art, vectorization

ACM Reference Format:
Edoardo Alberto Dominici, Nico Schertler, Jonathan Griffin, Shayan Hoshyari, Leonid Sigal, and Alla Sheffer. 2020. PolyFit: Perception-Aligned Vectorization of Raster Clip-Art via Intermediate Polygonal Fitting. *ACM Trans. Graph.* 1, 1, Article 77 (June 2020), 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Edoardo Alberto Dominici, University of British Columbia, dedoardo@cs.ubc.ca; Nico Schertler, University of British Columbia, nschertl@mail.ubc.ca; Jonathan Griffin, University of British Columbia, jcgriff@cs.ubc.ca; Shayan Hoshyari, Adobe Inc., s.hoshyari@gmail.com; Leonid Sigal, University of British Columbia, lsigal@cs.ubc.ca; Alla Sheffer, University of British Columbia, sheffa@cs.ubc.ca.

0730-0301/2020/6-ART77 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

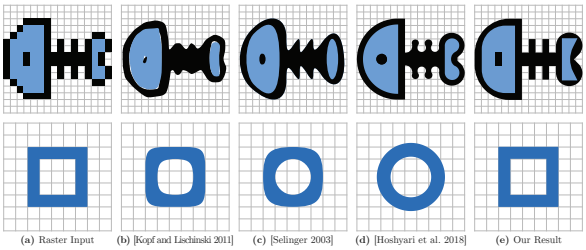


Fig. 2. Given a raster input (a), human observers prefer the vectorizations in (e) created by PolyFit over those in (b-d), created using the methods of Kopf [Kopf and Lischinski 2011], Potrace [Selinger 2003], and [Hoshyari et al. 2018] by factors of 10:0, 10:0 and 7:3, respectively, for the example in the first row, and 9:1, 10:0, 10:0 for the example in the second row. This preference persists despite the fact that rasterizing all three vector images in the bottom row would reproduce the input raster exactly.

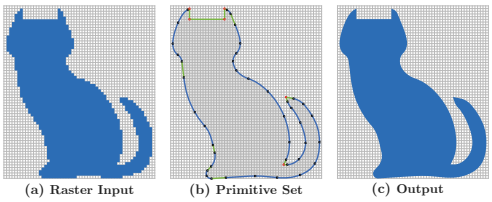


Fig. 3. Vector representation: (a) raster input; (b) primitive set (lines in green, Bézier curves in blue, C^0 corners in red). (c) output vector image.

1 INTRODUCTION

Artist-drawn *clip-art* images consist of distinctly colored regions delineated by piecewise smooth visually pronounced boundaries, and are ubiquitously used in digital media. Clip-art images can be compactly and losslessly represented in vector form; yet, for a variety of reasons, large numbers of clip-art images are still created and stored in raster format, e.g., the Adobe Stock Image database contains over nine million raster images labeled as clip-art or icons [Hoshyari et al. 2018]. Vectorizing these images would enable resolution-free reuse of artwork created for legacy displays and facilitate a range of operations such as resizing or editing, which are easier to perform on vector rather than raster data. Vectorizing this data in a manner consistent with viewer expectations poses unique challenges, motivating the development of algorithms specifically designed for clip-art vectorization (e.g., [Hoshyari et al. 2018; Kopf and Lischinski 2011; Selinger 2003]). Notably, while prior methods [Kopf and Lischinski 2011] successfully segment raster clip-art into regions consistent with viewer expectations, vectorizing the raster boundaries between these regions while meeting observer expectations remains challenging; existing algorithms still often fail to produce vector boundaries consistent with human preferences (Figures 1b-c, 2b-d). We present a new approach for vectorization of region boundaries in raster clip-art images that significantly outperforms these earlier approaches, producing results much better aligned with viewer expectations (Figures 1e, 2e).

Given a raster segmentation obtained using off-the-shelf methods [Kopf and Lischinski 2011], we aim to convert the raster boundaries between the obtained regions into vector form. This conversion requires computing a set of geometric primitives that jointly capture the input boundary shape as perceived by human observers (Figure 3). Computing these desirable vectorizations requires determining the number and type of primitives in the fitted sequence, the locations and types of the transitions between them (C^0 corners vs. smooth G^1 or G^2 transitions), and their individual parameters. Perception studies and computer graphics literature (Section 3) identify a number of core principles likely to impact human perception and affect the set of piecewise-smooth primitives human observers are likely to mentally fit to the given raster data. However, converting these principles into an actionable algorithm requires learning the interaction between them and quantifying their combined impact on the mental fitting process. Learning this relationship directly from pairs of raw raster images and manually vectorized counterparts is impractical [Hoshyari et al. 2018]: vectorizing a single region requires 30-45 minutes of a computer artist's time, making construction of a large-scale training set problematic. Furthermore, such pairs cannot be created by simply rasterizing vector boundaries; since rasterization is lossy, the vectorization output humans expect given a raster image is often not the same as such originating vector data (see Figure 2 (bottom) where all vector outputs once rasterized reproduce the input). Additionally, directly searching for the desired vectorization requires operating on a huge mixed discrete (e. g., number and type of primitives) continuous (e. g., primitive parameters) solution space, which is computationally challenging.

We overcome both challenges and successfully compute the desired vectorizations by solving a simplified variant of the vectorization problem first, and then using this intermediate output to guide the rest of the computation (Section 3). Specifically, instead of directly solving for a piecewise smooth approximation of the raster input, we first approximate this input using a coarse polygonal, or piecewise *linear*, approximation designed to match human expectations (Fig. 1d, Section 4). This choice is motivated by the observation that the approximating polylines human observers prefer (Appendix A) are strongly correlated with their preferred piecewise smooth vectorizations in terms of primitive counts, as well as corner and other transition locations. We formulate polygonal fitting as a computation of a shortest cycle over a graph of possible polyline edges, and follow perception principles and observations gleaned from human-traced polygons (Appendix A) to define the costs of traversing different edges in this graph.

We use these *intermediate* polygons as auxiliary input to compute piecewise smooth vectorizations of the raster inputs (Section 5). We use a learned classifier, trained on a compact set of manually annotated polygons, to identify the polygon corners that correspond to final output corners and to identify the piecewise smooth set of primitives best suited for fitting the boundary next to each polygon corner. We optimize the shape of the selected primitives to obtain a final globally consistent spline. Perception studies point to a strong viewer preference for more regular input interpretations [Koffka 1955; Wagemans et al. 2012], stemming from an expectation that observed data conforms to a regular, axis-aligned, and symmetric pattern. We thus enforce the fitted polygons and splines to conform to regular patterns detected on the raster data and regularize near-regular polygon and spline structures (Section 6). The combined method efficiently computes desirable solutions on a wide range of tested inputs.

We validate our approach by applying it to more than a hundred inputs and comparing the results to algorithmic alternatives both visually and via an extensive user study (Section 8). Study participants preferred our results by a margin of 3:1 compared to the closest-ranking prior method across all tested input resolutions, with the margin growing to 16:1 for images with region sizes below 16×16. We also improve on the closest-ranking prior method [Hoshyari et al. 2018] in terms of runtime and generality: our method is an order of magnitude faster, taking about a second to vectorize a typical 64×64 input compared to over a minute. While the prior method requires separately trained models for different input resolutions, we use a single model trained using a compact set of annotated examples for all resolutions.

To summarize, our core contribution is a new, resolution independent method for clip-art vectorization that significantly outperforms prior art and is particularly well suited for low-resolution inputs, where existing frameworks perform especially poorly (e.g., Figure 2,top). We achieve this goal by leveraging the connections between perception-aligned polygonal and piecewise smooth raster boundary approximations.

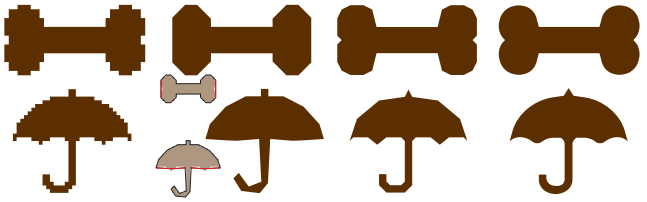
2 RELATED WORK

We build on research across a number of domains outlined below.

Approximating Sample Points. Vectorization shares some commonalities with the classical problems of fitting curves or polygons to sample points. Traditional fitting methods [Farin 2002; Fleishman et al. 2005; Liu and Wang 2008] approximate unordered noisy point samples with smooth spline curves by balancing fit accuracy against curve fairness. Computational statistics and operations methods approximate dense input points using coarse polylines, balancing accuracy against edge count [Aronov et al. 2004; Bellman and Roth 1969; Goldberg et al. 2014]. The inputs and goals of our boundary fitting method are distinctly different from those employed in both settings. Instead of dense samples, we process raster boundaries with axis-aligned edges; instead of denoising, interpolating, or approximating this input within a given accuracy threshold, we seek to recover it's piecewise smooth interpretation consistent with human perception.

Stroke Beautification. Our goals are related to those of methods that seek to beautify, or regularize, artist strokes on free-hand sketches [Baran et al. 2010; McCrae and Singh 2008, 2011]. However, the raster polyline inputs we process strongly deviate from the assumptions these methods make about the input stroke geometry. In particular, the approaches for tangent and curvature estimation these methods employ are ill-suited for raster data. As a result, applying these frameworks to raster boundaries leads to inadequate results [Hoshyari et al. 2018].

Natural Image Vectorization. Techniques for natural image vectorization address two distinct challenges: segmenting the input images into a set of compact, color-coherent raster regions, and fitting the raster region boundaries using sequences of curve primitives. Most natural image vectorization methods focus on the image segmentation stage [Favreau et al. 2016; Lecot and Levy 2006; Orzan et al. 2008; Sun et al. 2007; Wang et al. 2017; Xia et al. 2009; Xie et al. 2014].



(a) Raster Input (b) Potrace Polygon (c) Our Polygon (d) Our Vectorization

Fig. 4. Potrace [Selinger 2003] polygons (b) tend to smooth out details in the input (as highlighted in the overlays shown in the insets), which are not recoverable by the subsequent spline fit. Our method (c-d) preserves details at the polygon and final fit levels.

Due to the inherent complexity of such images, these methods tend to produce large sets of segments with highly irregular boundaries. Consequently, in the absence of interactive human adjustments [Jun et al. 2017], the boundary fitting stage of these methods focuses on simplifying and smoothing these boundaries [Adobe 2017; Caselles et al. 1997; Figueiredo et al. 2000; Kass et al. 1988]. This lossy smoothing approach is well suited for natural images but fails to preserve the visually distinct, piecewise smooth segment boundaries of artist-designed clip-art [Hoshyari et al. 2018; Yang et al. 2016]. Super-resolution methods that target natural images [Wang et al. 2015] perform equally poorly on artist data [Hoshyari et al. 2018]. Our focus is on vectorization of artist-generated content, which on the one hand is simpler to segment, but on the other hand requires a much more careful boundary analysis and fitting.

