

Deep Vectorization of Technical Drawings

Vage Egiazarian^{1*}, Oleg Voynov^{1*}, Alexey Artemov¹, Denis Volkhonskiy¹, Aleksandr Safin¹, Maria Taktasheva¹, Denis Zorin^{2,1}, and Evgeny Burnaev¹

¹ Skolkovo Institute of Science and Technology, 3 Nobel Street, Skolkovo 143026, Russian Federation

² New York University, 70 Washington Square South, New York NY 10012, USA
 {vage.egiazarian, oleg.voynov, a.artemov, denis.volkhonskiy, aleksandr.safin, maria.taktasheva}@skoltech.ru, dzorin@cs.nyu.edu, e.burnaev@skoltech.ru

adase.group/3ddl/projects/vectorization

Abstract. We present a new method for vectorization of technical line drawings, such as floor plans, architectural drawings, and 2D CAD images. Our method includes (1) a deep learning-based cleaning stage to eliminate the background and imperfections in the image and fill in missing parts, (2) a transformer-based network to estimate vector primitives, and (3) optimization procedure to obtain the final primitive configurations. We train the networks on synthetic data, renderings of vector line drawings, and manually vectorized scans of line drawings. Our method quantitatively and qualitatively outperforms a number of existing techniques on a collection of representative technical drawings.

Keywords: transformer network, vectorization, floor plans, technical drawings

1 Introduction

Vector representations are often used for technical images, such as architectural and construction plans and engineering drawings. Compared to raster images, vector representations have a number of advantages. They are scale-independent, much more compact, and, most importantly, support easy primitive-level editing. These representations also provide a basis for higher-level semantic structure in drawings (*e.g.*, with sets of primitives hierarchically grouped into semantic objects).

However, in many cases, technical drawings are available only in raster form. Examples include older drawings done by hand, or for which only the hard copy is available, and the sources were lost, or images in online collections. When the vector representation of a drawing document is unavailable, it is reconstructed,

* Equal contribution

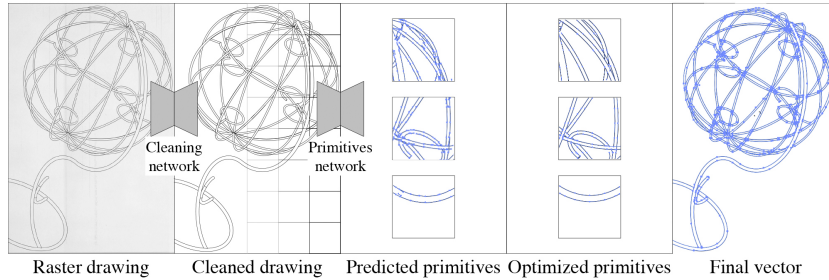


Fig. 1: An overview of our vectorization method. First, the input image is cleaned with a deep CNN. Then, the clean result is split into patches, and primitive placement in each patch is estimated with a deep neural network. After that, the primitives in each patch are refined via iterative optimization. Finally, the patches are merged together into a single vector image.

typically by hand, from scans or photos. Conversion of a raster image to a vector representation is usually referred to as *vectorization*.

While different applications have distinct requirements for vectorized drawings, common goals for vectorization are:

- approximate the semantically or perceptually important parts of the input image well;
- remove, to the extent possible, the artifacts or extraneous data in the images, such as missing parts of line segments and noise;
- minimize the number of used primitives, producing a compact and easily editable representation.

We note that the first and last requirements are often conflicting. *E.g.*, in the extreme case, for a clean line drawing, 100% fidelity can be achieved by “vectorizing” every pixel with a separate line.

In this paper, we aim for geometrically precise and compact reconstruction of vector representations of technical drawings in a fully automatic way. Distinctive features of the types of drawings we target include the prevalence of simple shapes (line segments, circular arcs, etc.) and relative lack of irregularities (such as interruptions and multiple strokes approximating a single line) other than imaging flaws. We develop a system which takes as input a technical drawing and vectorizes it into a collection of line and curve segments (Figure 1). Its elements address vectorization goals listed above. The central element is a deep-learning accelerated optimization method that matches geometric primitives to the raster image. This component addresses the key goal of finding a compact representation of a part of the raster image (a *patch*) with few vector primitives. It is preceded by a learning-based image preprocessing stage, that removes background and noise and performs infill of missing parts of the image, and is followed by a simple heuristic postprocessing stage, that further reduces the number of primitives by merging the primitives in adjacent patches.

Our paper includes the following contributions:

1. We develop a novel vectorization method. It is based on a learnable deep vectorization model and a new primitive optimization approach. We use the model to obtain an initial vector approximation of the image, and the optimization produces the final result.
2. Based on the proposed vectorization method, we demonstrate a complete vectorization system, including a preprocessing learning-based cleaning step and a postprocessing step aiming to minimize the number of primitives.
3. We conduct an ablation study of our approach and compare it to several state-of-the-art methods.

2 Related work

Vectorization. There is a large number of methods for image and line drawing vectorization. However, these methods solve somewhat different, often imprecisely defined versions of the problem and target different types of inputs and outputs. Some methods assume clean inputs and aim to faithfully reproduce all geometry in the input, while others aim, *e.g.*, to eliminate multiple close lines in sketches. Our method is focused on producing an accurate representation of input images with mathematical primitives.

One of the widely used methods for image vectorization is Potrace [36]. It requires a clean, black-and-white input and extracts boundary curves of dark regions, solving a problem different from ours (*e.g.*, a single line or curve segment is always represented by polygon typically with many sides). Recent works [29,22] use Potrace as a stage in their algorithms.

Another widely used approach is based on curve network extraction and topology cleanup [11,3,30,6,5,31,18]. The method of [11] creates the curve network with a region-based skeleton initialization followed by morphological thinning. It allows to manually tune the simplicity of the result trading off its fidelity. The method of [3] uses a polyvector field (crossfield) to guide the orientation of primitives. It applies a sophisticated set of tuneable heuristics which are difficult to tune to produce clean vectorizations of technical drawings with a low number of primitives. The authors of [30] focus on speeding up sketch vectorization without loss of accuracy by applying an auxiliary grid and a summed area table. We compare to [3] and [11] which we found to be the best-performing methods in this class.

Neural network-based vectorization. To get the optimal result, the methods like [3,11] require manual tuning of hyper-parameters for each individual input image. In contrast, the neural network-based approach that we opt for is designed to process large datasets without tuning.

The method of [26] generates vectorized, semantically annotated floor plans from raster images using neural networks. At vectorization level, it detects a limited set of axis-aligned junctions and merges them, which is specific to a subset of floor plans (*e.g.*, does not handle diagonal or curved walls).

In [10] machine learning is used to extract a higher-level representation from a raster line drawing, specifically a program generating this drawing. This approach does not aim to capture the geometry of primitives faithfully and is restricted to a class of relatively simple diagrams.

A recent work [13] focuses on improving the accuracy of topology reconstruction. It extracts line junctions and the centerline image with a two headed convolutional neural network, and then reconstructs the topology at junctions with another neural network.

