Approximating shapes in images with low-complexity polygons

Muxingzi Li Florent Lafarge
Université Côte d'Azur, Inria
firstname.lastname@inria.fr

Renaud Marlet
Valeo.ai & LIGM, Ecole des Ponts,
Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France
renaud.marlet@enpc.fr

Abstract

We present an algorithm for extracting and vectorizing objects in images with polygons. Departing from a polygonal partition that oversegments an image into convex cells, the algorithm refines the geometry of the partition while labeling its cells by a semantic class. The result is a set of polygons, each capturing an object in the image. The quality of a configuration is measured by an energy that accounts for both the fidelity to input data and the complexity of the output polygons. To efficiently explore the configuration space, we perform splitting and merging operations in tandem on the cells of the polygonal partition. The exploration mechanism is controlled by a priority queue that sorts the operations most likely to decrease the energy. We show the potential of our algorithm on different types of scenes, from organic shapes to man-made objects through floor maps, and demonstrate its efficiency compared to existing vectorization methods.

1. Introduction

Extracting objects in images is traditionally operated at the pixel scale, one object being represented as a group of pixels. Such a resolution-dependent representation is often not adapted to the end-users. In many application scenarios as urban mapping or sketching, objects need to be captured with more compact and editable vector representations. In particular, polygons with floating coordinates allow both the approximation of free-form shapes, e.g., organic objects, and the fine description of piecewise-linear structures, e.g., buildings and many other man-made objects.

We consider the task of capturing objects in images by polygons with three objectives. First, *fidelity*: the output polygons should approximate well the object silhouettes in the input image. Second, *complexity*: the output polygons should be composed of a small number of edges to offer a compact and editable representation. Last, *geometric guarantees*: the output polygons should be intersection-free, closed, potentially with holes, and form an image partition.

The simplest way to capture the silhouette of an object as a polygon is to vectorize a chain of pixels representing the object contours [9, 12, 38]. While the complexity of the polygon can be easily controlled, these simplification processes do not take into account structural information contained in the input image. Consequently, output polygons are often imprecise, typically with edges that do not fit accurately the object silhouettes. Recent works on the partitioning of images into polygonal cells [1, 3, 11, 16] suggest that grouping cells from these partitions can produce more accurate results than traditional vectorization methods. This strategy however suffers from imprecise partitions, typically with some polygonal cells overlapping two different objects. Existing works in the field focus on merging polygonal cells only and omit the necessity of splitting operations to deliver more precise results.

In this work, we propose an algorithm to capture objects by compact floating polygons. Inspired by mesh deformation techniques in Geometry Processing, the main idea consists in refining the geometry of an imprecise polygonal partition while labeling each cells by a semantic class.

Our algorithm relies on two key contributions. First, we design an energy function to measure the quality of a polygonal partition by taking into account both the fidelity to input data (image and semantic information) and the complexity of the output polygons. Second, we propose an efficient optimization scheme to minimize that energy. We explore the solution space by splitting and merging cells within the polygonal partition. The mechanism is controlled by a priority queue that sorts the operations that are most likely to decrease the energy.

We demonstrate the potential of our method on different types of scenes, from organic shapes to man-made objects through floor maps and line-drawing sketches, and show its efficiency with respect to existing vectorization approaches.

2. Related work

We distinguish four families of existing, related methods.

Vectorization pipelines. The most popular strategy consists in extracting the object contours by chains of pixels that are then simplified into polygons. Contour extraction can be performed by various methods such as Grabcut [33], superpixel grouping [23] or the popular object saliency detection algorithms [7, 24, 37]. The subsequent simplification step traditionally relies upon the Douglas-Peucker algorithm [38] or mechanisms that simplify Delaunay triangulations [12, 9]. Because these algorithms only measure the geometric deviation from an initial configuration of highly complex polygons, their output can easily drift from the object silhouettes, leading to high accuracy loss in practice.

Methods based on geometric primitives. Another strategy consists in detecting geometric primitives such as line segments in the input image and assemble them into closed contours. The assembling step can be performed by analyzing an adjacency graph between line segments [34], or by gap filling reasoning [39]. These algorithms however do not guarantee the output polygons to be intersection-free. Polygonal Markov random fields [22] are an alternative to sample polygons from images directly. But this model is very slow to simulate in practice and operates on simple synthetic images only. Delaunay point process [14] allows the sampling of vertices within a Delaunay triangulation while grouping the triangulation facets into polygons.

NN architectures. Polygon-RNN [5] and its improved version [2] offer a semi-automatic object annotation with polygons. These models produce polygons with possible self-intersections and overlaps, let alone because the RNN-decoders considers only three preceding vertices when predicting the next vertex at each time step. In contrast, Poly-CNN [19] is automatic and avoids self-intersections. This CNN-based architecture is however restricted to output simple polygons with four vertices. PolyMapper [25] proposes a more advanced solution based on CNNs and RNNs with convolutional long-short term memory modules. In practice, these deep learning techniques give good results for extracting polygons with a low number of edges, typically residential buildings from remote sensing images. However, extracting more complex shapes with potentially hundred of edges per polygon is still a challenging issue.

Methods based on polygonal partitions. A last strategy consists in over-segmenting an image into polygonal cells, and then grouping them to approximate the object silhouettes. The vectorization of superpixels [1] is a straightforward way to create a polygonal partition, that is however composed of non-convex cells whose spatial connection is not clearly defined. Polygonal partitions can be more robustly created by fitting a geometric data structure on the input image. Many methods have been built upon the Line Segment Detector [36] to geometrically characterize object contours with a set of disconnected line segments. The latter are then used for constructing a Voronoi diagram whose edges conform to these line segments [11], a convex mesh with constrained edges [16], or a planar graph using a kinetic framework [3]. The cells of such polygonal partitions are then grouped to form polygons, either by graph-cut [3] or other aggregation mechanisms [26, 32]. This strategy delivers accurate results when the polygonal partition fits well the input image, which is rarely the case in practice. Unfortunately, the refinement of polygonal partitions has not been deeply explored in the literature. The only solution proposed to our knowledge consists in a splitting phase which incrementally refines a Delaunay triangulation before merging the triangles [18]. Unfortunately, handling triangular cells does not allow to produce compact polygons.