Vectorization of Artist-Generated Imagery. Several prior works focus specifically on vectorization of artist-generated imagery, including cartoons, binary imagery, pixel art, and clip-art. Zhang *et al.* [2009] and Sykora *et al.* [2005] vectorize cartoon images and animations. Both methods fit boundaries between pairs of regions using G^1 or G^2 continuous curves and introduce discontinuities only at corners where multiple regions meet. Fatemi *et al.* [2016] upsample binary images using similar continuity assumptions. These assumptions do not hold for clip-art images, which often contain visually pronounced C^0 corners and curvature discontinuities along individual region boundaries (Figures 2, 3). Yang *et al.* [2016] employ user-provided discontinuity and corner locations to fit a piecewise Bézier spline to raster boundaries; providing such locations as input is a time-consuming task. Our method successfully locates viewer-expected discontinuities algorithmically with no user input.

Dedicated methods target upscaling [Eagle 1997; Mazzoleni 2001; Stepin 2003] and vectorization [Kopf and Lischinski 2011] of low-resolution pixel-art images. Kopf and Lischinski [2011] focus on resolving topological ambiguities during image segmentation. Like earlier methods, they employ G^2 curve fitting to vectorize individual boundaries between raster regions. This strategy is inadequate on inputs that contain visibly distinct C^0 corners (e. g., Figure 2).

Several commercial packages (e. g., [ScanFont 2017; Selinger 2003; Vector Magic 2017; Weber and Herzog 2004]) specifically address clip-art vectorization. As demonstrated by Hoshyari *et al.* [2018], Potrace [Selinger 2003] and VectorMagic [2017] often exhibit undesirable artifacts on typical clip-art inputs; others, e. g., [Weber and Herzog 2004] are no longer maintained. ScanFont [ScanFont 2017] is designed specifically for fonts and uses character recognition priors.

Our approach of fitting an intermediate polygon is inspired by Potrace [Selinger 2003], who first compute a polygonal approximation of the raster and then replace it with a smooth spline. Unfortunately, the polygon fitting approach they use results in loss of fine details (Figure 4b), especially noticeable on lower resolution inputs. This detail loss is then propagated to the final vector fits. We constrain our polygons to accurately preserve input details, leading to more accurate final fits (Figure 4c). Hoshyari *et al.* [2018] detect the location of perceptually plausible C^0 corners along the vectorized boundaries using a learned classifier. They acknowledge that learning such placement from corner annotation alone is challenging and overcome classifier inaccuracies by integrating the classifier within a complex perception-motivated pipeline that empirically determines which classifier-suggested corners to discard. Our two-step vectorization process is able to more accurately pinpoint viewer-expected corner locations and achieves better fitting quality (Figs. 1, 2) at a fraction of the compute time (Section 8). Our validation study participants preferred PolyFit results over those of Hoshyari *et al.* 75.5% of the time (Section 8).

3 METHOD OVERVIEW

The input to our method is raster clip-art images, which are expected to consist of a number of distinctly colored regions with clearly identifiable inter-region boundaries. To vectorize this input we first segment it into single-color regions using the framework of [Kopf and Lischinski 2011]. Our core focus is to subsequently convert the raster, or piecewise axis-aligned, boundaries between the resulting regions into piecewise smooth curves consistent with viewer perception.

Output Properties. Before describing our algorithm, we discuss the main criteria that prior perception and computer graphics research [Hoshyari et al. 2018; Koffka 1955; Wagemans et al. 2012], identifies as likely to impact the human mental vectorization process: accuracy, simplicity, regularity, and continuity (Figure 6). We subsequently employ these criteria in our vectorization framework.

Accuracy predicts that the vectorizations humans envision are geometrically close to the input raster boundaries. Promoting accuracy encourages vector outputs that are maximally close to the raw raster input, and in particular ones that, when rasterized, reproduce (or nearly reproduce) this input [Hoshyari et al. 2018]. The *simplicity* principle of Gestalt psychology indicates that human observers prefer simpler, more compact explanations of observed phenomena [Wagemans et al. 2012]. In the context of vectorization, simplicity argues for an output that consists of a minimal number of geometric primitives, for prioritizing straight lines over curves, and for preferring curves with smaller curvature variation [Baran et al. 2010; Hoshyari et al. 2018; McCrae and Singh 2008]. Prior research emphasizes that human observers are highly sensitive to inflections, or curvature sign changes, that are present in the output and are not self-evident in the input [Baran et al. 2010; McCrae and Singh 2008]. The simplicity principle also indicates that observers prefer to group data into *regular* patterns. Research indicates that viewers expect observed exact or approximate regularities such as parallelism, symmetries, or axis-alignment present in the raster input to be preserved in the vector output. Finally, and critically, studies indicate that observers are likely to mentally convert raster region boundaries into piecewise *continuous* curves, with a small number of C^0 corners or abrupt curvature changes [Baran et al. 2010; Wagemans et al. 2012].

Given a raster input, we expect observers to balance these, frequently conflicting, properties to mentally conjure a piecewise smooth output.

Boundary Vectorization. In the general case, our method seeks to approximate an axis-aligned polyline raster boundary network using a connected network of spline curves, each consisting of a sequence of curve primitives. For simplicity, the formulation presented here and in Sections 4 and 5 addresses vectorization of a single closed raster boundary. Section 7 describes the extension of this method to the case of multiple regions separated by a network of boundaries. The main difference between this setup and the single boundary one is the need to resolve fitting inconsistencies that may arise near network junctions formed by three or more boundary segments (Figure 13).

Given a raster boundary defined as a sequence of vertices located at pixel corners, we seek to vectorize, or fit, it using a piecewise smooth spline curve, defined using a sequence of *primitives*. Each spline *primitive* is expected to approximate a *segment* of the input boundary. Neither the number of primitives, nor the segments they each correspond to, are known *a priori*, and both the lengths of the segments approximated by each primitive and the primitives' geometry can vary significantly (see Figure 3). Consequently, searching directly for the desired vectorization for a given raster boundary requires operating on a very large mixed discrete and continuous solution space which includes the number of output spline primitives, their types, their geometric parameters, and the locations of their endpoints. More importantly, while prior research identifies the core principles behind the vectorizations we desire, it does not quantify the specific balance required among them.

We sidestep the challenges of directly learning this balance from raster-vector pairs by using a two-step vectorization process (Figure 5). Instead of computing a piecewise smooth vectorization directly, we first compute a polygonal approximation of the raster input using the same perceptual criteria. Since this setting is much simpler, we can empirically define a fitting cost function and a corresponding algorithm (Section 4) that jointly lead to polygonal approximations consistent with observer expectations (Figure 5c). We perform all our cost function measurements (including curvature) with respect to a coordinate system where a pixel edge length is one. We believe that low-resolution clipart images are meant to be displayed at their original size, making this unit constant and consequently the cost function is free of resolution dependent parameters.

We conjecture that our fitted polylines provide a rough approximation of the piecewise smooth vectorizations we seek. In particular, we note that the viewer-perceived corners on the piecewise smooth vectorization are typically a subset of the corners on these polygonal approximations, and that the slopes of the polyline edges are strongly correlated with their adjacent spline tangents (Figure 5c and g). We use these observations to define plausible spline primitives connecting consecutive polygon edge midpoints, and to

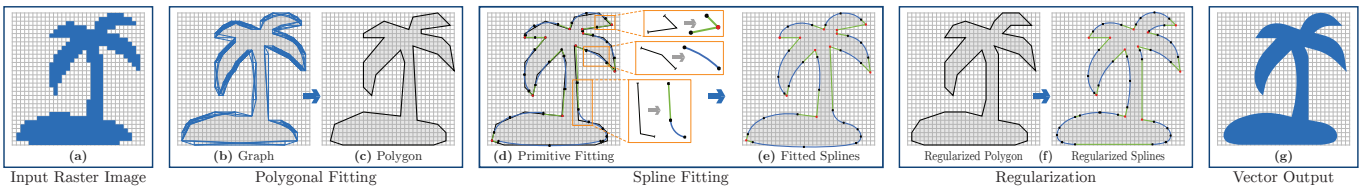


Fig. 5. PolyFit overview: given an input image (a), our first polygonal fitting stage (b-c) generates a graph of all possible edges that connect input vertices within a given accuracy threshold (b) and computes the shortest cycle on this graph with respect to a perception-motivated cost function to obtain a polygonal fit well aligned with viewer expectations (c). Our spline fitting step (d-e) uses a learned classifier to fit best-suited primitive combinations to each polygon corner (singe-curve, curve-line, line-line) (d) and computes a best-fitting spline (e) using the obtained set of primitives. We obtain more regular results by regularizing both the polygon and its corresponding spline (f) to obtain the final vector output (g).

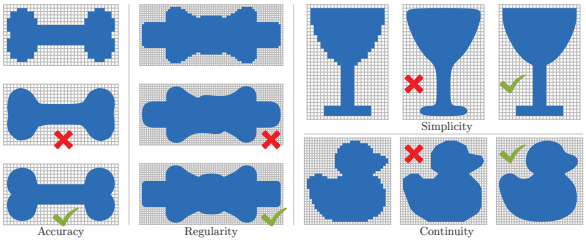


Fig. 6. Perceptual cues impacting mental vectorization: accuracy, regularity, simplicity, continuity.

learn the perceptual mapping from such corners to their best fitting primitives (C^0,G^1 or G^2) based on the properties of these fitted primitives. We perform this classification using a random forest classifier learned from a compact set of annotated combinations of consecutive polygon edges, their corresponding raster segments, and locally fitted primitives. We then optimize the geometry of the computed primitive sequence to best fit the raster input. The combined algorithm (Section 5) reliably produces the desired piecewise smooth vectorizations (Fig. 5g).

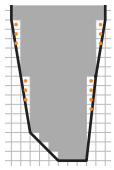
4 INTERMEDIATE POLYGONAL APPROXIMATION

We formulate the extraction of an intermediate polygonal approximation as the problem of finding a cycle that minimizes an energy function defined on pairs and triplets of edges in a directed graph $G = (V, E)$, where the vertices V represent candidate corners and the edges E represent candidate polygon segments (see Fig. 5b). Computing an optimal cycle in a directed weighted graph is known to have cubic complexity in the size of the processed graph [Skiena 2008]; to facilitate efficient computation, we keep the size of the graphs we operate on small by only including vertices and edges which are reasonably likely to be part of the final polygon (Sec. 4.1). We then compute an optimal cycle on this graph that balances accuracy, simplicity, and continuity (Sec. 4.2, Fig. 5c) via a graph construction that reduces the problem of finding the optimal cycle of an energy function defined on pairs and triplets of edges to a classical shortest-cycle problem.

4.1 Graph Construction

Our perceptual study, in which users were asked to draw approximating polygons for given raster inputs (Appendix A), indicates that viewers expect the vast majority of corners of the approximating polygon to be a subset of the corners of the input raster boundary;we define a boundary vertex as a *corner* if the raster edges emanating from it are orthogonal to one another.

The only notable exception some study participants made was to place corners at the centers of one- or two-pixel long raster segments surrounded by symmetrical corners (see inset). We speculate that this choice is motivated by the desire for more symmetric outputs. Following these observations, and to keep the graph compact we define the set of graph vertices V to include the raster boundary corners as well as the midpoints of length-one and -two raster segments.