The algorithm of [12] has similarities to our method: it uses a neural network-based initialization for a more precise geometry fit for Bézier curve segments. Only simple input data (MNIST characters) are considered for line drawing reconstruction. The method was also applied to reconstructing 3D surfaces of revolution from images.

An interesting recent direction is generation of sketches using neural networks that learn a latent model representation for sketch images [14,43,19]. In principle, this approach can be used to approximate input raster images, but the geometric fidelity, in this case, is not adequate for most applications. In [42] an algorithm for generating collections of color strokes approximating an input photo is described. While this task is related to line drawing vectorization it is more forgiving in terms of geometric accuracy and representation compactness.

We note that many works on vectorization focus on sketches. Although the line between different types of line drawings is blurry, we found that methods focusing exclusively on sketches often produce less desirable results for technical line drawings (*e.g.*, [11] and [9]).

Vectorization datasets. Building a large-scale real-world vectorization dataset is costly and time-consuming [25,38]. One may start from raster dataset and create a vector ground-truth by tracing the lines manually. In this case, both location and the style may be difficult to match to the original drawing. Another way is to start from the vector image and render the raster image from it. This approach does not necessarily produce realistic raster images, as degradation suffered by real-world documents are known to be challenging to model [21]. As a result, existing vectorization-related datasets either lack vector annotation (*e.g.*, CVC-FP [17], Rent3D [27], SydneyHouse [7], and Raster-to-Vector [26] all provide semantic segmentation masks for raster images but not the vector ground truth) or are synthetic (*e.g.*, SESYD [8], ROBIN [37], and FPLAN-POLY [34]).

Image preprocessing. Building a complete vectorization system based on our approach requires the initial preprocessing step that removes imaging artefacts. Preprocessing tools available in commonly used graphics editors require manual parameter tuning for each individual image. For a similar task of conversion of hand-drawn sketches into clean raster line drawings the authors of [38,35] use convolutional neural networks trained on synthetic data. The authors of [25] use a neural network to extract structural lines (*e.g.*, curves separating image regions) in manga cartoon images. The general motivation behind the network-based approach is that a convolutional neural network automatically adapts to

different types of images and different parts of the image, without individual parameter tuning. We build our preprocessing step based on the ideas of [25,38].

Other related work. Methods solving other vectorization problems include, *e.g.*, [41,20], which approximate an image with adaptively refined constant color regions with piecewise-linear boundaries; [28] which extracts a vector representation of road networks from aerial photographs; [4] which solves a similar problem and is shown to be applicable to several types of images. These methods use strong build-in priors on the topology of the curve networks.

3 Our vectorization system

Our vectorization system, illustrated in Figure 1, takes as the input a raster technical drawing cleared of text and produces a collection of graphical primitives defined by the control points and width, namely line segments and quadratic Bézier curves. The processing pipeline consists of the following steps:

1. We preprocess the input image, removing the noise, adjusting its contrast, and filling in missing parts;
2. We split the cleaned image into patches and for each patch estimate the initial primitive parameters;
3. We refine the estimated primitives aligning them to the cleaned raster;
4. We merge the refined predictions from all patches.

3.1 Preprocessing of the input raster image

The goal of the preprocessing step is to convert the raw input data into a raster image with clear line structure by eliminating noise, infilling missing parts of lines, and setting all background/non-ink areas to white. This task can be viewed as semantic image segmentation in that the pixels are assigned the background or foreground class. Following the ideas of [25,38], we preprocess the input image with U-net [33] architecture, which is widely used in segmentation tasks. We train our preprocessing network in the image-to-image mode with binary cross-entropy loss.

3.2 Initial estimation of primitives

To vectorize a clean raster technical drawing, we split it into patches and for each patch independently estimate the primitives with a feed-forward neural network. The division into patches increases efficiency, as the patches are processed in parallel, and robustness of the trained model, as it learns on simple structures.

We encode each patch $I_p \in [0, 1]^{64 \times 64}$ with a ResNet-based [15] feature extractor $X^{\text{im}} = \text{ResNet}(I_p)$, and then decode the feature embeddings of the primitives X_i^{pr} using a sequence of n_{dec} Transformer blocks [40]

$$X_i^{\text{pr}} = \text{Transformer}(X_{i-1}^{\text{pr}}, X^{\text{im}}) \in \mathbb{R}^{n_{\text{prim}} \times d_{\text{emb}}}, \quad i = 1, \dots, n_{\text{dec}}. \quad (1)$$

Each row of a feature embedding represents one of the n_{prim} estimated primitives with a set of d_{emb} hidden parameters. The use of Transformer architecture allows to vary the number of output primitives per patch. The maximum number of primitives is set with the size of the 0th embedding $X_0^{\text{pr}} \in \mathbb{R}^{n_{\text{prim}} \times d_{\text{emb}}}$, initialized with positional encoding, as described in [40]. While the number of primitives in a patch is *a priori* unknown, more than 97% of patches in our data contain no more than 10 primitives. Therefore, we fix the maximum number of primitives and filter out the excess predictions with an additional stage. Specifically, we pass the last feature embedding to a fully-connected block, which extracts the coordinates of the control points, the widths of the primitives $\Theta = \{\theta_k = (x_{k,1}, y_{k,1}, \dots, w_k)\}_{k=1}^{n_{\text{prim}}}$, and the confidence values $\mathbf{p} \in [0, 1]^{n_{\text{prim}}}$. The latter indicate that the primitive should be discarded if the value is lower than 0.5. We detail more on the network in supplementary.

Loss function. We train the primitive extraction network with the multi-task loss function composed of binary cross-entropy of the confidence and a weighted sum of L_1 and L_2 deviations of the parameters

$$L(\mathbf{p}, \hat{\mathbf{p}}, \Theta, \hat{\Theta}) = \frac{1}{n_{\text{prim}}} \sum_{k=1}^{n_{\text{prim}}} \left(L_{\text{cls}}(p_k, \hat{p}_k) + L_{\text{loc}}(\theta_k, \hat{\theta}_k) \right), \quad (2)$$

$$L_{\text{cls}}(p_k, \hat{p}_k) = -\hat{p}_k \log p_k - (1 - \hat{p}_k) \log (1 - p_k), \quad (3)$$

$$L_{\text{loc}}(\theta_k, \hat{\theta}_k) = (1 - \lambda) \|\theta_k - \hat{\theta}_k\|_1 + \lambda \|\theta_k - \hat{\theta}_k\|_2^2. \quad (4)$$

The target confidence vector $\hat{\mathbf{p}}$ is all ones, with zeros in the end indicating placeholder primitives, all target parameters $\hat{\theta}_k$ of which are set to zero. Since this function is not invariant w.r.t. to permutations of the primitives and their control points, we sort the endpoints in each target primitive and the target primitives by their parameters lexicographically.

3.3 Refinement of the estimated primitives

We train our primitive extraction network to minimize the average deviation of the primitives on a large dataset. However, even with small average deviation, individual estimations may be inaccurate. The purpose of the refinement step is to correct slight inaccuracies in estimated primitives.