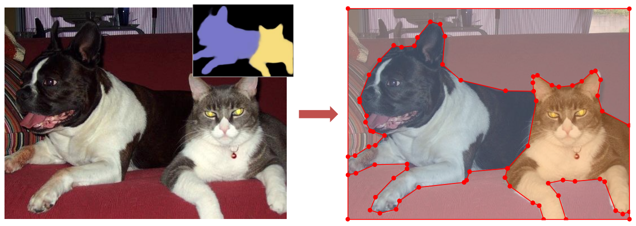


Figure 1. Goal of our approach. Our algorithm takes as input an image with a rough semantic probability map and outputs a set of low-complexity polygons capturing accurately the objects of interest, here dogs and cats.

3. Overview

The algorithm takes as input an image and an associated probability map that estimates the probability of each pixel to belong to the different classes of interest. This probability map is typically generated by state-of-the-art semantic segmentation methods or saliency detection algorithms.

The algorithm departs from a polygonal partition generated by kinetic propagation of line segments [3]. Each cell of this partition is enriched by a semantic label chosen as the class of interest with the highest mean over the inside pixels in the probability map. The goal of our algorithm is then to refine this *semantic polygonal partition* by splitting and merging cells in tandem. These refinement operations are guided by an energy that accounts for both fidelity to input data and complexity of output.

The algorithm ends when no splitting or merging operations can decrease the energy anymore. Each cell in the output is a polygon associated with a class of interest, as illustrated in Fig. 1. By construction, the set of output polygons is guaranteed to recover the entire image domain without overlaps, to be closed and intersection-free, and does not contain edge-adjacent cells with the same semantic label.

4. Algorithm

We denote a semantic polygonal partition by $\mathbf{x} = (\mathbf{m}, \mathbf{l})$ where $\mathbf{m}$ defines a 2D polygon mesh on the image domain while $\mathbf{l}$ represents the semantic labels associated to the facets of $\mathbf{m}$. We denote by $\mathcal{F}_{\mathbf{x}}$ (respectively $\mathcal{E}_{\mathbf{x}}$) the set of facets (resp. non-border edges) of the polygon mesh $\mathbf{m}$.

4.1. Energy formulation

We measure the quality of a semantic polygonal partition $\mathbf{x}$ with an energy function U of the form:

$$U(\mathbf{x}) = (1-\lambda)U_{\mathit{fidelity}}(\mathbf{x}) + \lambda U_{\mathit{complexity}}(\mathbf{x}) \tag{1}$$

The first term U_{fidelity} measures the coherence of the configuration $\mathbf{x}$ with the input data while $U_{\mathit{complexity}}$ encourages low-complexity outputs. These two terms, that are balanced by a model parameter $\lambda \in [0,1]$, are typically expressed with local energies on the edges and facets of the mesh $\mathbf{m}$.

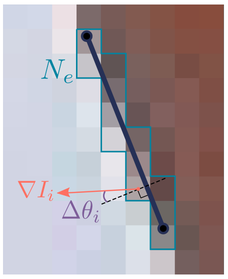
Fidelity term U_{fidelity} has two objectives: (i) encouraging the semantic label of each facet to be coherent with the probability map, and (ii) encouraging edges to align with high gradients of the input image. These objectives are balanced by parameter β, set to 10^{-3} in our experiments:

$$U_{\mathit{fidelity}}(\mathbf{x}) = \sum_{f \in \mathcal{F}_{\mathbf{x}}} -w_f \log P_{map}(l_f) + \beta \sum_{e \in \mathcal{E}_{\mathbf{x}}} w_e A(e) \tag{2}$$

where w_f is the ratio of the area of facet f to the area of the whole image domain, $P_{map}(l_f)$ is the mean of the probability map for class l_f over the pixels inside facet f, and w_e is the inverse of the length of the image diagonal if the two adjacent facets f and f' of edge e have different labels $l_f \neq l_{f'}$, and 0 otherwise. Finally, $A(e)$ is a function measuring the alignment of edge e with image gradients:

$$A(e) = \sum_{i \in N_e} r_i \big[1 - \hat{F}(m_i) \exp\big(-\frac{\Delta\theta_i^{\,2}}{2\sigma^2}\big)\big] \tag{3}$$

where N_e is the set of pixels that overlap with edge e, r_i is the inverse of the number of edges that overlap pixel i, $\Delta\theta_i$ is the angular difference between the gradient direction at pixel i and the normal vector of edge e, and σ is a model parameter set to $\frac{\pi}{8}$ in our experiments. Denoting $\hat{F}$ the empirical cumulative density distribution of gradient magnitudes over the input image, $\hat{F}(m_i)$ is the probability that the gradient magnitude of a random pixel in the input image is smaller than the gradient magnitude m_i at pixel i. Note that, instead of image gradients, more general discontinuity maps such as [21, 10] could be used by modifying the density distribution $\hat{F}$ in Eq. (3).



Complexity term $U_{\mathit{complexity}}$ penalizes a complex polygon mesh with the number of edges (the lower, the better):

$$U_{\mathit{complexity}}(\mathbf{x}) = |\mathcal{E}_{\mathbf{x}}| \tag{4}$$

As illustrated in Figure 2, the model parameter λ is a trade-off between fidelity to input data and complexity of the output polygons. Note that our data term measures data fidelity independently of polygon complexity. In particular, $A(e)$ is designed as a linear function so that, if an edge e is composed of two collinear edges e_1 and e_2, then $A(e) = A(e_1) + A(e_2)$. The linearity of $A(e)$ requires that each gradient pixel should not contribute multiple times to the total energy, which explains the factor r_i in Eq. (3).