Our study indicates that viewers expect polygon edges to closely follow the raster shape, but allow for edges whose rasterization deviates from the boundary as long as the Manhattan (axis-aligned) distance from all pixel corners to the edge is below one and all misclassified pixels (highlighed in the inset) are to one side of the edge; we view a pixel as missclassified if it is inside (to the left of) the raster boundary but its center is outside (to the right) with respect the edge, or vice versa (using the counter-clockwise orientation convention). We consequently connect each pair of vertices (v_m, v_n) with an edge $e_{m,n} \in E$ if and only if the straight line $l_{m,n} = \{v_m, v_n\}$ between them satisfies these criteria. We discard all edges that violate the Manhattan distance criterion and require all misclassified pixels to lie to one side of the line, discarding edges that do not satisfy this property. We speculate that observers relax the strict raster reproduction constraint and allow edges that lead to pixel missclassification, to accommodate inputs containing rasterizations of nearly horizontal or nearly vertical lines which are characterized by long pixel runs in a single direction, followed by a single step in the orthogonal direction (as in the inset). In all such cases the misclassified pixels lie to one side of the line. We set the directions of the edges to be counter-clockwise with respect to the region they bound. We compact the graph by merging edges with a 180° angle between them as they are perceived as a single line.

4.2 Optimal Polygon Computation

In assessing the optimality of a polygon $P = e_{ij}, e_{jk}, \ldots, e_{mi}$, we evaluate it with respect to the three perceptual criteria identified in Sec. 3, accuracy, simplicity and continuity. When assessing simplicity we evaluate both edge count and curvature variation. Since

we operate on polygons, we use discrete metrics to measure both continuity and curvature variation, expressing those in terms of polygon angles. We note that both metrics can only be meaningfully compared for angles that are sufficiently flat (in the case of continuity) or sufficiently similar (in the case of curvature variation), motivating us to use saturation limits for both ($l_{\text{ang}} = 135°$ for continuity, and $l_{\text{cont}} = 60°$ for curvature variation).

We measure the accuracy of each edge with respect to the raster boundary using the previously computed set P_{ij} of misclassified pixels. For each such pixel we measure the degree to which a pixel violates the in/out partition criterion using the distance $d(p)$ from the centers of the wrongly partitioned pixels p to the line. We define the overall per edge accuracy error as:

$$f_{\text{acc}}(e_{ij}) = \begin{cases} 0 & P = \emptyset \\ 2 \cdot \max_{p \in P_{ij}} d(p) & \text{otherwise.} \end{cases} \tag{1}$$

Note that $f_{\text{acc}} \in [0, 1]$ for all edges, approaching 1 when the Manhattan distance is violated. Edges that conform perfectly to the raster have zero energy.

We use the angles at the polygon corners $A_{ijk} = \angle e_{ij} e_{jk}$ as a discrete proxy for continuity. Note that A_{ijk} is the smaller angle out of the interior and exterior angle at the corner. We prefer flatter angles and thus more continuous transitions by setting:

$$f_{\text{cont}}(e_{ij}, e_{jk}) = \min\left(1, \frac{180° - A_{ijk}}{l_{\text{ang}}}\right), \tag{2}$$

We express simplicity as a preference for minimal change in curvature and for smaller polygon edge counts. In our discrete case, we measure curvature change as the similarity between consecutive angles. We differentiate between cases where the curvature sign is the same (measured angles are on the same side of the polyline) and inflection scenarios, where the curvature sign changes (measured angles are on opposite sides), (see inset). We distinguish between these two scenarios since, as noted in Sec. 3, viewers are particularly sensitive to inflections.

$$f_{\text{cv}}(e_{ij}, e_{jk}, e_{kl}) = \min\left(1, \begin{cases} \frac{|A_{ijk} - A_{jkl}|}{l_{\text{cont}}} & \text{if same side} \\ b_{\text{infl}} + (1 - b_{\text{infl}}) \frac{180° - \min(A_{ijk}, A_{jkl})}{180° - l_{\text{infl}}} & \text{if inflection} \end{cases}\right) \tag{3}$$

To minimize the appearance of inflections (independent of magnitude) we add an absolute inflection penalty of $b_{\text{infl}} = 0.1$ for each inflection edge and use the smaller of the two angles (i. e., the more acute one) to assess the inflection's magnitude. We empirically set the saturation limit on this magnitude to $l_{\text{infl}} = 90°$. This formulation avoids inflections when possible and prioritizes more gradual side changes over more abrupt ones if necessary.

Simplicity also argues for reducing the number of polygon edges. To this end, we assign a small penalty $\epsilon = 1e^{-3}$ to all graph edges, and further penalize *redundant* pixel-long edges. We consider such edges as redundant when they contain previously inserted midpoints. As noted by prior research [Hoshyari et al. 2018] and confirmed by our study (see Appendix), viewers prioritize approximation that incorporate such midpoints as corners, and rarely utilize redundant short edges.

$$f_{\text{simp}}(e_{ij}) = \begin{cases} 0.5 & \text{if } e_{ij} \text{ is redundant} \\ \epsilon & \text{otherwise} \end{cases} \tag{4}$$

The combined polygon cost is the sum of these terms over all edges, and pairs and triplets of consecutive edges, where we weight curvature variation less than all other terms ($\omega_{\text{cv}} = 0.25$, unit weight otherwise):

$$C(P) = \sum_{ij} f_{\text{acc}}(e_{ij}) + \sum_{ijk} f_{\text{cont}}(e_{ij}, e_{jk}) + \omega_{\text{cv}} \sum_{ijkl} f_{\text{cv}}(e_{ij}, e_{jk}, e_{kl}) + \sum_{ij} f_{\text{simp}}(e_{ij}) \tag{5}$$

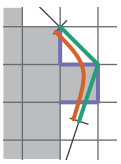
Optimization. Classical shortest cycle computations account for weights of edges, but do not allow for the evaluation of costs of pairs or triplets of edges in a cycle. We therefore reformulate the shortest cycle computation on G as the problem of finding the shortest cycle on a graph $\tilde{G} = (\tilde{N}, \tilde{E})$ whose nodes $\tilde{n}_{ijk}$ correspond to triplets of vertices v_i, v_j, v_k in the original graph G. We introduce a node $\tilde{n}_{ijk}$ into this graph if the original graph G contains the edges $e_{ij} = (v_i, v_j)$ and $e_{jk} = (v_j, v_k)$. We then connect each node pair $\tilde{n}_{ijk}$ and $\tilde{n}_{jkl}$ with an edge $\tilde{e}_{ijkl}$. Note that, by construction, these edges correspond to quadruplets of vertices in the original graph that are connected by sequences of three edges. We define the cost of each such edge as

$$C(\tilde{e}_{ijkl}) = f_{\text{acc}}(e_{ij}) + f_{\text{cont}}(e_{ij}, e_{jk}) + \omega_{\text{cv}} f_{\text{cv}}(e_{ij}, e_{jk}, e_{kl}) + f_{\text{simp}}(e_{ij})$$

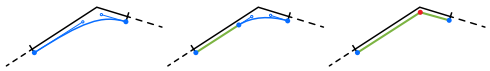
The shortest cycle on this graph defines a cycle on the original graph that minimizes the polygon cost $C(P)$.

5 SPLINE FITTING

The second stage of our algorithm fits a closed, piecewise smooth spline S to the extracted polygon such that the resulting outline is perceptually consistent with the input raster boundary R. Computing this fit requires solving for several sets of variables: the sequence of primitives (defined by their type) that jointly define the spline, the mapping between the primitives and the raster boundary segments they fit to (specifically the correspondences between primitive endpoint and raster boundary locations), and finally the shape parameters of these primitives (control point locations). Methods such as [Hoshyari et al. 2018] iteratively update the different sets of variables through trial and error until a satisfactory solution is found, a highly time-consuming process. We uncouple the computation of the three sets of variables and enable a fitting method that is both effective and fast by utilizing the geometric information provided by the intermediate polygon.



We recall that both the polygon and the spline are expected to accurately approximate the input raster and that the tangents of the intermediate polygon edges are expected to strongly correlate with the tangents of the final spline. In particular, we expect the spline to pass close to polygon-edge midpoints and for spline tangents in the vicinity of these midpoints to be similar to the edge tangents. This observation suggests using edge midpoints



(a) Single-Curve (b) Curve-Line (c) Line-Line

Fig. 7. The three primitive configuration types used for a polygon section around a corner. Control points shown as disks.

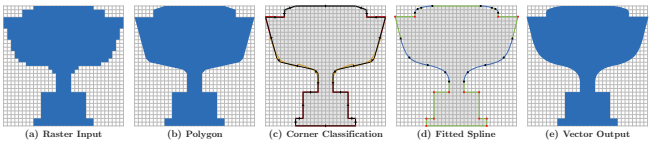


Fig. 8. (a,b) raster input and fitted polygon; (c) corner classification (single curve - orange, curve-line - grey, line-line red), (d) final fitted spline with curves colorized based on type, (e) vector output.

as natural G^2 transition points between primitives and constraining primitive tangents at these midpoints to align with polygon edge tangents. In the following, we refer to each polygon section between consecutive midpoints as a *polygon corner* (see inset green) and denote their corresponding raster segments and spline sections as *corner segments* (purple) and *corner sections* (orange), respectively. We further note that both polygonal and spline fits are expected to provide a similar balance between continuity and simplicity. Thus, we expect corner spline sections to be at least as continuous as their matching polygon corners (namely to have at most one C^0 discontinuity) and expect them to respect the upper bound on the number of primitives imposed by the polygon corner (i. e., two).

Following these observations, we identify three corner-spanning primitive configurations that reflect all different balance choices between simplicity and continuity and that satisfy the criteria above (Sec. 5.1, Fig. 7). We obtain the sequence of primitives that best approximates the input raster boundary by searching for the primitive configurations at all polygon corners that are best aligned with human expectations. We note that assessing all possible primitive configurations over the entire spline at once is intractable as the runtime would grow exponentially with the number of polygon corners. Instead, we successfully compute the desired primitive sequence by assigning a primitive configuration to each spline corner section independently using a learned classifier (Sec. 5.2, Figs. 5d, 8b). To perform the classification we fit the three different primitive configurations to each polygon corner and measure the compatibility of each configuration to this polygon corner and its underlying raster segment across a number of criteria. These measurements are then used as input for the classifier that determines the best configuration. The local spline fits we use to obtain the measurements balance tangent and positional accuracy terms subject to different boundary conditions as discussed in Sec. 5.3. Given the finalized sequence of primitives, we compute a globally consistent spline that balances accuracy, tangent alignment, and fairness (minimal curvature variation) by optimizing the control points of all curve primitives (Sec. 5.3, Fig. 8d).