To refine the estimated primitives and align them to the raster image, we design a functional that depends on the primitive parameters and raster image and iteratively optimize it w.r.t. the primitive parameters

$$\Theta^{\text{ref}} = \underset{\Theta}{\operatorname{argmin}} E(\Theta, I_p). \quad (5)$$

We use physical intuition of attracting charges spread over the area of the primitives and placed in the filled pixels of the raster image. To prevent alignment of different primitives to the same region, we model repulsion of the primitives.

We define the optimized functional as the sum of three terms per primitive

$$E(\Theta^{\text{pos}}, \Theta^{\text{size}}, I_p) = \sum_{k=1}^{n_{\text{prim}}} E_k^{\text{size}} + E_k^{\text{pos}} + E_k^{\text{rdn}}, \quad (6)$$

where $\Theta^{\text{pos}} = \{\theta_k^{\text{pos}}\}_{k=1}^{n_{\text{prim}}}$ are the primitive position parameters, $\Theta^{\text{size}} = \{\theta_k^{\text{size}}\}_{k=1}^{n_{\text{prim}}}$ are the size parameters, and $\theta_k = (\theta_k^{\text{pos}}, \theta_k^{\text{size}})$.

We define the position of a line segment by the coordinates of its midpoint and inclination angle, and the size by its length and width. For a curve arc, we define the midpoint at the intersection of the curve and the bisector of the angle between the segments connecting the middle control point and the endpoints. We use the lengths of these segments, and the inclination angles of the segments connecting the “midpoint” with the endpoints.

Charge interactions. We base different parts of our functional on the energy of interaction of unit point charges $\mathbf{r}_1, \mathbf{r}_2$, defined as a sum of close- and far-range potentials

$$\varphi(\mathbf{r}_1, \mathbf{r}_2) = e^{-\frac{\|\mathbf{r}_1 - \mathbf{r}_2\|^2}{R_c^2}} + \lambda_f e^{-\frac{\|\mathbf{r}_1 - \mathbf{r}_2\|^2}{R_f^2}}, \quad (7)$$

parameters R_c, R_f, λ_f of which we choose experimentally. The energy of interaction of the uniform positively charged area of the k^{th} primitive Ω_k and a grid of point charges $\mathbf{q} = \{q_i\}_{i=1}^{n_{\text{pix}}}$ at the pixel centers \mathbf{r}_i is then defined by the following equation, that we integrate analytically for lines

$$E_k(\mathbf{q}) = \sum_{i=1}^{n_{\text{pix}}} q_i \iint_{\Omega_k} \varphi(\mathbf{r}, \mathbf{r}_i) d\mathbf{r}. \quad (8)$$

We approximate it for curves as the sum of integrals over the segments of the polyline flattening this curve.

In our functional we use three different charge grids, encoded as vectors of length n_{pix} : $\hat{\mathbf{q}}$ represents the raster image with charge magnitudes set to intensities of the pixels, \mathbf{q}_k represents the rendering of the k^{th} primitive with its current values of parameters, and \mathbf{q} represents the rendering of all the primitives in the patch. The charge grids \mathbf{q}_k and \mathbf{q} are updated at each iteration.

Energy terms. Below, we denote the componentwise product of vectors with \odot , and the vector of ones of an appropriate size with $\mathbf{1}$.

The first term is responsible for growing the primitive to cover filled pixels and shrinking it if unfilled pixels are covered, with fixed position of the primitive:

$$E_k^{\text{size}} = E_k([\mathbf{q} - \hat{\mathbf{q}}] \odot \mathbf{c}_k + \mathbf{q}_k \odot [\mathbf{1} - \mathbf{c}_k]). \quad (9)$$

The weighting $c_{k,i} \in \{0, 1\}$ enforces coverage of a continuous raster region following the form and orientation of the primitive. We set $c_{k,i}$ to 1 inside the largest region aligned with the primitive with only shaded pixels of the raster,

as we detail in supplementary. For example, for a line segment, this region is a rectangle centered at the midpoint of the segment and aligned with it.

The second term is responsible for alignment of fixed size primitives

$$E_k^{\text{pos}} = E_k ([\mathbf{q} - \mathbf{q}_k - \hat{\mathbf{q}}] \odot [\mathbf{1} + 3\mathbf{c}_k]). \quad (10)$$

The weighting here adjusts this term with respect to the first one, and subtraction of the rendering of the k^{th} primitive from the total rendering of the patch ensures that transversal overlaps are not penalized.

The last term is responsible for collapse of overlapping *collinear* primitives; for this term, we use $\lambda_f = 0$:

$$E_k^{\text{rdn}} = E_k (\mathbf{q}_k^{\text{rdn}}), \quad q_{k,i}^{\text{rdn}} = \exp \left(- [|\mathbf{l}_{k,i} \cdot \mathbf{m}_{k,i}| - 1]^2 \beta \right) \|\mathbf{m}_{k,i}\|, \quad (11)$$

where $\mathbf{l}_{k,i}$ is the direction of the primitive at its closest point to the i^{th} pixel, $\mathbf{m}_{k,i} = \sum_{j \neq k} \mathbf{l}_{j,i} q_{j,i}$ is the sum of directions of all the other primitives weighted w.r.t. their “presence”, and $\beta = (\cos 15^\circ - 1)^{-2}$ is chosen experimentally.

As our functional is based on many-body interactions, we can use an approximation well-known in physics — mean field theory. This translates into the observation that one can obtain an approximate solution of (5) by viewing interactions of each primitive with the rest as interactions with a static set of charges, *i.e.*, viewing each energy term E_k^{pos} , E_k^{size} , E_k^{rdn} as depending only on the parameters of the k^{th} primitive. This enables very efficient gradient computation for our functional, as one needs to differentiate each term w.r.t. a small number of parameters only. We detail on this heuristic in supplementary.

We optimize the functional (28) by Adam. For faster convergence, every few iterations we join lined up primitives by stretching one and collapsing the rest, and move collapsed primitives into uncovered raster pixels.

3.4 Merging estimations from all patches

To produce the final vectorization, we merge the refined primitives from the whole image with a straightforward heuristic algorithm. For lines, we link two primitives if they are close and collinear enough but not almost parallel. After that, we replace each connected group of linked primitives with a single least-squares line fit to their endpoints. Finally, we snap the endpoints of intersecting primitives by cutting down the “dangling” ends shorter than a few percent of the total length of the primitive. For Bézier curves, for each pair of close primitives we estimate a replacement curve with least squares and replace the original pair with the fit if it is close enough. We repeat this operation for the whole image until no more pairs allow a close fit. We detail on this process in supplementary.

4 Experimental evaluation

We evaluate two versions of our vectorization method: one operating with lines and the other operating with quadratic Bézier curves. We compare our method

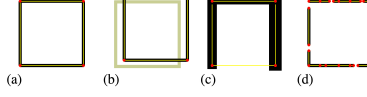


Fig. 2: (a) Ground-truth vector image, and artefacts w.r.t. which we evaluate the vectorization performance (b) skeleton structure deviation, (c) shape deviation, (d) overparameterization.

against FvS [11], CHD [9], and PVF [3]. We evaluate the vectorization performance with four metrics that capture artefacts illustrated in Figure 2.