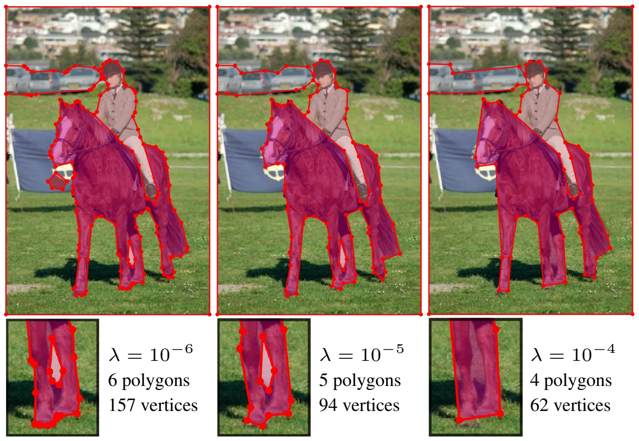


Figure 2. Trade-off between fidelity to data and complexity to output polygons. Increasing λ gives more compact, yet less accurate, output polygons. Objects of interest: horses, persons and cars.

4.2. Exploration mechanism

Both continuous variables for representing the polygon mesh and discrete semantic labels are involved in the minimization of the (non-convex) energy U. Inspired by edge contraction algorithms for simplifying triangle meshes [4, 17], we explore efficiently such a large solution space via an iterative mechanism based on local operators that split and merge facets of the polygon mesh $\mathbf{m}$. Starting from an initial configuration, we compute the energy variations for splitting each facet as well as the energy variations for merging each pair of adjacent facets. All the energy variations (values to add to the energy if performing the corresponding operation) are sorted into a priority queue in ascending order, i.e., with more negative energy variations first. The exploration mechanism then consists in operating the splitting or merging at the top of the priority queue, i.e., the move that gives the highest energy decrease. This modification is followed by an update of the priority queue. A pseudo-code of the exploration mechanism is given in Algorithm 1. We now detail the main components of this algorithm.

Algorithm 1 Pseudo-code of exploration mechanism

```
1: Initialize the semantic polygonal partition $\mathbf{x}$
2: Initialize the priority queue $Q$
3: while The top operation $i$ of $Q$ decreases energy U do
4:     Update $\mathbf{x}$ with the merging or splitting operation $i$
5:     Update $Q$
6: end while
```

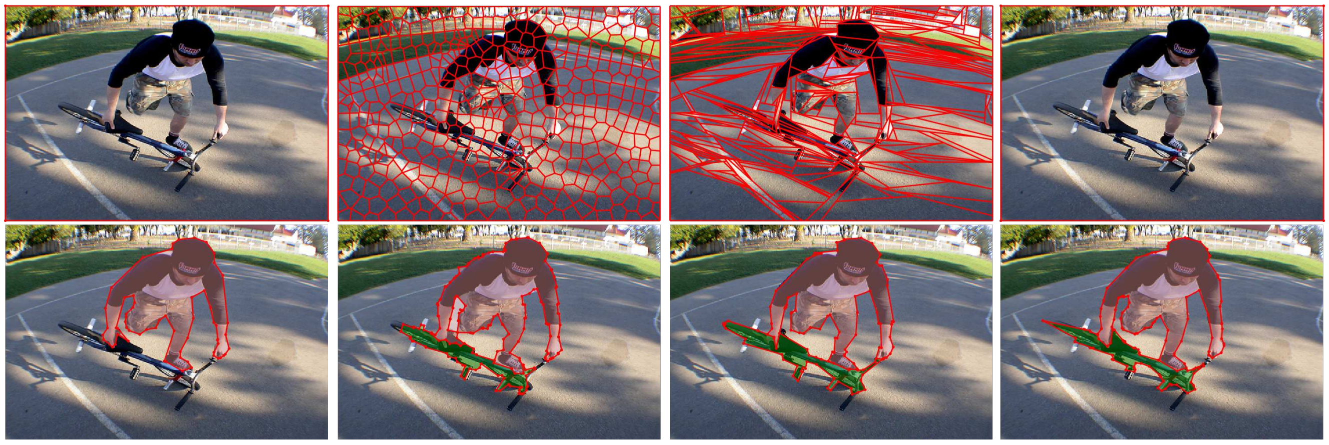
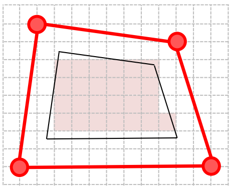


Figure 3. Initialization. The top (resp. bottom) row shows the initial partitions (resp. output polygons). Objects of interest are persons and bikes. Starting the exploration mechanism from a partition composed of one rectangular facet (column 1) typically produces results with missing objects such as the bike. An initial Voronoi partition [11] (column 2) is too fragmented to output low complexity polygons. Our algorithm performs best from kinetic partitions [3] (column 3) with a good trade-off between accuracy and polygon complexity. This option returns similar results than a simulated annealing exploration (column 4) but with processing times reduced by two orders of magnitude. For clarity reasons, here and in the following figures, we do not display the background polygons (at the image border) in the visual results.

Initialization. Because the exploration mechanism finds a local minimum, a good initial configuration is required. In our experiments, we build the initial semantic polygonal partition using the kinetic partitioning method proposed in [3]. It produces in a fast and scalable manner a partition of polygonal cells that captures well the homogeneous regions in images. This partition is turned into a 2D polygon mesh. We then assign to each facet the semantic label that returns the highest mean over the inside pixels in the probability map. The impact of initialization is illustrated in Figure 3.

Merging operator. The merging operator merges two facets with at least one edge in common into a single facet. The update consists in removing all common edges in between the two original facets as illustrated in Figure 4. The semantic label of the new, merged facet is chosen as the most probable label with respect to the probability map.