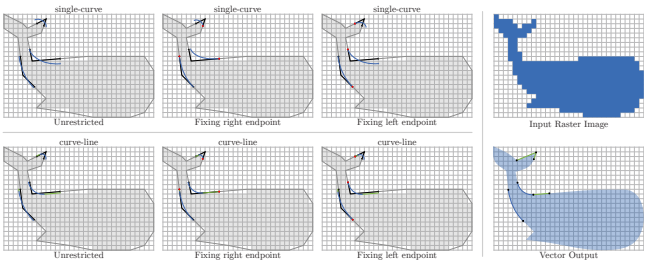


Fig. 9. Representative fits used to assess the suitability of the single-curve (top row) and curve-line (bottom row) primitive configurations. From left to right: basic boundary constraints, fixing left endpoint, fixing right endpoint.

5.1 Primitive Configuration Candidates

We expect the primitive combinations fit to each corner to be at least as continuous and as simple as the polygon corners themselves. Thus we enforce the spline corner sections to have at most one discontinuity and to consist of at most two primitives. We require the spline shape to mimic that of the corresponding polygon corner: spline tangents at section endpoints should be similar to the corresponding edge tangents and the spline tangent near the polygon corner should be close to the linear average of these two tangents. These expectations lead us to use the following local fitting primitive combinations for each spline corner section (Figure 3): (1) a single cubic Bézier curve whose tangents approximately linearly interpolate the end-point and corner tangents (the most continuous and simple but potentially least accurate solution); (2) a pair of straight lines aligned with the endpoint tangents (simple, sufficiently accurate, but discontinuous solution); and (3) a curve-line combination, where one Bézier curve end point is placed next to the midpoint closest to the corner and the other is placed symmetrically along the other edge, the line's tangent matches the midpoint tangent, and the Bézier curve smoothly interpolates between the midpoint tangents. The last alternative has the same primitive count as the polygon corner, is G^1 continuous, and is likely to be somewhere between the other two primitive combinations in terms of accuracy.

5.2 Primitive Configuration Classification

Our overall goal is to produce a spline that best balances smoothness, simplicity, and accuracy with respect to the input raster. To find such a spline, during classification we consider the configuration types using an ordering that prioritizes continuity over simplicity. Whenever possible, we use the single-curve configuration and downgrade to line-curve, and subsequently to line-line only when the first attempted fits are deemed *inadequate*.

When assessing adequacy locally, we lack the global context (position and tangent boundary conditions at section endpoints) that comes into play when fitting an entire spline. To account for the global nature of the fitting problem when determining the adequacy of each primitive configuration, instead of only assessing one possible fit, we examine three representative fits – each with different boundary conditions; we then base our decision on the measurements taken across all three fits.

In the first representative fit, we allow both section endpoints to move orthogonally to their corresponding polygon edges and fix

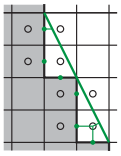
the end tangent directions to those of the polygon. In the other two fits, in addition to these constraints, we fix either the left or right endpoint position to the respective polygon edge midpoint. Jointly, these three options account for plausible additional constraints that are imposed by neighboring corner sections during the global fit. Figure 9 shows these representative fits for an example model.

We use these representative fits to obtain a list of features, which we then utilize to categorize the configurations for each corner section into *adequate* and *inadequate* using a random forest classifier [Breiman 2001]. We use the Random Forest implementation provided by Andres *et al.* [2016]. The classifier receives features from all representative fits at the same time and produces a single binary label. In selecting the features, we seek to account for cues human observers are likely to use during mental vectorization.

(1) For each smooth fit we include its accuracy (computed as described next) as well as the difference between the accuracy of the smooth and underlying polygonal fits (6 values). We speculate that both the absolute and relative accuracy play a role – viewers would be more likely to discard a fit if it is objectively inaccurate, or if it is far less accurate than the less continuous alternative.
(2) We also include the maximal curvature along each fitted spline (3 values). We expect viewers to prefer less continuous but lower curvature solutions when the curvature of the continuous solutions grows.
(3) Lastly, we provide the polygon corner angle itself (1 value), as we expect it to strongly correlate with viewer's mental balance of continuity and simplicity.

This choice gives a total of ten features, from which the classifier computes its categorization. Notably, we do not provide the classifier with internal details of the fit such as the specific primitive configuration or raster resolution. This makes the categorization independent of any specific fitting strategy, and uses a common coordinate frame defined by the raster resolution across all image sizes, and thus more general and robust.

In measuring accuracy, we perform a more fine-grained analysis than in the polygon fitting stage. We measure both the degree to which the vector output violates the in/out pixel-center classification imposed by the raster boundary and the actual distance between the raster and the vectorization or polygon fits. This fine-grained measurement is motivated by the WYSIWYG principle, which indicates that humans prefer interpretations closely aligned with the data observed [Wagemans et al. 2012]. More specifically, for each pixel edge on the corresponding raster boundary, we use the axis-aligned distance of its midpoint to the vectorization or polygon edge as the pixel's accuracy measure if the pixel center is correctly classified with respect to the line or spline (see inset top pixel). If the pixel has the wrong in/out center classification, we consider the Euclidean distance between the pixel center and the vectorization or polygon edge (Eq. 1) (see inset bottom pixel). More formally, given the axis-aligned distance from the pixel midpoint $d_\perp$ and the Euclidean distance from the pixel center $d_\nearrow$:



$$d = \begin{cases} d_\perp & \text{if center classification is correct} \\ 0.5 + d_\nearrow & \text{otherwise.} \end{cases} \tag{6}$$

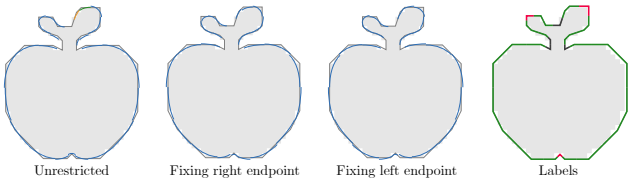


Fig. 10. Training model with representative fits for the Curve configuration type (first three columns) with annotations (last column). The fits for each corner in the polygon can be *Adequate* (red), *Inadequate* (red) or missing due to regularities (gray).

Note that this accuracy metric allows a simple test to determine if the pixel center classification is correct (by comparing with 0.5). Given this per-pixel accuracy metric, we define the vectorization's or polygon edge's accuracy as the maximum over all corresponding raster pixels.

We use the classifier to determine if each of the curve and curve-line configuration is adequate (we view the line-line as the fallback, always adequate option). If the curve configuration is deemed adequate we use it, otherwise we fall-back to the curve-line if it is deemed adequate, and fallback to line-line otherwise. In cases where the classifier deems the curve configuration as adequate but with a low confidence (probability $< 75\,\%$), we select the curve-line configuration if it is deemed *adequate* with a high confidence (probability $\geq 75\,\%$). We found that using confidence in addition to the binary decision leads to more robust classifications.

After finding the primitive types for each section, we perform a global spline fit (Section 5.3). We perform a final adequacy check on this global fit by feeding the actual fit measurements to our classifier (as three equal representative fits). If the classifier deems any of the corner sections inadequate, which can happen occasionally due to the more accurate boundary conditions, we replace the corresponding section configuration with the next one in our preference order.

Training and Validation. We trained a forest with 100 trees on manually annotated polygon corners across 23 input images at different resolutions, exhibiting a wide variety of geometric configurations. Figure 10 shows an example of a training model with a red or green label for *inadequate* and *adequate* Curve fits, the complete set is provided in the supplementary material. Thanks to the generic design of our feature space, this small training dataset is sufficient to train a robust classifier, in contrast to Hoshyari *et al.* [2018], who require more than 100 annotated input images and train separate classifiers for different resolutions.

Our training does not limit the depth of trees in the random forest; however, in practice it produces naturally compact trees with an average of 40 leaves per tree, which is further evidence that the description of corner fit properties using the proposed features generalizes well. This observation is further supported by the validation test we performed on the classifier. From all the samples we annotated, we use 50% for training and 50% for testing. Our trained classifier achieves an accuracy of 99.3% on the test data set. In addition to a random forest, we also performed tests with a linear classifier and a multi-layer perceptron (MLP) neural network.

Both achieve very similar classification precision on our test set with only one misclassification more than the random forest. We chose the random forest because it is relatively simple, expressive, and robust, and unlike MLP, requires little parameter tuning and no feature normalization.

5.3 Spline Shape Optimization

Our framework performs two types of spline fitting: individual per-corner section fits using different primitive combination are computed during the corner classification stage, and a global fit of a known (closed) sequence of primitives is performed at the end to produce the final result. We first explain how we model the individual classification fits and then describe the modifications for the final fit.

When computing the per-corner fits, we seek spline degrees of freedom (i. e., control points) that minimize the following objective function:

$$\begin{aligned} \arg\min_{S} \; & F_s(S) = \omega_{\text{sacc}} \cdot f_{\text{sacc}}(S) + \omega_{\text{tang}} \cdot f_{\text{tang}}(S) \\ & \text{s.t. } G^1 \text{ continuity between consecutive primitives,} \end{aligned} \tag{7}$$

where f_{sacc} promotes proximity to the input raster and f_{tang} promotes alignment with polygon tangents. We solve this problem using the Levenberg-Marquardt algorithm[Nocedal and Wright 2006] while enforcing constraints by appropriate elimination of variables.

Initialization. The iterative Levenberg-Marquardt algorithm requires an initial guess. Furthermore, as we outline below, we define the constituent energies based on a set of reference primitive curves. For both purposes, we require a set of initial curves that is sufficiently close to the optimal solution. We find these initial curves using an analytic construction described in Appendix B.

Accuracy Energy. To assess a spline's accuracy, we sample the raster boundary corresponding to the spline at pixel midpoints M, and measure the distances from these samples to the spline. For an efficient optimization, we project all midpoints onto the initial curves, resulting in pairs of points and corresponding curve parameter values (p, t). We then linearize the objective, setting

$$f_{\text{sacc}}(S) = \frac{1}{|M|} \sum_{(p,t)\in M} (S(t) - p)^2 \tag{8}$$

This energy promotes closeness to the raster boundary. Since the raster is only a very coarse approximation of the vectorized shape, we assign a relatively small weight of $\omega_{\text{sacc}} = 0.01$.

Tangent Energy. We employ a similar sampling-based approach to promote tangent alignment. We define a tangent objective where we prescribe tangents at polygon edge midpoints, corners, and between them. To this end, we first calculate an equidistant sampling T of the initial curves and assign target tangents that we derive from linearly interpolating the nearest midpoint and corner angles using arc length. Dense even sampling provides better control over the spline shape compared to sampling at projections of pixel corners or midpoints which is by default less even and often more sparse.

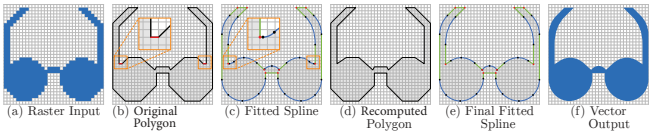


Fig. 11. Left to right: raster input, polygon (with accidental edges highlighted), fitted spline, simplified polygon, and final fit.