Intersection-over-Union (IoU) reflects deviations in two raster shapes or rasterized vector drawings R_1 and R_2 via $\text{IoU}(R_1, R_2) = \frac{R_1 \cap R_2}{R_1 \cup R_2}$. It does not capture deviations in graphical primitives that have similar shapes but are slightly offset from each other.

Hausdorff distance

$$d_H(X, Y) = \max \left\{ \sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y) \right\}, \quad (12)$$

and **Mean Minimal Deviation**

$$d_M(X, Y) = \frac{1}{2} \left(\widetilde{d}_M(X \rightarrow Y) + \widetilde{d}_M(Y \rightarrow X) \right), \quad (13a)$$

$$\widetilde{d}_M(X \rightarrow Y) = \int_{x \in X} \inf_{y \in Y} d(x, y) dX \bigg/ \int_{x \in X} dX \quad (13b)$$

measure the difference in skeleton structures of two vector images X and Y , where $d(x, y)$ is Euclidean distance between a pair of points x, y on skeletons. In practice, we densely sample the skeletons and approximate these metrics on a pair of point clouds.

Number of Primitives $\#P$ measures the complexity of the vector drawing.

4.1 Clean line drawings

To evaluate our vectorization system on clean raster images with precisely known vector ground-truth we collected two datasets.

To demonstrate the performance of our method with lines, we compiled **PFP vector floor plan dataset** of 1554 real-world architectural floor plans from a commercial website [2].

To demonstrate the performance of our method with curves, we compiled **ABC vector mechanical parts dataset** using 3D parametric CAD models from ABC dataset [24]. They have been designed to model mechanical parts with sharp edges and well defined surface. We prepared $\approx 10k$ vector images via projection of the boundary representation of CAD models with the open-source software Open Cascade [1].

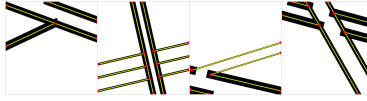


Fig. 3: Examples of synthetic training data for our primitive extraction network.

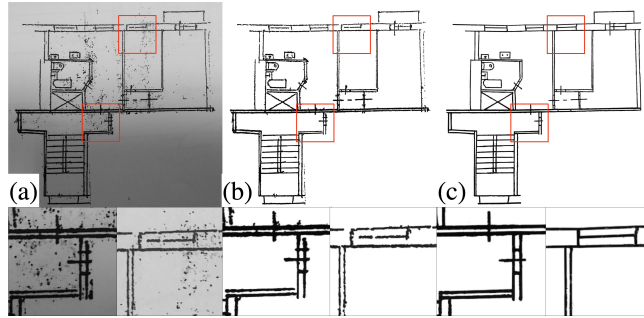


Fig. 4: Sample from DLD dataset: (a) raw input image, (b) the image cleaned from background and noise, (c) final target with infilled lines.

We trained our primitive extraction network on random 64×64 crops, with random rotation and scaling. We additionally augmented PFP with synthetic data, illustrated in Figure 3.

For evaluation, we used 40 hold-out images from PFP and 50 images from ABC with resolution $\sim 2000 \times 3000$ and different complexity per pixel. We specify image size alongside each qualitative result. We show the quantitative results of this evaluation in Table 1 and the qualitative results in Figures 5 and 6. Since the methods we compare with produce widthless skeleton, for fair comparison w.r.t. IoU we set the width of the primitives in their outputs equal to the average on the image.

There is always a trade-off between the number of primitives in the vectorized image and its accuracy, so the comparison of the results with different number of primitives is not fair. On PFP, our system outperforms other methods w.r.t. all metrics, and only loses in primitive count to FvS. On ABC, PVF outperforms our full vectorization system w.r.t. IoU, but not our vectorization method without merging, as we discuss below in ablation study. It also produces much more primitives than our method.

4.2 Degraded line drawings

To evaluate our vectorization system on real raster technical drawings, we compiled **Degraded line drawings dataset (DLD)** out of 81 photos and scans of floor plans with resolution $\sim 1300 \times 1000$. To prepare the raster targets, we manually cleaned each image, removing text, background, and noise, and refined the line structure, inpainting gaps and sharpening edges (Figure 4).

	PFP				ABC				DLD	
	IoU,%	d _H , px	d _M , px	#P	IoU,%	d _H , px	d _M , px	#P	IoU,%	#P
FvS [11]	31	381	2.8	696	65	38	1.7	63		
CHD [9]	22	214	2.1	1214	60	9	1	109	47	329
PVF [3]	60	204	1.5	38k	89	17	0.7	7818		
Our	86/88	25	0.2	1331	77/77	19	0.6	97	79/82	452

Table 1: Quantitative results of vectorization. For our method we report two values of IoU: with the average primitive width and with the predicted.

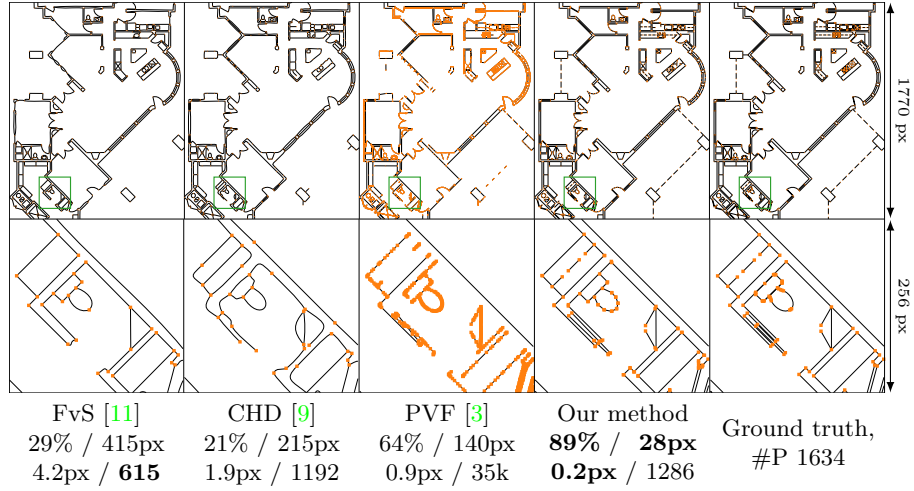


Fig. 5: Qualitative comparison on a PFP image, and values of IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.

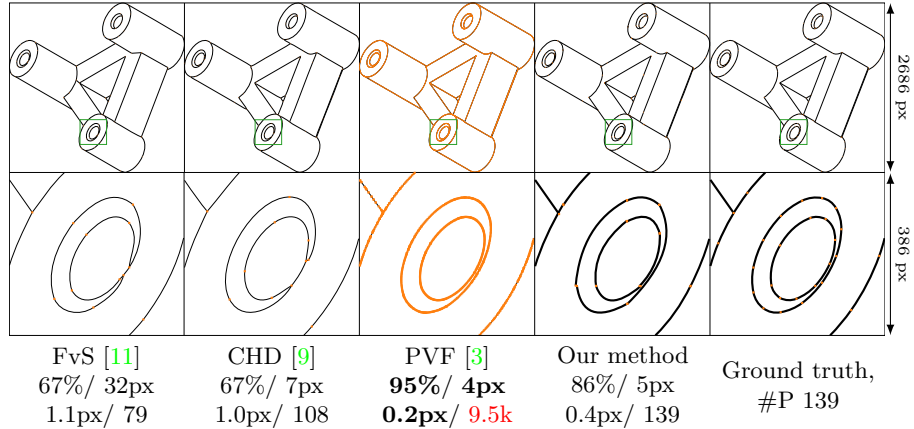


Fig. 6: Qualitative comparison on an ABC image, and values of IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.