Splitting operator. This operator divides a facet into multiple facets by inserting new edges and vertices. We first detect a set of cutting directions inside the original facet. These directions are found by fitting line segments to the input image with a region growing algorithm [31]. To avoid detecting line segments overlapping the edges of the facet, only pixels inside the facet shrunk by 2 pixels are considered for the fitting (see the set of pink pixels inside the red facet in the inset). The detected line segments are then extended until they collide with the outside edges of the original facet or themselves, as illustrated in Figure 4. The collision points (respectively the prolonged line segments) correspond to new vertices (resp. edges) inserted in the 2D polygon mesh. For each new facet,



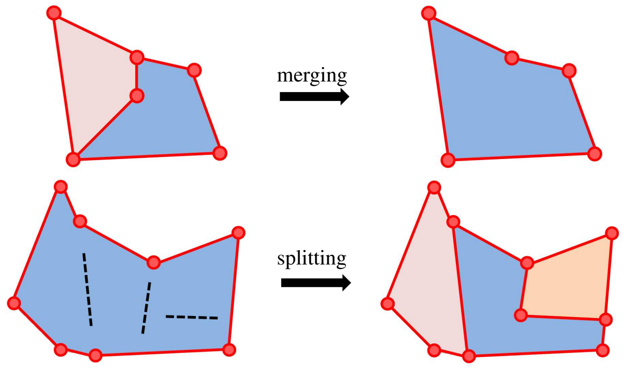


Figure 4. Merging and splitting operators. The merging operator merges two adjacent facets with different semantic labels by removing the common edges (top). The splitting operator divides one facet into multiple facets that have different semantic labels (bottom). The black dashed lines indicates the cutting directions detected in the input image (bottom left).

we associate the most probable semantic label with respect to the probability map. If two new adjacent (sub)facets have the same semantic label, they are immediately merged, as part of the splitting operation.

Priority queue. After a configuration $\mathbf{x}$ is modified, the priority queue must be updated. We first remove from the priority queue the current operation and all the merging or splitting operations concerning the modified facets. We then compute the energy variations of all possible operations that can affect the new facets and insert them in the priority queue, appropriately sorted. Because the energy is formulated as the sum of local terms and a global complexity term, these variations are not costly to compute. When a split occurs, only the parent facet, its new split facets and

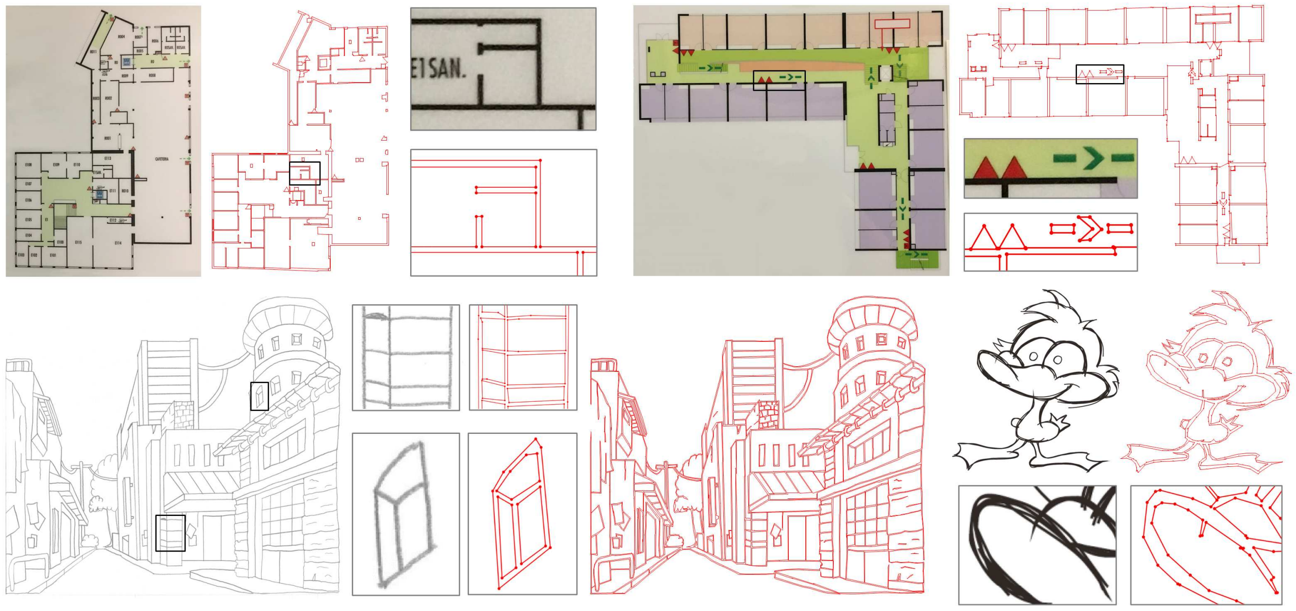


Figure 5. Vectorization of linear structures. Our algorithm can be used to vectorize floor map photographs (top) or line-drawings (bottom). While thin, these linear structures can be captured by compact polygons with a good accuracy (see closeups).

the edges composing these facets are involved in the energy updates of the priority queue. These updates are fast and local; they do not propagate through the whole mesh. In our experiments, the average number of facets created per split is 2.1 and the average number of updated edges is 7.2.

Stopping criterion. The exploration mechanism ends when the energy variations sorted in the priority queue become all positive, i.e., when no operation can decrease the energy anymore. Note that this criterion guarantees the exploration mechanism to converge quickly without bumping effects. Besides, the final solution cannot contain two edge-adjacent polygons with the same semantic class, as merging them necessarily decreases the energy (lower $U_{fidelity}$, thanks to the convexity of $-\log$, and lower $U_{complexity}$).

Details for speeding-up the exploration. The exploration mechanism is local. This choice is motivated by low running time and the presence of good initial configurations. (An alternative could be to use a non-local optimization algorithm such as the simulated annealing, cf. Figure 8.)

Observing that a complex initial partition often over-segments the probability map, we initially (before exploration) merge all adjacent facets that contain only pixels classified with the same label. This highly reduces the processing time without affecting the results.

To reduce the time for detecting line segments when new splitting operations are considered, we allow a merged facet to inherit the already-detected line segments of its parent facets. We detect new line segments only in the area around the removed edges. In addition to time savings, this allows us to refine the edges between two adjacent facets by operating a merging and then a splitting on the same facet.

5. Experiments

Our algorithm has been implemented in C++ using the Computational Geometry Algorithms Library (CGAL) [35]. All experiments have been done on a single computer with Intel Core i7 processor clocked at 2.4GHz.