We gather these tangents T consisting of tangent directions d and parameter values t and set

$$f_{\text{tang}}(S) = \frac{1}{|T|} \sum_{(d,t)\in T} \left(\frac{\dot{S}(t)}{\left\|\dot{S}(t)\right\|} - d \right)^2 \tag{9}$$

Since these tangents are much more indicative of the expected vectorization than the raster samples, we assign tangent alignment a significantly higher weight of $\omega_{\text{tang}} = 1$.

Final Fit. The goal of the final fit is to produce a fitted spline consisting of a known set of primitives that best balances accuracy, tangent alignment, and simplicity (minimal curvature variation). To account for simplicity across all primitives we augment the optimized energy F_s (Eq. 7) with an additional *shape* energy term f_{shape} which minimizes curvature variation across all curve primitives and enforces soft G^2 constraints between them:

$$f_{\text{shape}}(S) = f_{\text{fair}}(S) + f_{G^2}(S) \tag{10}$$

We measure curvature variation using the discretization presented by Lu *et al.* [2018] ($\tilde{\mathcal{E}}_{CVE}$, Equation (8)) and sum over all Bèzier curves $f_{\text{fair}}(S) = \sum_{b\in S} \tilde{\mathcal{E}}_{CVE}(b)$. To target G^2 continuity, we find all spline parameter values T_{G^2} at which two Bèzier curves meet and penalize curvature differences:

$$f_{G^2}(S) = \sum_{t\in T_{G^2}} \left(\frac{\kappa_S(t-) - \kappa_S(t+)}{|\kappa_S(t-)| + |\kappa_S(t+)| + \epsilon} \right)^2, \tag{11}$$

where $\kappa_S(t-)$ and $\kappa_S(t+)$ are the left- and right-sided curvatures at t, respectively, and $\epsilon = 10^{-3}$ is a small number to avoid division by zero. Note that the summands of f_{G^2} are approximately 1 whenever the two curvatures have opposite signs. This reflects the desire to penalize curvature differences for curves with the same sign as reducing those improves curve simplicity. In inflection scenarios, however, the change in sign by itself is already disruptive and the degree of disruptiveness does not significantly change as the curvature magnitudes change. For the shape energy, we set a weight $\omega_{\text{shape}} = 0.1$ that is between ω_{acc} and ω_{tang}.

Corner Feedback Loop. Our polygon computation aims to maximize continuity and minimize curvature change across all polygon corners. Thus, once our classifier determines the C^0 corners we could theoretically rerun the polygon computation to obtain a simpler solution by not optimizing either criterion at such discontinuous corners. However, such iterative processing is both time-consuming and, in general, unnecessary as it rarely impacts the cycle choices. In our experiments the only instances where such iterated processing meaningfully impacted the results were when the C^0 corner was adjacent to short (two pixel or less) edges. Even in these cases the impact on the cycle was limited to the immediate vicinity of the

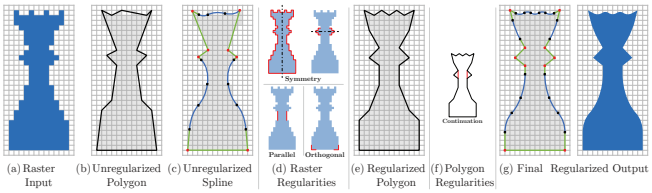


Fig. 12. We first identify raster regularities (a) and use them to modify the unregularized polygon (b) producing a more regular polygon (e). We combine detected raster regularities with regularities detected on the polygon (f) to regularize the output spline (g).

corners. Consequently, to avoid expensive and redundant recomputation once corners are detected, we only adjust the cycle locally and only in the presence of *accidental* edges (ones of length two or less). In this local path computation, we fix all polygon edges except a small neighborhood of three edges around the accidental edge. We then run a shortest path extraction on the graph G between these bounding edges with a modified cost (Eq. 5) that does not include continuity and curvature variation costs across the C^0 corner and recompute the spline fit (Figure 11).

6 REGULARIZATION

Perceptual and graphics research suggest that human observers recognize regularities in input data, and expect those to be preserved during fitting or vectorization [Hoshyari et al. 2018; Li et al. 2011; Mehra et al. 2009]. We therefore aim to preserve input regularities in our final output. Similar to prior work [Hoshyari et al. 2018; Li et al. 2011; Mehra et al. 2009], we focus on symmetries, parallelism, continuations, and axis-alignment. To differentiate between accidental and non-accidental raster and polygon level regularities, we only consider axis-aligned edges in the regularity computation if they are at least two pixels long. When regularity conflicts with accuracy or simplicity, we prioritize regularity over the other cues, unless stated otherwise. Since regularity enforcement needs to be exact, regularities cannot be addressed by adding soft terms to our polygon of spline fitting energies. Instead, we enforce regularities at the polygon level using a combination of graph pruning and polygon modification. At the spline level, we enforce regularities in two ways: we modify the primitive configuration classifications to reflect detected regularities, and add hard constraints to the final fit optimization when necessary. To keep our classifier training robust and general, we remove all samples from the training and test data set that are affected by regularities that impact classification.

Orthogonal and Axis-Aligned Edges. We orthogonalize polygon corners where one edge is non-accidental and already axis-aligned, if this operation does not decrease accuracy, or simplicity (measured via inflection count). During fitting, we make axis aligned edges whose lengths are at least 50 % of the extent of the respective bounding box side more prominent by disallowing the use of (single) curve primitive configurations for the two incident polygon corners (thus forcing the method to use the curve-line or line-line configuration along these edges).

Parallel Edges. We detect and enforce prominent axis-aligned parallel edges present in the raster input (we consider the edges as prominent if the distance between them is no longer than their overlap length) if the parallelism constraint is unenforceable at the fitting level otherwise (i.e. their slopes are not a priori in the same quadrant).

In the fitting stage, we classify edges as parallel using the distance criterion above and an angle threshold of 20° [Hess and Field 1999], and constrain the tangents of line segments corresponding to these polygon edges to the average polygon edge direction. Furthermore, to account for distinctly visible translational symmetries, if a pair of parallel edges is connected by exactly one other polygon edge, we make the primitive configurations of this edge's corners equal (choosing the lower continuity one if they differ).

Symmetries. Raster data can only capture exact symmetries whose reflection planes have an orientation of integer multiples of 45°. We detect and attempt to enforce all such symmetries at the polygon level. If two raster symmetries conflict, we prioritize symmetries present *a priori* in the unregularized polygon, and prioritize longer symmetric boundary paths over shorter ones. We prioritize simplicity over symmetry, since raster symmetries are inherently noisy. In the fitting stage, we equalize section classifications for polygon corners that are associated to each other via a raster symmetry by downgrading the one with higher priority. Further handling in the final fitting stage is not necessary as symmetric fits emerge naturally from the fitting formulation.

Continuations. Continuations [Wagemans et al. 2012] are parts of a boundary that are interrupted by a protrusion or other part that does not logically belong to the base shape (see inset). In all such cases, viewers expect the interrupted curve to be continuous at the expense of making the vector outline discontinuous at protrusion end-points. We detect and enforce continuations both at the raster level (axis-aligned) and at the polygon level (arbitrary orientation).

We preserve axis-aligned continuations at the raster level, when the edges incident to the participating corners are non-accidental by forcing all cycles to pass through the continuation vertices.

We detect continuations of arbitrary orientation using the empty-circle and same-line test from [Hoshyari et al. 2018]. Instead of only considering pairs of closest concave corners, we follow perception literature [Field et al. 1993] and allow continuations between all pairs of concave corners if the continuation does not exceed a total angle of 60°. While locating continuations, we also consider those that would result from moving polygon corners by one pixel in either direction along the raster boundary. We greedily choose the continuations that result in flatter continuation angles and relocate the polygon vertex if a continuation with an offset was chosen, effectively aligning the vertex pair connected by the continuation to each other. In the final fitting stage, since the protrusion is effectively separated from the base shape by the continuation, we assign the line-line primitive configuration to the participating polygon corners.

7 MULTICOLOR INPUTS

For simplicity, our description so far addressed the baseline scenario of a drawing consisting of two uniformly colored regions separated

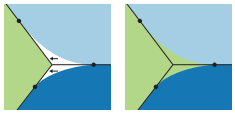
(a) Raster Input (b) Separate Region Vectorizations (c) With Junction Resolution

Fig. 13. Multicolor input processing: (a) raster input, (b) result of processing each region boundary independently, (c) our result with consistent junction resolution.

by a single closed boundary. Real-life images however typically contain multiple colored regions.

When extending our framework to handle such data, we need to ensure fitting consistency along shared boundaries, and in particular resolve junctions where more than two regions meet. To enforce these constraints at the polygon level, we first constrain the polygons to go through the junctions. We apply this constraint for each region independently by adding junction vertices to the graph vertex set V, even if these are flat, and removing all graph edges that bypass them. We then unify the polygon subpaths between junctions shared by pairs of regions, by selecting the subpath candidate with lower polygon cost (Eq. 5) and using it for both regions. This process results in a consistent polygonal network that approximates the input raster.

We compute the spline primitive classifications for each region separately and then enforce the following consistency rules. We observe that at both valence three and valence four junctions (the only two possible on raster data) we can have at most two continuous spline pairs. Thus if a region classifies the junction as a smooth primitive configuration type (either single-curve or curve-line), at least one neighboring region needs to classify it as a line-line (C^0 corner) configuration. We further note that adding flat vertices to the polygon edge graph, can generate false positives, making the polygon appear smoother than it is. We therefore allow two neighboring regions to both classify the junction as a smooth configuration only if vectorizing the regions separately would generate a polygon that uses the junction as a vertex with a smooth configuration type. If this is not the case, we view the smooth configuration as likely false positive and only allow at most one of the two neighboring regions to be classified as smooth. In practice, we found that this criterion can be approximated to a sufficient degree of accuracy by testing if the polygon edge incident to both regions is axis-aligned. If any of the above criteria are not fulfilled, we adjust the section classification of the most acute polygon section around the junction to the line-line configuration until all criteria are met. For all other valence-2 sections shared between two regions, we unify their classifications by picking the less continuous one if they differ (this can happen due to different regularities being present in the two regions).



Finally, we perform a global fit for all primitives across all regions. We constrain primitives along the same polygon subpaths away from the junctions to have the same geometry for both regions. To produce a water-tight output, we replace line primitives of a line-line configuration of a junction section with the primitives of a neighboring smooth fit (see inset, black lines denote polygon edges, black dots denote edge midpoints). This junction resolution still adheres to the continuity choices made locally and guarantees gap- and overlap-free vectorizations.

8 RESULTS AND VALIDATION

We have tested our method on 143 diverse inputs, including both man-made and organic clip-art shapes, across a range of resolutions. 43 of these are shown in the paper and the rest are included in the supplementary material. We assessed its performance on inputs used by the closest prior method of [Hoshyari et al. 2018], as well as on 86 new inputs collected from numerous repositories (e. g., Figures 14, 16). Our test set contains 52 images consisting of two regions separated by a single closed boundary as well as 91 multi-color examples. For images with large single color regions, simply enforcing accuracy is often sufficient to generate adequate vectorizations [Hoshyari et al. 2018]. Thus, the more sophisticated mechanism proposed in our work is most necessary for vectorizing low- to medium-resolution data, where accuracy alone is not a sufficient criterion. Our test-set is consequently dominated by inputs with region resolutions below 100×100, with many inputs at low resolutions of 32×32 and less. Visual inspection confirms that our results are consistently well aligned with viewer expectations.