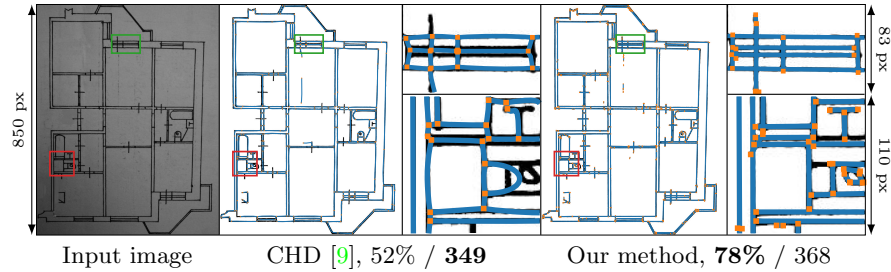


Fig. 7: Qualitative comparison on a real noisy image, and values of IoU / #P with best in bold. Primitives are shown in blue with endpoints in orange on top of the cleaned raster image.

	IoU,%	PSNR
MS [38]	49	15.7
Our	92	25.5

Table 2: Quantitative evaluation of the preprocessing step.

To train our preprocessing network, we prepared the dataset consisting of 20000 synthetic pairs of images of resolution 512×512 . We rendered the ground truth in each pair from a random set of graphical primitives, such as lines, curves, circles, hollow triangles, etc. We generated the input image via rendering the ground truth on top of one of 40 realistic photographed and scanned paper backgrounds selected from images available online, and degrading the rendering with random blur, distortion, noise, etc. After that, we fine-tuned the preprocessing network on DLD.

For evaluation, we used 15 hold-out images from DLD. We show the quantitative results of this evaluation in Table 1 and the qualitative results in Figure 7. Only CHD allows for degraded input so we compare with this method only. Since this method produces widthless skeleton, for fair comparison w.r.t. IoU we set the width of the primitives in its outputs equal to the average on the image, that we estimate as the sum of all nonzero pixels divided by the length of the predicted primitives.

Our vectorization system outperforms CHD on the real floor plans w.r.t. IoU and produces similar number of primitives.

Evaluation of preprocessing network. We separately evaluate our preprocessing network comparing with public pre-trained implementation of MS [38]. We show the quantitative results of this evaluation in Table 2 and qualitative results in Figure 8. Our preprocessing network keeps straight and repeated lines commonly found in technical drawing while MS produces wavy strokes and tends to join repeated straight lines, thus harming the structure of the drawing.

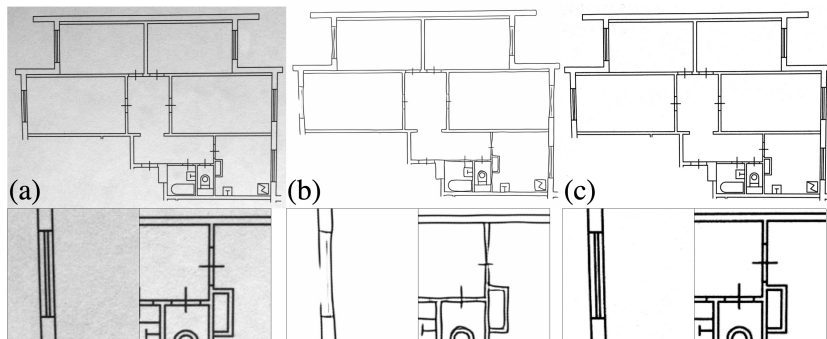


Fig. 8: Example of preprocessing results: (a) raw input image, (b) output of MS [38], (c) output of our preprocessing network. Note the tendency of MS to combine close parallel lines.

	IoU, %	d_H , px	d_M , px	#P
NN	65	52	1.4	309
NN + Refinement	91	19	0.3	240
NN + Refinement + Postprocessing	77	19	0.6	97

Table 3: Ablation study on ABC dataset. We compare the results of our method with and without refinement and postprocessing

4.3 Ablation study

To assess the impact of individual components of our vectorization system on the results, we obtained the results on the ABC dataset with the full system, the system without the postprocessing step, and the system without the postprocessing and refinement steps. We show the quantitative results in Table 3 and the qualitative results in Figure 20.

While the primitive extraction network produces correct estimations on average, some estimations are severely inaccurate, as captured by d_H . The refinement step improves all metrics, and the postprocessing step reduces the number of primitives but deteriorates other metrics due to the trade-off between number of primitives and accuracy.

We note that our vectorization method without the final merging step outperforms other methods on ABC dataset in terms of accuracy metrics.

5 Conclusion

We presented a four-part system for vectorization of technical line drawings, which produces a collection of graphical primitives defined by the control points and width. The first part is the preprocessing neural network that cleans the input image from artefacts. The second part is the primitive extraction network, trained on a combination of synthetic and real data, which operates on patches

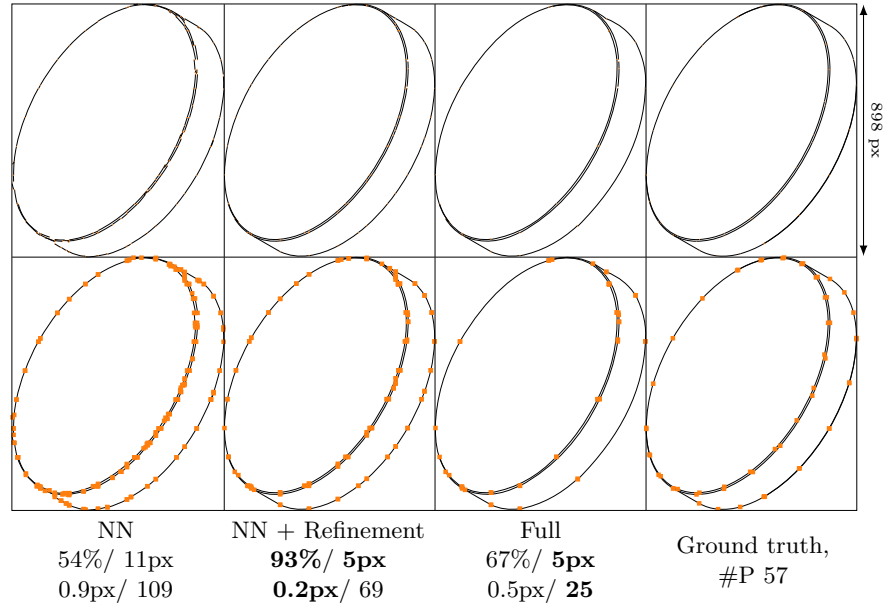


Fig. 9: Results of our method on an ABC image with and without refinement and postprocessing, and values of IoU / d_H / d_M / #P with best in bold. The endpoints of primitives are shown in orange.

of the image. It estimates the primitives approximately in the right location most of the time, however, it is generally geometrically inaccurate. The third part is iterative optimization, which adjusts the primitive parameters to improve the fit. The final part is heuristic merging, which combines the primitives from different patches into single vectorized image. The evaluation shows that our system, in general, performs significantly better compared to a number of recent vectorization algorithms.