Parameters. We have 3 model parameters λ, β, σ, that are set respectively to 10^{-5}, 10^{-3}, $\frac{\pi}{8}$ in all experiments, despite the dataset variety. (Note that our algorithm does not need any threshold to stop the exploration.) The values of λ and β were chosen based on a grid search; σ was set to roughly model the standard deviation of gradient directions.

Flexibility and robustness. Our algorithm has been tested on different types of scenes and objects. Piecewise-linear structures such as buildings are captured with fine details as long as probability maps have a good accuracy. Organic shapes such as humans and animals are approximated by low complexity polygons. In addition to the silhouettes of objects in images, our algorithm can also be used to vectorize floor map photographs or line-drawing sketches. These two applications usually require the use of specialized methods to detect, filter and connect corner points into a floor map [27] or strokes into a network of parametric curves [15]. In contrast, our algorithm finely reconstructs these linear structures, as illustrated in Figure 5. Our algorithm offers a good robustness to imprecise probability maps thanks to the second part of the data term that favors the alignment of edges with image discontinuities. As illustrated in Figure 6, the output polygons can accurately capture the silhouette of objects even if the probability map is ambiguous where different objects meet.

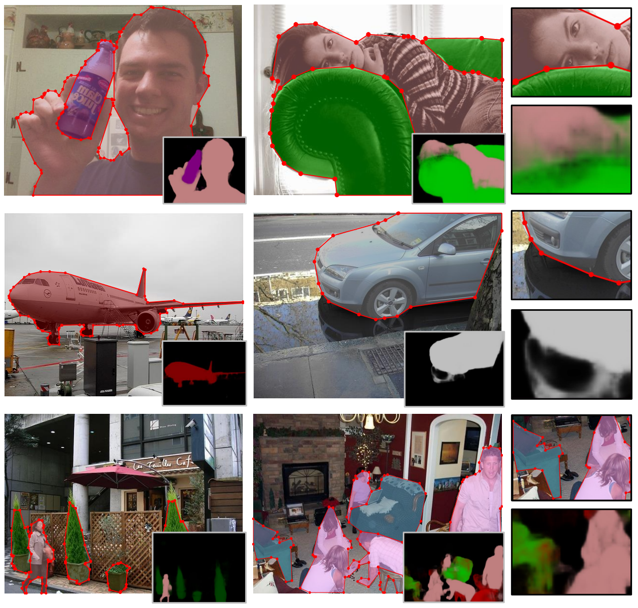
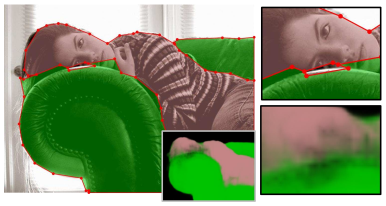


Figure 6. Vectorization of multi-class objects. Probability maps are often ambiguous and only roughly indicate the shape of the objects (see colors for different classes). Our algorithm captures the silhouette of theses objects with low-complexity polygons with a good precision. Note in particular how the polygons nicely delineate close objects, such as the lady face and the couch (see close-ups). A failure case is shown in the bottom right example where the quality of the probability map is too poor to capture the underlying object. Images are from the PASCAL VOC2012 dataset.

Ablation study. As semantic maps are often blurry at object boundaries, using only the first data term yields polygons that do not contour well the objects, as illustrated in the inset. This result, obtained with β=0, must be compared with the results obtained in Figure 6 in the same initial conditions but with $\beta\neq$0.



Quantitative evaluation. We compared our algorithm to state-of-the-art methods on three different datasets.

We first tested our algorithm on the HKU-IS dataset [24] designed to evaluate salient object detection methods. We computed the probability map for each image using the algorithm of Li and Yu [24]. We compared our algorithm to two vectorization pipelines in which the same saliency maps [24] are binarized before chaining and simplifying the pixels on the object contours, either by the popular Douglas-Peucker algorithm [38] or by polyline decimation [12]. We also compared to two cell grouping algorithms that generate a polygonal partition by Voronoi diagram construction [11] or by kinetic propagation of line segments [3]. The polygons are then extracted from these partitions by thresholding the saliency map averaged over each cell. We denote these methods respectively by Voronoi and Kippi. The accuracy is measured using Intersection-over-Union of our pixelized output polygons against the ground truth. We also measure compression as the ratio of the number of pixels of the ground truth region boundary to the number of polygon vertices. In practice, we produce polygons at different complexity by varying λ, as shown in Figure 2.

	Compression ratio					
Method	10	15	20	25	33	50
Voronoi [11]	77.7	75.2	71.6	68.2	64.1	57.5
Kippi [3]	79.2	77.1	72.8	69.5	65.6	62.1
Douglas-Peucker [38]	83.8	83.3	81.2	79.4	76.0	65.7
Polyline [12]	83.9	83.7	82.5	81.2	77.5	69.0
Ours	**84.1**	**84.0**	**83.7**	**83.1**	**81.3**	**77.0**

Table 1. Accuracy (%) vs compression on HKU-IS.

	Compression ratio					
Method	10	15	20	25	33	50
Voronoi [11]	87.9	86.4	83.6	81.3	77.7	74.3
Kippi [3]	88.9	87.6	85.4	83.0	79.5	75.2
Douglas-Peucker [38]	91.2	90.9	90.1	88.8	86.6	79.8
Polyline [12]	91.2	91.1	90.6	89.9	88.1	85.8
Ours	**91.7**	**91.6**	**91.5**	**91.4**	**91.2**	**89.7**

Table 2. Accuracy (%) vs compression on PASCAL VOC2012.

Table 1 shows the evolution of accuracy vs compression on the HKU-IS dataset. While all methods exploit the same saliency maps, only our algorithm maintains high accuracy at high compression ratios, i.e., when the output polygons have a very low number of vertices. Fig. 7 shows visual comparisons of the methods at low and high compression. At low compression, the vectorization pipelines Douglas-Peucker and Polyline produce accurate polygons, similarly to our algorithm. Because these pipelines simplify the geometry of polygons without data consistency, their accuracy significantly drops for higher compression ratios, typically from 25. Cell grouping methods Voronoi and Kippi suffer from imperfect polygonal partitions where cells often overlap several types of objects. In contrast, the merging and splitting operations of our algorithm allow us to refine cells with respect to the probability map and the input image.