Comparison to Prior Work. We quantitatively asses our results by comparing them to algorithmic alternatives via a comparative perceptual study. Study participants were shown input images, together with our result and an alternative or ground truth result using the following layout. The input was shown at the top and marked as 'A', and the two vectorizations were placed at the bottom and marked as 'B' and 'C'. Participants were then asked to select the result that resembles the input the most: "Which of the two images on the bottom 'B' or 'C' better represents image 'A'? If both are equally good then select 'Both', and if neither represent 'A' then select 'Neither' ". The answer options were "B", "C", "Both", and "Neither". We collected answers for each query from ten different participants using the protocol described in Appendix A. The results of the study are detailed next. All study data is provided in the supplementary.

In the study, we compared our results to those produced by the two most highly ranked methods in the recent study of Hoshyari *et al.* [2018] across 81 inputs (Potrace [Selinger 2003], and [Hoshyari et al. 2018]). For completeness, we also included 28 comparisons to the popular method of [Kopf and Lischinski 2011] on low-resolution color data, the target domain of their method, plus the simple examples in Figure 2. The test dataset consisted of 37 inputs with a single region boundary, 28 low-resolution multi-color inputs, and 16 multi-color inputs of medium to high resolution. We used default parameters for all methods. Representative examples of all methods across different resolutions are shown in Figures 1, 2, and 14. Our comparisons focus on inputs with region resolutions of up to 100 since, as noted earlier, for higher resolutions, fitting accuracy dominates all other cues, with all methods producing fairly similar results. At lower resolutions, however, especially 16×16 and below, our method clearly dominates the alternatives and was preferred in 94% of the cases over the results of [Hoshyari et al. 2018]. We, furthermore, conducted a t-test on the study results, which shows

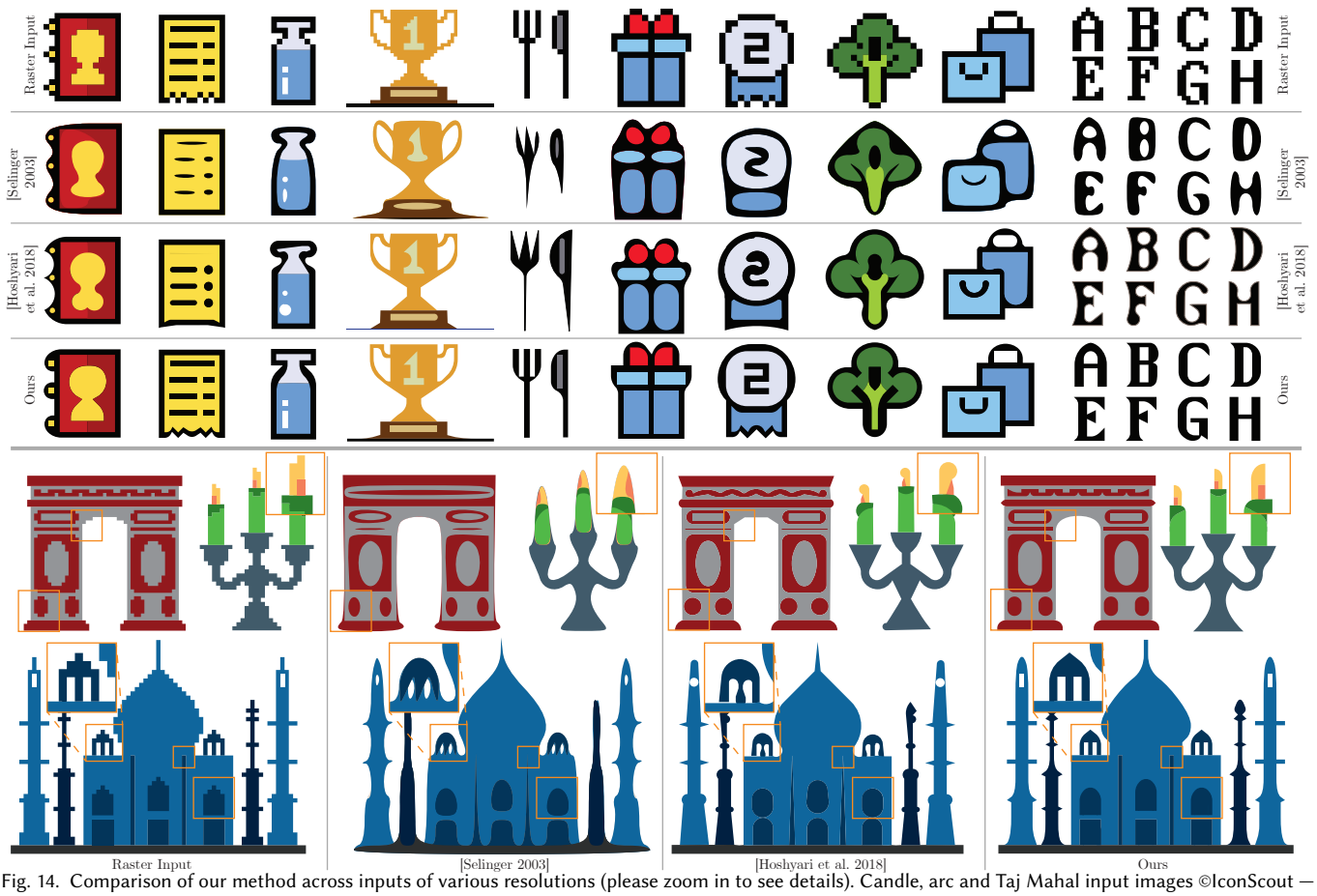


Fig. 14. Comparison of our method across inputs of various resolutions (please zoom in to see details). Candle, arc and Taj Mahal input images ©IconScout — www.iconscout.com

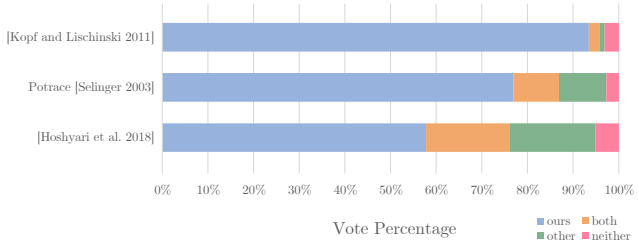


Fig. 15. The percentages of user preference in our study. Each row shows the results of comparing our results to those of the respective alternative method.

that our method's improvement over each of the prior methods is highly statistically significant.

Comparisons to Manual Vectorization. Manually vectorizing multi-color inputs is extremely time consuming (as a single region takes over 30 minutes to fit accurately [Hoshyari et al. 2018]). We consequently only test our method against 15 vectorized binary images provided by the authors of [Hoshyari et al. 2018]. Figure 17 shows three side-by-side comparisons of artist-created and our results. All artist images are included in the supplementary material. Overall, participants ranked our results as on par or better than those created by the artist 55% of the time. When analyzing the inputs on which artist results were ranked as superior (e. g., Figure 17c) we conjecture that this preference is due to recognition - as the artist outputs reflect the author's knowledge of the depicted shapes.

Non-Uniform Shading. Our focus is on inputs with distinct, uniquely defined boundaries, thus our examples are dominated by inputs where each region has a uniform color. However, many clip art images have varying shading near inter-region boundaries, e. g., due to anti-aliased rasterization (Figure 18a). In this case identifying the optimal location for the boundary itself, becomes a challenge. To extend our method to handle such data we modify the graph construction and accuracy terms used for polygon extraction and smooth fitting.

We compute region boundaries to use as a starting point for our algorithm, as follows. We observe that the region boundary we use for binary data is an isoline at the average color of two neighboring regions. This definition directly generalizes to non-uniformly shaded data. We locate all intersections of the isoline with the pixel grid, similar to the first stage of the Marching Squares algorithm. Since

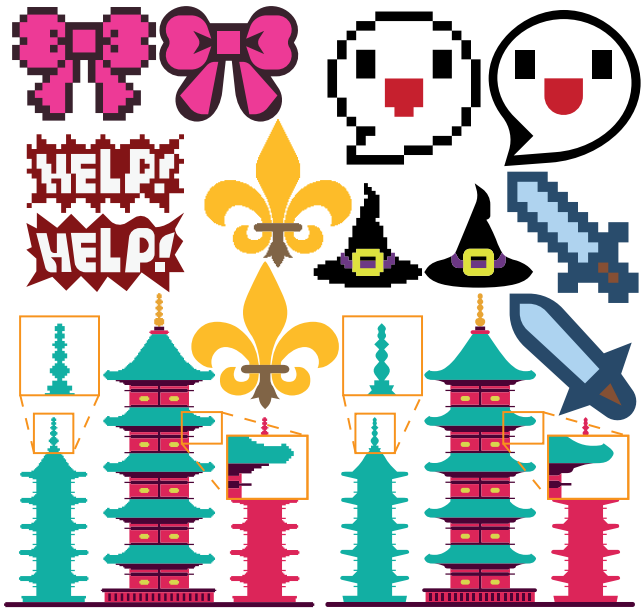


Fig. 16. Additional multi-color results across multiple resolutions. Building image ©IconScout — www.iconscout.com

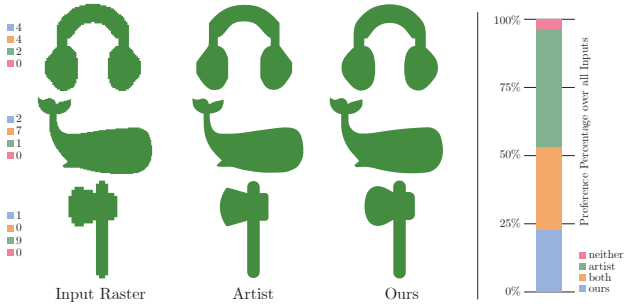


Fig. 17. Comparison against artist-produced vectorizations (left) and the distribution of votes from our user study (right). While viewers preferred our results on the top two inputs, they preferred the artist result on the raster axe (inset numbers).

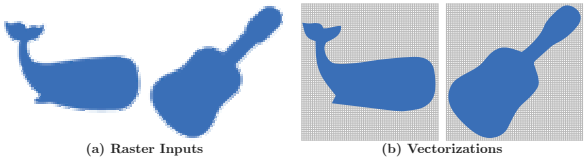


Fig. 18. Vectorizations (right) of raster inputs (left) with fuzzy (anti-aliased) boundaries.

the rest of our algorithm expects axis-aligned boundaries, and in order to preserve sharp features, we then connect these points with axis-aligned edges to form each region boundary.

Our accuracy term (see Section 4.1) is based on the in-out partitioning of pixels imposed by a graph edge and intuitively measures the distance that a pixel has to be moved to lie in the correct partition.