Modifications of individual parts of our system would allow it to be applied to different, related tasks. For example, adjustment of the preprocessing network and the respective training data would allow for application of our system to extraction of wireframe from a photo. Modification of the optimized functional and use of the proper training data for primitive extraction network would allow for sketch vectorization. Integration with an OCR system would allow for separation and enhancement of text annotations.

Acknowledgements: We thank Milena Gazdieva and Natalia Soboleva for their valuable contributions in preparing real-world raster and vector datasets, as well as Maria Kolos and Alexey Bokhovkin for contributing parts of shared codebase used throughout this project. We acknowledge the usage of Skoltech CDISE HPC cluster Zhores for obtaining the presented results. The work was partially supported by Russian Science Foundation under Grant 19-41-04109.

References

1. Open CASCADE Technology OCCT. <https://www.opencascade.com/>, accessed: 2020-03-05 9
2. PrecisionFloorplan. <http://precisionfloorplan.com>, accessed: 2020-03-05 9
3. Bessmeltsev, M., Solomon, J.: Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics (TOG)* **38**(1), 9 (2019) 3, 9, 11, 18, 21, 22, 23, 24, 26, 27
4. Chai, D., Forstner, W., Lafarge, F.: Recovering line-networks in images by junction-point processes. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 1894–1901 (2013) 5
5. Chen, J., Du, M., Qin, X., Miao, Y.: An improved topology extraction approach for vectorization of sketchy line drawings. *The Visual Computer* **34**(12), 1633–1644 (2018) 3
6. Chen, J., Lei, Q., Miao, Y., Peng, Q.: Vectorization of line drawing image based on junction analysis. *Science China Information Sciences* **58**(7), 1–14 (2015) 3
7. Chu, H., Wang, S., Urtasun, R., Fidler, S.: Housecraft: Building houses from rental ads and street views. In: *European Conference on Computer Vision*. pp. 500–516. Springer (2016) 4
8. Delalandre, M., Valveny, E., Pridmore, T., Karatzas, D.: Generation of synthetic documents for performance evaluation of symbol recognition & spotting systems. *International Journal on Document Analysis and Recognition (IJDAR)* **13**(3), 187–207 (2010) 4
9. Donati, L., Cesano, S., Prati, A.: A complete hand-drawn sketch vectorization framework. *Multimedia Tools and Applications* **78**(14), 19083–19113 (2019) 4, 9, 11, 12, 18, 21, 22, 23, 24, 26, 27, 28
10. Ellis, K., Ritchie, D., Solar-Lezama, A., Tenenbaum, J.: Learning to infer graphics programs from hand-drawn images. In: *Advances in neural information processing systems*. pp. 6059–6068 (2018) 4
11. Favreau, J.D., Lafarge, F., Bousseau, A.: Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)* **35**(4), 120 (2016) 3, 4, 9, 11, 18, 21, 22, 23, 24, 26, 27
12. Gao, J., Tang, C., Ganapathi-Subramanian, V., Huang, J., Su, H., Guibas, L.J.: Deepspine: Data-driven reconstruction of parametric curves and surfaces. *arXiv preprint arXiv:1901.03781* (2019) 4
13. Guo, Y., Zhang, Z., Han, C., Hu, W.B., Li, C., Wong, T.T.: Deep line drawing vectorization via line subdivision and topology reconstruction. *Comput. Graph. Forum* **38**, 81–90 (2019) 4
14. Ha, D., Eck, D.: A neural representation of sketch drawings (2018) 4
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778 (2016) 5
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016) 18
17. de las Heras, L.P., Terrades, O.R., Robles, S., Sánchez, G.: Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing tool. *International Journal on Document Analysis and Recognition (IJDAR)* **18**(1), 15–30 (2015) 4

18. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis & Machine Intelligence* (6), 890–904 (2006) [3](#)
19. Kaiyrbekov, K., Sezgin, M.: Stroke-based sketched symbol reconstruction and segmentation. *arXiv preprint arXiv:1901.03427* (2019) [4](#)
20. Kansal, R., Kumar, S.: A vectorization framework for constant and linear gradient filled regions. *The Visual Computer* **31**(5), 717–732 (2015) [5](#)
21. Kanungo, T., Haralick, R.M., Baird, H.S., Stuehle, W., Madigan, D.: A statistical, nonparametric methodology for document degradation model validation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(11), 1209–1223 (2000) [4](#)
22. Kim, B., Wang, O., Öztireli, A.C., Gross, M.: Semantic segmentation for line drawing vectorization using neural networks. In: *Computer Graphics Forum*. vol. 37, pp. 329–338. Wiley Online Library (2018) [3](#)
23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014) [18](#)
24. Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., Panozzo, D.: Abc: A big cad model dataset for geometric deep learning. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 9601–9611 (2019) [9](#)
25. Li, C., Liu, X., Wong, T.T.: Deep extraction of manga structural lines. *ACM Transactions on Graphics (TOG)* **36**(4), 117 (2017) [4](#), [5](#)
26. Liu, C., Wu, J., Kohli, P., Furukawa, Y.: Raster-to-vector: revisiting floorplan transformation. In: *Proceedings of the IEEE International Conference on Computer Vision*. pp. 2195–2203 (2017) [3](#), [4](#)
27. Liu, C., Schwing, A.G., Kundu, K., Urtasun, R., Fidler, S.: Rent3d: Floor-plan priors for monocular layout estimation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 3413–3421 (2015) [4](#)
28. Mátyus, G., Luo, W., Urtasun, R.: Deeproadmapper: Extracting road topology from aerial images. In: *Proceedings of the IEEE International Conference on Computer Vision*. pp. 3438–3446 (2017) [5](#)
29. Munusamy Kabilan, V., Morris, B., Nguyen, A.: Vectordefense: Vectorization as a defense to adversarial examples. *arXiv preprint arXiv:1804.08529* (2018) [3](#)
30. Najgebauer, P., Scherer, R.: Inertia-based fast vectorization of line drawings. *Comput. Graph. Forum* **38**, 203–213 (2019) [3](#)
31. Noris, G., Hornung, A., Sumner, R.W., Simmons, M., Gross, M.: Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG)* **32**(1), 4 (2013) [3](#)
32. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., Garnett, R. (eds.) *Advances in Neural Information Processing Systems* 32, pp. 8024–8035. Curran Associates, Inc. (2019), <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf> [18](#)
33. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*. pp. 234–241. Springer (2015) [5](#), [18](#)