We also tested our algorithm on the Pascal VOC2012 dataset [13] designed for multi-class segmentation tasks. This dataset contains 20 object classes and 1 background class. The evaluation was done on the validation set. We compared our algorithm to the same four methods (Douglas-Peucker, Polyline, Voronoi and Kippi) with the same accuracy and compression metrics. Probability maps were generated by the DeepLab algorithm [6] by taking the output layer before the final argmax operation over class channels. Table 2 shows the evolution of accuracy against compression for the five algorithms. Similarly to the quantitative results obtained on the HKU-IS dataset, our algorithm outclasses the other methods, in particular with a significant accuracy gain at high compression. Figure 6 shows visual results obtained by our algorithm on different object classes.

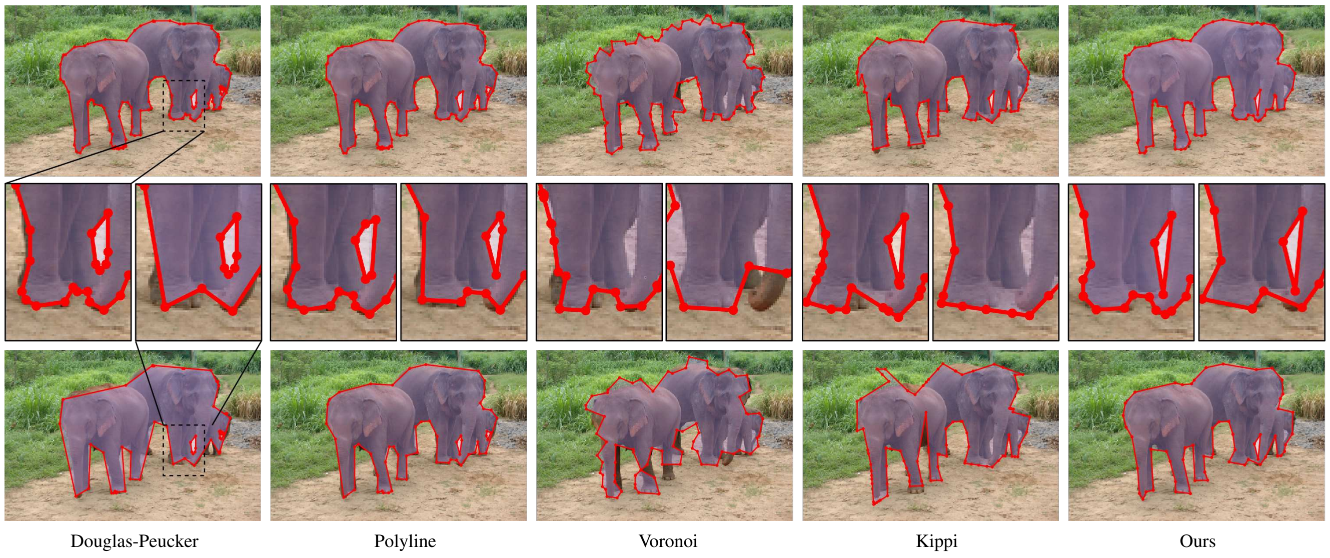


Figure 7. Visual comparisons at two compression ratios: 10 (top) and 33 (bottom). While the vectorization pipelines Polyline and, to a lesser extent, Douglas-Peucker yield accurate polygons at low compression, their precision drops at high compression, with polygons not aligning well with silhouettes anymore (cf. closeups). The cell-grouping algorithms Voronoi and Kippi are less accurate on such free-form shapes where cells often overlap several object classes. In contrast, we accurately capture the elephants at both compression ratios.

Method	AP	AP_{50}	AP_{75}	AR	AR_{50}	AR_{75}
R-CNN [20]	41.9	67.5	48.8	47.6	70.8	55.5
PANet [28]	50.7	73.9	62.6	54.4	74.5	65.2
PolyMapper[25]	55.7	86.0	65.1	62.1	88.6	71.4
Ours	**65.8**	**87.6**	**73.4**	**78.7**	**94.3**	**86.1**

Table 3. Performance on the CrowdAI mapping challenge dataset. Average precision (AP) and average recall (AR) in %.

We finally tested our algorithm on the CrowdAI mapping challenge dataset [29] which is composed of ∼60k satellite images of urban landscapes. Probability maps were generated using a U-Net variant [8]. We followed the same experimental protocol than in [25] for extracting the contours of buildings from this dataset. In particular, we used the same average precision (AP) and average recall (AR) metrics. We compared our algorithm with the deep learning methods PolyMapper [25], Mask R-CNN [20] based on the implementation of [30], and PANet [28]. Table 3 presents the quantitative results on these four methods. Our algorithm obtains the best average precision and average recall scores. In particular, our algorithm outclasses Polymapper with significant gains. This difference is partly explained by the iterative mechanism of vertex insertion of Polymapper whose efficiency decreases for complex shapes. By refining polygonal cells on a topologically-valid partition, our algorithm does not suffer from this problem. Figure 9 shows visual results on an urban scene of the CrowdAI dataset.

Performance. Figure 8 shows that our exploration mechanism reaches similar energies as a non-local simulated annealing while being two orders of magnitude faster.

Our exploration mechanism is inspired by edge contraction algorithms for mesh simplification. While local, greedy and old, such algorithms, e.g., [4, 17], are still very popular and commonly used in the field.

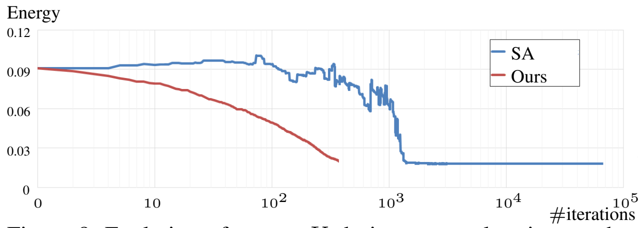
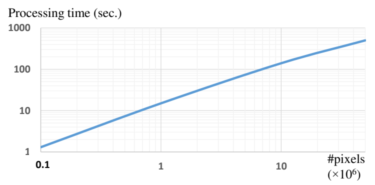


Figure 8. Evolution of energy U during our exploration mechanism (red curve) and a simulated annealing optimization (SA, blue curve). While the two optimization techniques converge towards a similar energy, our exploration mechanism requires two orders of magnitude less iterations than the simulated annealing.