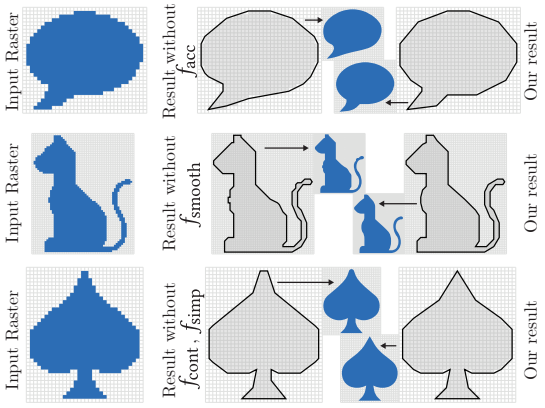


Fig. 19. Polygon ablation study: (Top three rows) Raster input (left), polygons and vectorizations obtained without using accuracy, smoothness, or simplicity cues (middle), and our result using the full energy.

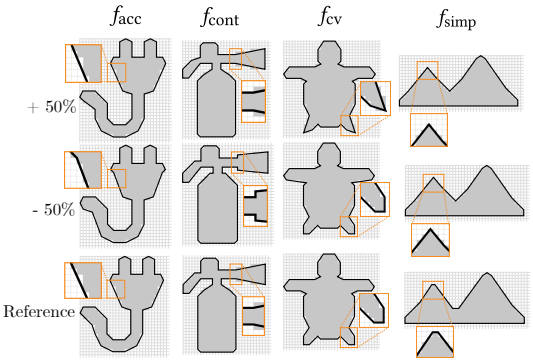
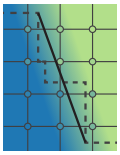


Fig. 20. For each term in Eq. 5 (columns) we show the polygon obtained using the reference weights (row four), increasing each energy's weight by 50% (row five) and decreasing it by 50% (last row).



In non-uniformly shaded inputs, this partitioning is not binary but continuous: the line imposes a linear color gradient between the region colors onto the pixel space (see inset).

Using this imposed gradient, we can equivalently define the pixel accuracy as the distance to the closest location whose imposed line gradient color equals the pixel color. Since the gradient bandwidth b depends on the rasterization method, we use separate per-pixel estimates that we derive from the pixel color and its distance d to the isoline: $b = d \cdot \frac{1}{2\Delta c}$, where $\Delta c \in [0, 0.5]$ is the pixel color difference to the isoline color, normalized by the region color difference. Doing so, we achieve zero accuracy error for lines that coincide with the isoline. Equivalently to the uniform shading case, the accuracy error of a graph edge is the maximum pixel accuracy error of all pixels around the isoline whose color is not one of the two region colors.

For the accuracy term in spline fitting (8), equivalently to the uniform shading case, we sample midpoints of region boundary edges and use those points in the respective accuracy formulations.

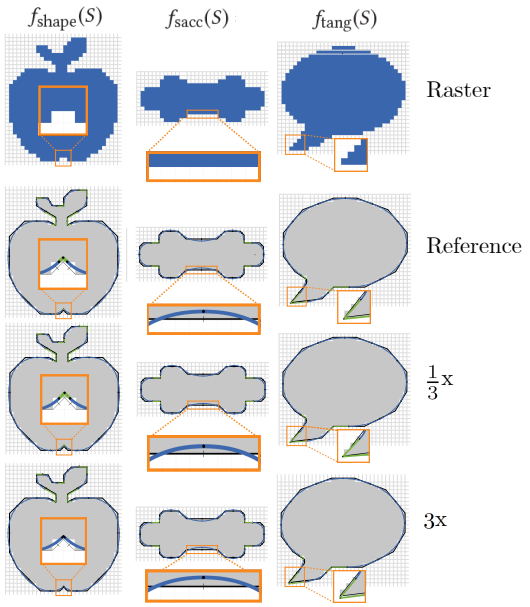


Fig. 21. For each term in Eq. 7 we show the final fit obtained using the reference weights (row two), decreasing each energy's weight to one third (row three) or increasing each energy's weight three times.

(a) Input Raster (b) Polygon (c) No Single-Curve (d) No Curve-Line (e) Our Result

Fig. 22. Fitting ablation study: Raster input (a), polygon (b), and vectorizations obtained without using one of the primitive configurations (c-d) and our result (e) using all configurations

Ablation Studies. Figures 19, 21 and 22 show the impact of the different energy terms in our optimized functions. The three rows in Figure 19 show the impact of dropping one of our three perceptual cues (accuracy, smoothness, simplicity) on the intermediate polygons. Figure 20 show the impact of reducing and increasing the weight of the terms corresponding to each cue by 50%. Increasing accuracy promotes closeness to the boundary, resulting in a polygon matching the raster boundary more closely. Increasing the continuity term results in flatter angles being preferred over more raster-aligned ones (e.g., plug's cord). Reducing the incentive to minimize curvature variation produces sharp features instead of smoother ones (e.g., turtle feet). Finally, simplicity proportionally affects the presence of short edges. Figure 21 show the impact of reducing (one third) and increasing(thrice) the weight of the terms used in the smooth fitting. The reference weights (third column) has fits better balancing the different energies, but the changes introduced in the first and second column are overall minor. Figure 22 shows the impact of reducing the number of primitive configurations. As demonstrated, the outputs produced with all features in place are significantly better than the alternatives.

Performance. Our method takes 0.5 and 1.2 seconds to vectorize a single region at resolutions of 32×32 and 64×64, respectively (medians across our binary dataset). The majority of the runtime is distributed approximately equally between the initial spline fitting and the subsequent corner feedback loop. The polygon extraction stage takes on average between 10% and 20% of the total runtime and increases as more regularities are present in the input, which require more regularization work on the polygon. The times are measured on an 8th Generation Intel Core i7 CPU at 3.7 GHz. These times are dramatically faster than those of [Hoshyari et al. 2018] who take on average between 30 and 50 times as long.

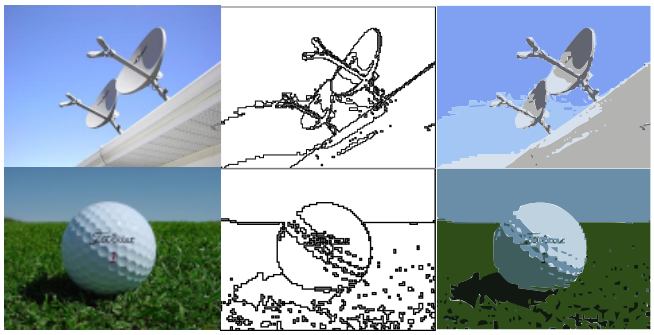


Fig. 23. Vectorization of segmented photographs from the MSRA10k dataset [Cheng et al. 2015]: (left to right) photo, segmentation output, vectorization.

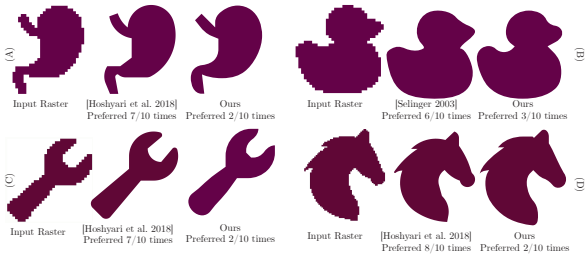


Fig. 24. Representative pairs of our and alternative method outputs where our results were consistently judged as inferior and the respective user preferences.

Natural Inputs. While our method is designed for clip-art we had tested it on pre-segmented photographic images (Figure 23). While the generated results are less regular, they appear overall well aligned with observer expectations.

Limitations. Since our goal is to produce results aligned with human perception, we consider it a failure when viewers consistently prefer an alternative result over ours. Across all the comparisons made our method failed on only 9 inputs; measured as cases where users preferred the alternative result more often than either ours or both (see Figure 24 and the supplementary material). Notably, all but two of these are 32 and 64 pixel black on white silhouette images that were used by prior work [Hoshyari et al. 2018], which is optimized for this exact data. There is no clear point of failure for these inputs – some of the differences can be attributed to different regularization strategies or different choice of fitting primitives, while others appear subjective (e.g., Fig. 24 (D)).

Our current implementation only regularizes individual boundaries. It achieves inter-boundary parallelism by default for many

cases. Explicit detection and enforcement of such parallelism would be an important practical extension of our method. In the future, one can similarly extend our regularization step to address regularities between regions, e.g., using continuation detection, complementing our core method.

9 CONCLUSION

We presented PolyFit a new method for clip-art vectorization, and demonstrated it to produce results better aligned with user expectations, compared to the existing alternatives. Our method performs particularly well on lower resolution data, where individual regions have dimensions of 32×32 or less. Contrary to prior work it does not require separate training for different resolutions and is significantly faster than the closest alternative. Key to the success of our method is the two-step processing, where we compute a polygonal fit using a perception-motivated energy function, and then use the output of this fit as input to a learned classifier, that facilitates our final vectorization.

Our work opens the door for several interesting follow-ups. Our method uses simple criteria for detecting and enforcing regularities; viewer-perceived regularity detection on raster data is an important stand-alone problem that merits further research. One of the most interesting problems in this context is detecting human-perceived regularities between different boundaries and regions. Human interpretation of raster clip-art is likely affected by object recognition; combining our method with recognition techniques can further improve vectorization outputs. Our method is sensitive to small changes in the raster, and to imperfections in regularities such as symmetries; human observers are capable of looking past such imperfections. Extending vectorization to support non pixel-perfect inputs is an important research direction. Lastly, more work is necessary to robustly handle anti-aliased inputs.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments, Enrique Rosales and Luciano S. Burla for their help at various stages of the project. This work has been supported by NSERC. Leonid Sigal is supported in part by NSERC Canada Research Chair, CIFAR AI Chair and the Vector Institute for AI.

REFERENCES

Adobe. 2017. Adobe Illustrator 2017: Image Trace. http://www.adobe.com/.

Bjoern Andres, Steffen Kirchhoff, and Evgeny Levinkov. 2016. A Fast C++ Implementation of Random Forests. https://github.com/bjoern-andres/random-forest

B. Aronov, T. Asano, N. Katoh, K. Mehlhorn, and T. Tokuyama. 2004. Polyline Fitting of Planar Points under Min-Sum Criteria. In *International Symposium on Algorithms and Computation*.

Ilya Baran, Jaakko Lehtinen, and Jovan Popović. 2010. Sketching clothoid splines using shortest paths. In *CGF*, Vol. 29,2. 655–664.

Richard Bellman and Robert Roth. 1969. Curve Fitting by Segmented Straight Lines. *J. Amer. Statist. Assoc.* 64, 327 (1969), 1079–1084.

Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. 1997. Geodesic Active Contours. *IJCV* 22, 1 (1997), 61–79.

Ming-Ming Cheng, Niloy J. Mitra, Xiaolei Huang, Philip H. S. Torr, and Shi-Min Hu. 2015. Global Contrast based Salient Region Detection. *IEEE TPAMI* 37, 3 (2015), 569–582. https://doi.org/10.1109/TPAMI.2014.2345401

Eagle. 1997. Eagle. http://everything2.com/index.pl?node_id=1859453.