34. Rusiñol, M., Borràs, A., Lladós, J.: Relational indexing of vectorial primitives for symbol spotting in line-drawing images. *Pattern Recognition Letters* **31**(3), 188–201 (2010) 4
35. Sasaki, K., Iizuka, S., Simo-Serra, E., Ishikawa, H.: Learning to restore deteriorated line drawing. *The Visual Computer* **34**(6-8), 1077–1085 (2018) 4
36. Selinger, P.: Potrace: a polygon-based tracing algorithm. Potrace (online), <http://potrace.sourceforge.net/potrace.pdf> (2009-07-01) (2003) 3
37. Sharma, D., Gupta, N., Chattopadhyay, C., Mehta, S.: Daniel: A deep architecture for automatic analysis and retrieval of building floor plans. In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). vol. 1, pp. 420–425. IEEE (2017) 4
38. Simo-Serra, E., Iizuka, S., Ishikawa, H.: Mastering sketching: adversarial augmentation for structured prediction. *ACM Transactions on Graphics (TOG)* **37**(1), 11 (2018) 4, 5, 12, 13
39. Tange, O.: Gnu parallel - the command-line power tool. ;login: The USENIX Magazine **36**(1), 42–47 (Feb 2011), <http://www.gnu.org/s/parallel> 18
40. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017) 5, 6, 18
41. Zhao, J., Feng, J., Zhou, B.: Image vectorization using blue-noise sampling. In: Imaging and Printing in a Web 2.0 World IV. vol. 8664, p. 86640H. International Society for Optics and Photonics (2013) 5
42. Zheng, N., Jiang, Y., Huang, D.: Strokenet: Aneural painting environment 4
43. Zhou, T., Fang, C., Wang, Z., Yang, J., Kim, B., Chen, Z., Brandt, J., Terzopoulos, D.: Learning to doodle with stroke demonstrations and deep q-networks. In: BMVC. p. 13 (2018) 4

Appendix

We provide additional details on the neural network architecture and training process in Sections A and B. Details of our postprocessing can be found in Section C. We compare runtimes of the methods in Section D. In Section E we show the performance of [3,9,11] on small patches in comparison to whole images. We show example results of our system for cartoon drawings in Section F. We provide additional comparisons from the ablation study in Section H and additional comparisons with other methods in Section G. In Section I we describe our refinement algorithm in detail.

A Neural Networks architectures

For image cleaning we use U-net [33] encoder-decoder architecture. It consists of blocks of layers, each containing convolutional and batch normalization layers and ReLU activations. We use seven such blocks interleaved with MaxPool downsampling in the encoder, and seven blocks interleaved with nearest neighbor upsampling in the decoder. We connect the blocks of the encoder and decoder with the same resolution of feature maps with skip connections, as in the original U-net.

We build our primitive extraction network from two parts: the encoder consisting of ResNet18 blocks [16], which extracts features from the raster, and the decoder Transformer model [40], which estimates the primitive parameters. The architecture of our primitive extraction network is shown in Figure 10.

We use $n_{\text{res}} = 1$ ResNet18 block with $c = 64$ channels in each convolution. We use $n_{\text{dec}} = 8$ Transformer blocks with 4 heads of multi-head attention and 512 neurons in the last fully-connected layer.

We set the hidden dimensionality of the primitive representations in the Transformer part of the network d_{emb} equal to the number of primitive parameters, 6 for lines and 8 for curves: one for the width, one for the confidence value and the rest for coordinates of the control points. We keep the other hyperparameter values the same for lines and curves.

B Training details

We used Pytorch 1.2 for GPU computations and model training [32] and GNU Parallel to speed up the metric calculations [39] for our methods. We trained our models for 15 epochs on ABC dataset and 17 epochs on PFP dataset. The batch size was 128. We used Adam [23] for optimization with a scheduler with the same hyperparameters as in the original Transformer paper [40]. It took us approximately four days to train each model on a single Nvidia v100. To speed up the training, we pre-calculated all data augmentations, including cropping, and trained our model on this augmented data. First, we split original images into train, validation, and test sets. Then we cropped and augmented images to prevent overfitting.

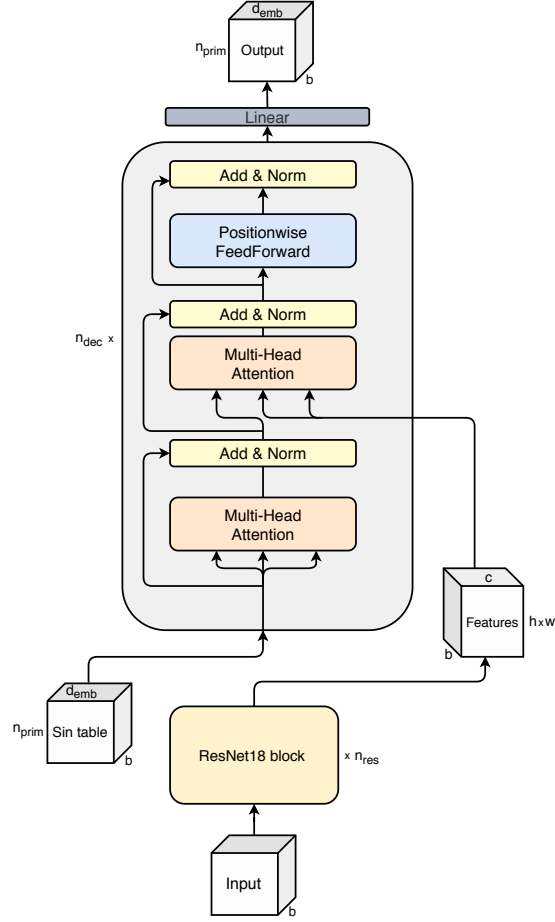


Fig. 10: Architecture of our vectorization network. A batch of b grayscale raster patches is first encoded with the sequence of n_{res} ResNet blocks. Then, c channel feature maps of size $h \times w$ are decoded with a sequence of n_{dec} Transformer blocks. Finally, the output of the last Transformer block is converted with a linear layer into n_{prim} sets of primitive parameters per sample in batch.

In Figure 11 and Figure 12 we provide metrics on train and validation sets for patches with size 64×64 from ABC and PFP datasets correspondingly.