As shown in the inset, our algorithm typically requires a few seconds for a 100K-pixel image and about 2 min for a 10M-pixel image.



Note that our code has not been optimized (beyond the general strategy expressed at the end of Sect. 4.2). In particular, the exploration mechanism runs sequentially on CPU (no parallelization). The most time-consuming operation is the update of the priority queue, and especially the simulation of splitting operations for the new large facets. If the initial partition contains N_f facets and N_e non-border edges, the priority queue is constructed by sorting the energy variations of the N_f possible splits and N_e possible merges; the running time for this is negligible ($< 0.1\%$ of total time). Last, the computation of cutting directions depends on the

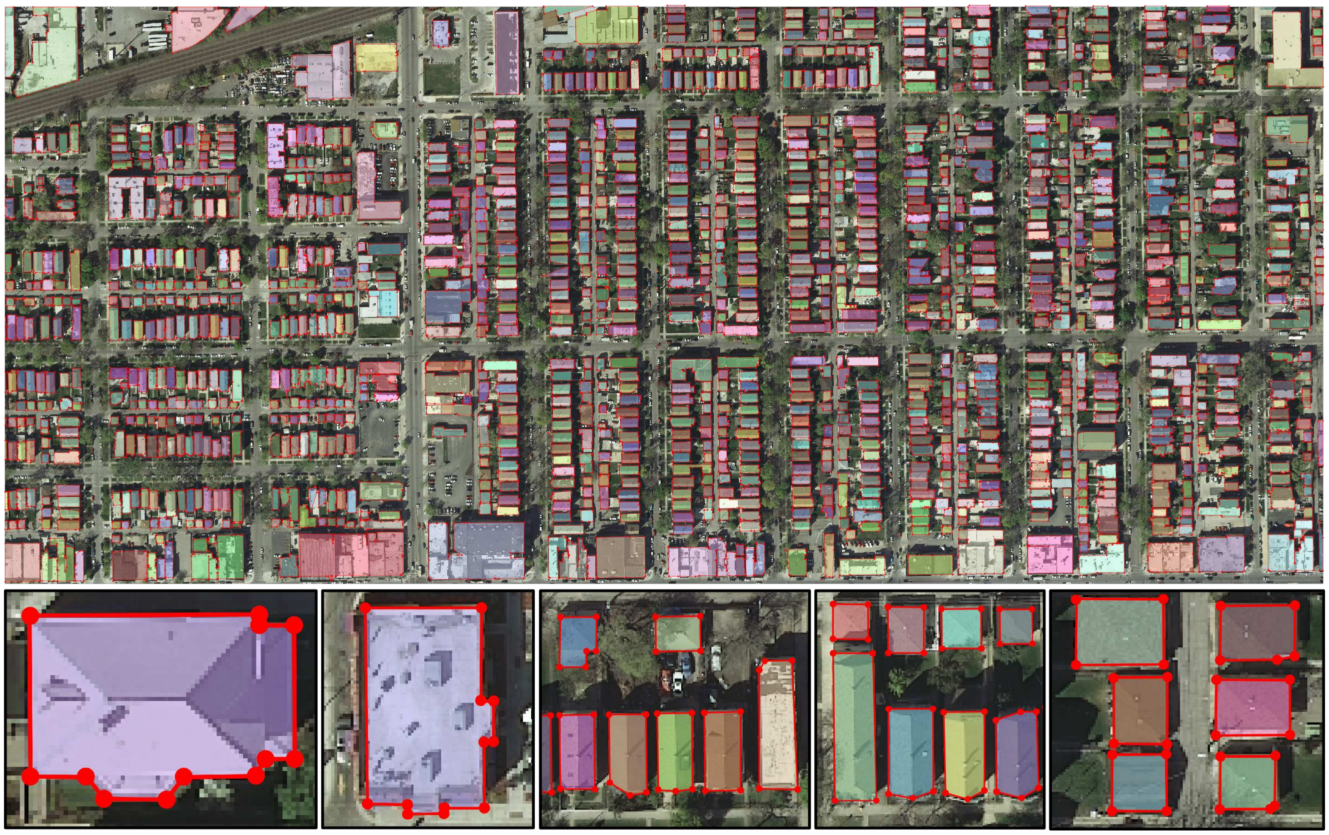


Figure 9. Extraction of buildings from satellite images with our algorithm: 1,178 buildings of a half square kilometer area of Chicago, USA, are extracted with low complexity polygons (8,683 vertices). While compact, the polygons capture some fine details (see closeups).

number of image pixels. It is very fast and performed only once at priority queue initialization. Getting split directions from the input image lowers the dependency on the initial partition and allows larger solution space explorations.

Limitations. As energy $U(\mathbf{x})$ is not convex and as our exploration mechanism is local, results depend on the quality of the initial partition. As shown in Fig. 3, splits provide robustness to a range of under-segmentations; yet, an initial partition that over-segments well the image leads to more accurate results. If a good initial partition cannot be provided or guaranteed, simulated annealing can be a better choice regarding accuracy, but not running time (cf. Fig. 8).

Thanks to the gradient alignment term in $U_{fidelity}$, our algorithm is robust to some level of error or ambiguity in semantic maps, in particular at object border; see, e.g., the polygons capturing the lady's face and the couch from the blurry semantic map in Fig. 6. Yet, the class probability of most pixels has to be correct, as is also the case for shape grammar parsers. Note that depending on external methods (initial partition, semantic map) is a strength: our performance will improve along with the related state of the art.

Also, while parameter λ balances data fidelity and output complexity, it does not allow to control the exact number of output vertices, contrary to vectorization pipelines.