Gerald Farin. 2002. *Curves and Surfaces for CAGD: A Practical Guide.* Morgan Kaufmann Publishers Inc.

M. Fatemi, A. Amini, L. Baboulaz, and M. Vetterli. 2016. Shapes From Pixels. *IEEE TIP* 25, 3 (2016), 1193–1206.

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. Simplicity: a Global Approach to Line Drawing Vectorization. *ACM SIGGRAPH* (2016).

David J. Field, Anthony Hayes, and Robert F. Hess. 1993. Contour integration by the human visual system: Evidence for a local "association field". *Vision Research* 33, 2 (1993), 173 – 193. https://doi.org/10.1016/0042-6989(93)90156-Q

M. A. T. Figueiredo, J. Leitão, and A. K. Jain. 2000. Unsupervised contour representation and estimation using B-splines and a minimum description length criterion. *IEEE TIP* 9, 6 (2000), 1075–187.

Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. 2005. Robust Moving Least-squares Fitting with Sharp Features. *ACM TOG* 24, 3 (2005), 544–552.

N. Goldberg, Y. Kim, S. Leyffer, and T. Veselka. 2014. Adaptively refining dynamic program for linear spline regression. *Computational Optimization and Applications* 58, 3 (2014), 523–541.

Robert Hess and David Field. 1999. Integration of contours: new insights. *Trends in cognitive sciences* 3, 12 (1999), 480–486.

Shayan Hoshyari, Edoardo Alberto Dominici, Alla Sheffer, Nathan Carr, Duygu Ceylan, Zhaowen Wang, and I-Chao Shen. 2018. Perception-Driven Semi-Structured Boundary Vectorization. *ACM Transaction on Graphics* 37, 4 (2018). https://doi.org/10.1145/3197517.3201312

Gašper Jaklič and Emil Žagar. 2011. Curvature variation minimizing cubic Hermite interpolants. *Appl. Math. Comput.* 218, 7 (2011), 3918–3924.

Xie Jun, Winnemöller Holger, Li Wilmot, and Schiller Stephen. 2017. Interactive Vectorization. In *Proceedings of SIGCHI 2017*. ACM.

Michael Kass, Andrew Witkin, and Demetri Terzopoulos. 1988. Snakes: Active contour models. *IJCV* 1, 4 (1988), 321–331.

K. Koffka. 1955. *Principles of Gestalt Psychology.* Routledge & K. Paul.

Johannes Kopf and Dani Lischinski. 2011. Depixelizing Pixel Art. *ACM TOG* 30, 4 (2011), 99:1–99:8.

Gregory Lecot and Bruno Levy. 2006. Ardeco: Automatic Region Detection and Conversion. In *EGSR*. 349–360.

Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. 2011. GlobFit: Consistently Fitting Primitives by Discovering Global Relations. *ACM TOG* 30, 4 (2011), 52:1–52:12.

Yang Liu and Wenping Wang. 2008. A Revisit to Least Squares Orthogonal Distance Fitting of Parametric Curves and Surfaces. In *Proc. Advances in Geometric Modeling and Processing*. 384–397.

Lizheng Lu, Chengkai Jiang, and Qianqian Hu. 2018. Planar cubic G1 and quintic G2 Hermite interpolations via curvature variation minimization. *Computers & Graphics* 70 (2018), 92–98.

Andrea Mazzoleni. 2001. Scale2x. http://www.scale2x.it/.

James McCrae and Karan Singh. 2008. Sketching Piecewise Clothoid Curves. In *Proc. Sketch-Based Interfaces and Modeling*.

James McCrae and Karan Singh. 2011. Neatening Sketched Strokes Using Piecewise French Curves. In *Proc. EG Symposium on Sketch-Based Interfaces and Modeling*. 141–148.

Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. 2009. Abstraction of Man-Made Shapes. *ACM TOG* 28, 5 (2009), 137:1–137:10.

J. Nocedal and S. Wright. 2006. *Numerical Optimization.* Springer New York. https://books.google.ca/books?id=VbHYoSyelFcC

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-shaded Images. *ACM TOG* 27, 3 (2008).

ScanFont. 2017. Font Lab, http://old.fontlab.com/font-converter/scanfont//.

Peter Selinger. 2003. Potrace: a polygon-based tracing algorithm. In *http://potrace.sourceforge.net*.

Steven S. Skiena. 2008. *The Algorithm Design Manual* (2nd ed.). Springer Publishing Company, Incorporated.

Maxim Stepin. 2003. Hqx. http://web.archive.org/web/20070717064839/www.hiend3d.com/hq4x.html.

Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. 2007. Image Vectorization Using Optimized Gradient Meshes. In *ACM SIGGRAPH*. Article 11.

Daniel Sýkora, Jan Buriánek, and Jiří Žára. 2005. Sketching Cartoons by Example. In *Proc. Sketch-Based Interfaces and Modeling*. 27–34.

Vector Magic. 2017. Cedar Lake Ventures http://vectormagic.com/.

J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R von der Heydt. 2012. A Century of Gestalt Psychology in Visual Perception I. Perceptual Grouping and Figure-Ground Organization. *Psychological Bulletin* 138, 6 (2012), 1172–1217.

C. Wang, J. Zhu, Y. Guo, and W. Wang. 2017. Video Vectorization via Tetrahedral Remeshing. *IEEE TIP* 26, 4 (April 2017), 1833–1844.

Zhaowen Wang, Ding Liu, Jianchao Yang, Wei Han, and Thomas Huang. 2015. Deep networks for image super-resolution with sparse prior. In *IEEE ICCV*. 370–378.

M. Weber and B. Herzog. 2004. Autotrace. http://autotrace.sourceforge.net.

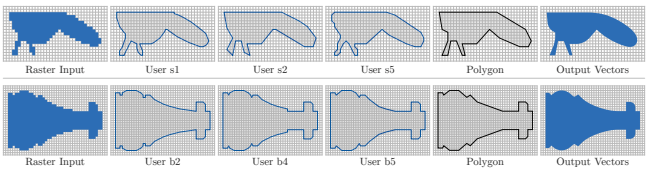


Fig. 25. Examples of participant-drawn perception study results (b-f) and our outputs (g) on the same inputs (a). Top: participants were allowed to use all pixel corners and edge midpoints as polygon corners. Bottom: participants were allowed to use the same raster boundary corners/midpoints as our method.

Tian Xia, Binbin Liao, and Yizhou Yu. 2009. Patch-based Image Vectorization with Automatic Curvilinear Feature Alignment. *ACM TOG* 28, 5 (2009).

Guofu Xie, Xin Sun, Xin Tong, and Derek Nowrouzezahrai. 2014. Hierarchical Diffusion Curves for Accurate Automatic Image Vectorization. *ACM Trans. Graph.* 33, 6 (2014), 230:1–230:11.

M. Yang, H. Chao, C. Zhang, J. Guo, L. Yuan, and J. Sun. 2016. Effective Clipart Image Vectorization through Direct Optimization of Bezigons. *IEEE TVCG* 22, 2 (2016), 1063–1075.

Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. 2009. Vectorizing Cartoon Animations. *IEEE TVCG* 15, 4 (2009), 618–629.

A PERCEPTION STUDIES OF POLYGON APPROXIMATION

Our polygon approximation algorithm is guided by observations from two small-scale perception studies.

Our first study was designed to address the question of where humans place corners when approximating raster inputs using polygons. Participants were shown an editing interface for polygon drawing (see supplementary for UI snapshot and study instructions) and asked to draw polygons best approximating the raster. The interface allowed them to place polygon corners at all pixel corners and pixel edge midpoints. The study included 5 participants and 7 representative, abstract, raster inputs. An example of the polygons participants drew is shown in Figure 25,top. The rest are included in the supplementary. Across all inputs the participants chose raster corners (Section 4) as polygon corners 64% of the time, and chose edge-midpoint immediately adjacent to those 19% of the time. They used midpoints of short symmetric two and one pixel edges 10% of the time. Only 7% of the corners they used were placed in other locations. Analysis of the cases where participants chose mid-points next to corners shows that these were largely chosen in pairs as a proxy for smoothing the corners, a task handled by our subsequent smooth fitting step. These observations motivated us to use only raster corners and short edge mid-points as corners for the graph computation (Section 4), as including more vertices would increase computation cost and bring few benefits in terms of vectorization outcome.

The goal of our second study was to analyze the type of edges humans choose and in particular to observe how they balance accuracy against other cues. The study used a similar interface to the first one but only allowed participants to place corners at the locations used by our algorithm. The study included 5 participants and 14 representative, abstract, raster inputs. Examples of the polygons participants drew are shown in Figure 25, bottom. The rest are included in the supplementary. 96% of the edges chosen by viewers were at Manhattan distance of less than 1 from pixel corners. From all edges, 27% produced pixel center classifications different from those induced by the raster boundaries where all misclassified pixels were on the same side, only 1% of edges had misclassified pixels on both sides of the line. These criteria motivated our choice of edges to include in the graph and our accuracy term.

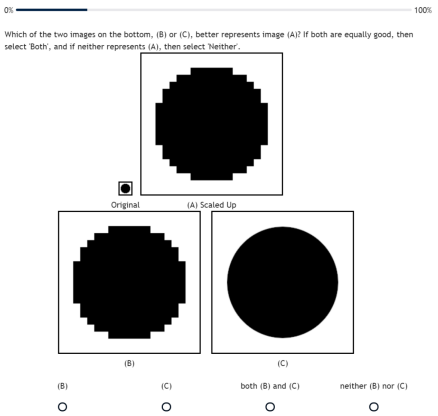


Fig. 26. The filter question for the study. All other questions had a similar layout

As both studies show, while there is some variance between participant results, overall the shapes they draw are very similar. Visual comparison of study results to our outputs further reinforces our algorithmic choices, as our polygons are very similar to the manually created ones (Figure 25, right).

B SPLINE FITTING INITIALIZATION

When fitting single-curves to corners, we initialize the endpoints of the Bézier curve to lie on the midpoints of the corresponding polygon edges. We place the interior control points along the line segments and derive their distance from the endpoints using the curvature variation-minimizing scheme of Jaklič *et al.* [2011]. For curve-line primitive configurations, we compute a circular arc that is tangential to the polygon edges and touches the shorter of the two edges at its midpoint. We then initialize a Bézier curve to approximate this arc and add a line segment along the longer edge. We initialize line-line primitive configurations by placing the line end points at the respective midpoints and corners.

C QUANTITATIVE ASSESSMENT

The study was conducted using the Mechanical Turk interface. Participants were shown two example questions, the one shown in Figure 26, and a second showing a raster square as input and a square and rounded square as answer options, and the correct answers to those (circle and square). Each participant was then shown twenty randomly selected queries with each query shown twice with ”B” and ”C” switched. To filter out inconsistent answer, we also asked participants the circle example question, the answer to which they were explicitly shown (Figure 26), and discarded all answers from participants who answered this question incorrectly. For consistency with [Hoshyari et al. 2018], we discarded inconsistent answers where a user chose different answers to the same duplicated query and all answers from participants who answered inconsistently over 60% of the queries. All questions are included in the supplementary.