6. Conclusion

We proposed an algorithm for extracting and vectorizing objects in images with low-complexity polygons. Our algorithm refines the geometry of an initial polygonal partition while labeling its cells by a semantic class. Based on local merging and splitting of cells, the underlying mechanism is simple, efficient and guaranteed to deliver intersection-free polygons. We demonstrated the robustness and the flexibility of our algorithm on a variety of scenes from organic shapes to man-made objects through floor maps and line-drawing sketches. We also showed on different datasets that it outperforms the state-of-the-art vectorization methods.

In future work, we plan to investigate the user control of the number of output vertices. One way could be to design a third operator that removes and adds relevant vertices in the partition. We would also like to generalize our algorithm to the extraction of Bezier cycles, i.e., polygons where two successive vertices are not connected by a straight line, but by a Bezier curve; it would allow us to capture free-form shapes with a better complexity-distortion trade-off.

Acknowledgments. We thank Jean-Philippe Bauchet for technical discussions. This work was partially supported by ANR-17-CE23-0003 project BIOM.

References

[1] R. Achanta and S. Susstrunk. Superpixels and polygons using simple non-iterative clustering. In *CVPR*, 2017. 1, 2

[2] D. Acuna, H. Ling, A. Kar, and S. Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *CVPR*, 2018. 2

[3] J.-P. Bauchet and F. Lafarge. Kippi: Kinetic polygonal partitioning of images. In *CVPR*, 2018. 1, 2, 4, 6

[4] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon Mesh Processing*. AK Peters / CRC Press, 2010. 3, 7

[5] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017. 2

[6] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 6

[7] M. Cheng, N. Mitra, X. Huang, P. Torr, and S.-M. Hu. Global contrast based salient region detection. *PAMI*, 37(3), 2015. 2

[8] J. Czakon, K. Kaczmarek, Andrzej P., and P. Tarasiewicz. Best practices for elegant experimentation in data science projects. In *EuroPython*, 2018. 7

[9] F. De Goes, D. Cohen-Steiner, P. Alliez, and M. Desbrun. An optimal transport approach to robust reconstruction and simplification of 2d shapes. *Computer Graphics Forum*, 30(5), 2011. 1, 2

[10] P. Dollar and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 3

[11] L. Duan and F. Lafarge. Image partitioning into convex polygons. In *CVPR*, 2015. 1, 2, 4, 6

[12] Christopher Dyken, Morten Dæhlen, and Thomas Sevaldrud. Simultaneous curve simplification. *Journal of Geographical Systems*, 11(3), 2009. 1, 2, 6

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. `http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html`. 6

[14] J.D. Favreau, F. Lafarge, A. Bousseau, and A. Auvolat. Extracting geometric structures in images with delaunay point processes. *PAMI*, 42(4), 2020. 2

[15] J.-D. Favreau, F. Lafarge, and A. Bousseau. Fidelity vs. Simplicity: a Global Approach to Line Drawing Vectorization. *ACM Trans. on Graphics*, 35(4), 2016. 5

[16] J. Forsythe, V. Kurlin, and A. Fitzgibbon. Resolution independent superpixels based on convex constrained meshes without small angles. In *International Symposium on Visual Computing*, 2016. 1, 2

[17] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH*, 1997. 3, 7

[18] T. Gevers and A. W. M. Smeulders. Combining region splitting and edge detection through guided delaunay image subdivision. In *CVPR*, 1997. 2

[19] N. Girard and Y. Tarabalka. End-to-End Learning of Polygons for Remote Sensing Image Classification. In *IGARSS*, 2018. 2

[20] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *ICCV*, 2017. 7

[21] P. Isola, D. Zoran, D. Krishnan, and E.H. Adelson. Crisp boundary detection using pointwise mutual information. In *ECCV*, 2014. 3

[22] R. Kluszczyński, M. N. M. van Lieshout, and T. Schreiber. Image segmentation by polygonal markov fields. *Annals of the Institute of Statistical Mathematics*, 59(3), 2007. 2

[23] A. Levinshtein, C. Sminchisescu, and S. Dickinson. Optimal contour closure by superpixel grouping. In *ECCV*, 2010. 2

[24] G. Li and Y. Yu. Deep contrast learning for salient object detection. In *CVPR*, 2016. 2, 6

[25] Z. Li, J. Dirk Wegner, and A. Lucchi. Topological map extraction from overhead images. In *ICCV*, 2019. 2, 7

[26] Z. Li, Wu Z.-M., and S.-F. Chang. Segmentation using superpixels: A bipartite graph partitioning approach. In *CVPR*, 2012. 2

[27] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *ICCV*, 2017. 5

[28] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 7

[29] S. Mohanty. Crowdai dataset: the mapping challenge. `https://www.aicrowd.com/challenges/`. 7

[30] S.P. Mohanty. Crowdai mapping challenge : Baseline with mask r-cnn. `https://github.com/crowdAI/crowdai-mapping-challenge-mask-rcnn/`. 7

[31] S. Oesau, Y. Verdie, C. Jamin, P. Alliez, F. Lafarge, and S. Giraudot. Point set shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2018. 4

[32] Z. Ren and G. Shakhnarovich. Image segmentation by cascaded region agglomeration. In *CVPR*, 2013. 2

[33] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. *ACM Trans. on Graphics*, 23(3), 2004. 2

[34] X. Sun, M. Christoudias, and P. Fua. Free-shape polygonal object localization. In *ECCV*, 2014. 2

[35] The CGAL Project. CGAL, computational geometry algorithms library. 5

[36] R. Von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *PAMI*, 32(4), 2010. 2

[37] L. Wang, L. Wang, H. Lu, P. Zhang, and X. Ruan. Saliency detection with recurrent fully convolutional networks. In *ECCV*, 2016. 2

[38] S. T. Wu and M. R. G. Marquez. A non-self-intersection Douglas-Peucker algorithm. In *IEEE Symposium on Computer Graphics and Image Processing*, 2003. 1, 2, 6

[39] Z. Zhang, S. Fidler, J. Waggoner, Y. Cao, S. Dickinson, J. Siskind, and S. Wang. Superedge grouping for object localization by combining appearance and shape informations. In *CVPR*, 2012. 2