C Merging algorithm

For lines we start by building a graph with the primitives as nodes and edges between nodes that correspond to a pair of lines that are close and collinear enough but not almost parallel (Figure 13 (a, b)). Then, we replace the lines in each connected component of the graph with a single least-squares line fit to

ABC Dataset

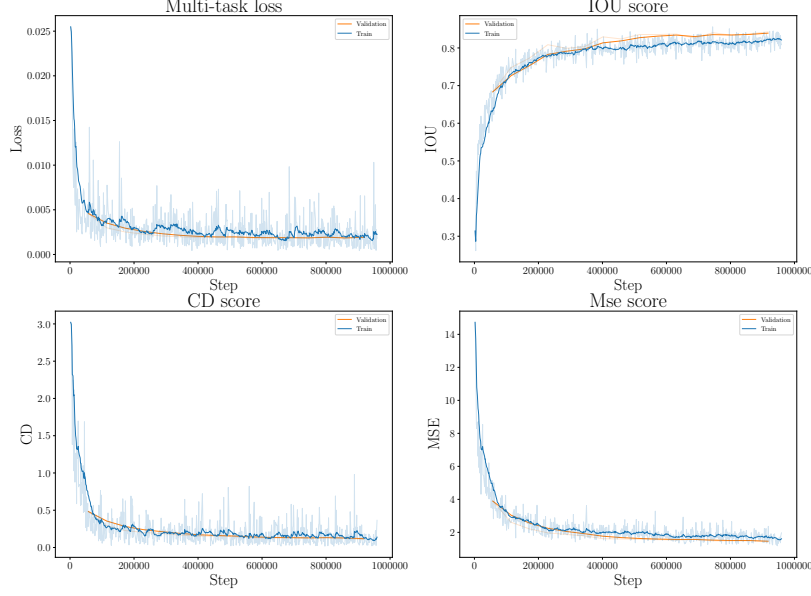


Fig. 11: Metrics and loss function for train and validation on ABC dataset. One step of X-axis represents calculations on a single batch.

their endpoints (Figure 13 (c, d)). Finally, we snap the endpoints of intersecting primitives by cutting down the “dangling” ends shorter than a few percent of the total length of the primitive. (Figure 13 (e)).

For quadratic Bézier curves, we iteratively try to replace pairs of curves with a single one. For each pair of curves $P(t)$, $t \in [0, 1]$, $Q(s)$, $s \in [0, 1]$, we first check if their widths are close (Figure 14 (b)). Then, we check if the “midpoint” (Figure 14 (a)) and the endpoints of the second curve are close to the first one, as illustrated in Figure 14 (c). If all checks are passed, we find a new quadratic Bézier curve $R(u)$, $u \in [0, 1]$ as a least-squares fit to the endpoints and midpoints of the curves in the pair (Figure 14 (d)). Specifically, we minimize the distances between the points

$$\begin{aligned} P_1 &= P(0), & P_b &= P(t_b), & P_3 &= P(1), \\ Q_1 &= Q(0), & Q_b &= Q(s_b), & Q_3 &= Q(1) \end{aligned} \quad (14)$$

and the points on the new curve

$$\begin{aligned} R(0), & \quad R(t_b u_{q1}/t_{q1}), & R(u_{q1}/t_{q1}), \\ R(u_{q1}), & \quad R(1 - (1 - s_b)(1 - u_{q1})), & R(1) \end{aligned} \quad (15)$$

respectively w.r.t. control points of the new curve. Here, t_b and t_{q1} are the parameter values of P_b and the projection of Q_1 on the first curve, s_b is the

PFP Dataset

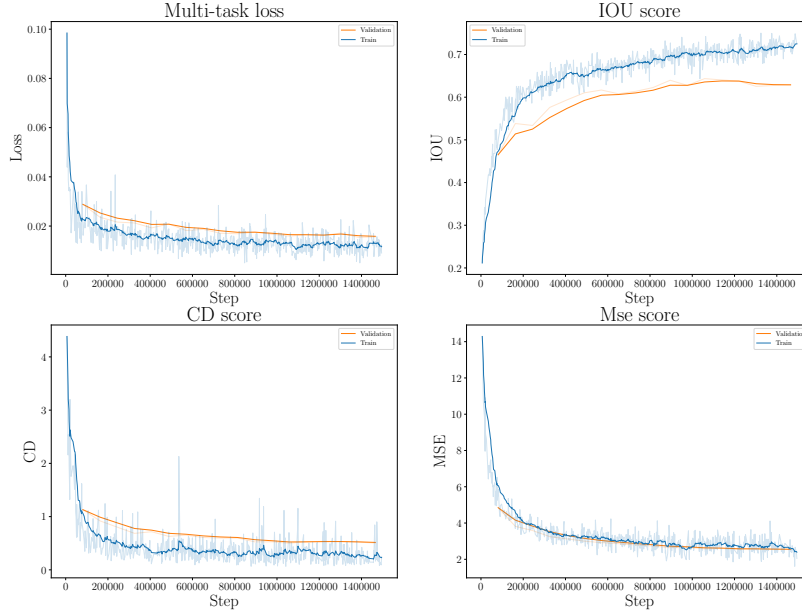


Fig. 12: Metrics and loss function for train and validation on PFP dataset. One step of the X-axis represents computations on a single batch.

parameter value of Q_b on the second curve, u_{q1} is the parameter value of Q_1 on the new curve. We find the value of u_{q1} with brute-force search and take the best fit. Finally, if the best fit is close enough, we replace the pair of the curves with the fit. We repeat this process until no more pairs allow for a close fit.

D Computation time

Our refinement step is iterative and allows trading longer computation times for more accurate results. In Table 4 we show example computation times for the prior work along with IoU values, and the computation times required by our system to reach similar IoU values.

Our system without the final merging step reaches the same IoU value as CHD [9] in a similar time, and the same IoU values as FvS [11] and PVF [3] in much less time. We note however that none of the methods were optimized for performance and that we run the methods in different environment because of technical requirements.

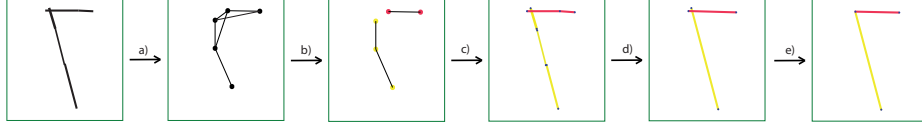


Fig. 13: Our algorithm of line merging: (a) we find close lines, (b, c) we join them in the connected components of the graph, (d) we fit the endpoints of the lines in each connected component with least squares, (e) and finally snap the endpoints of the lines.

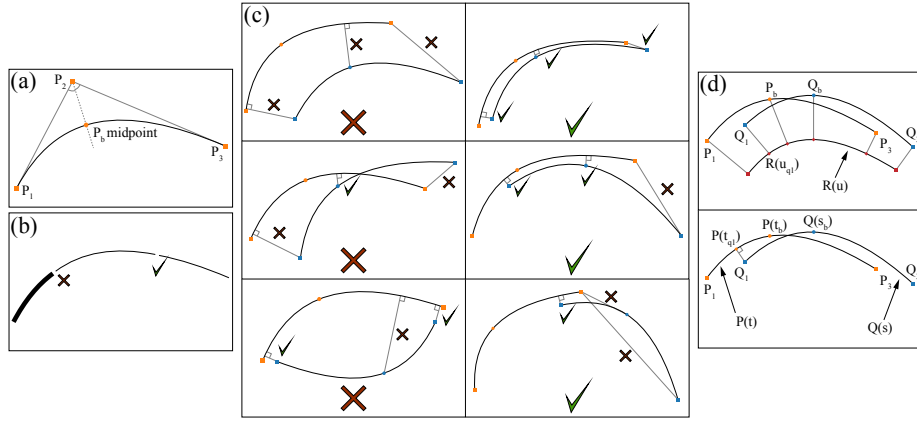


Fig. 14: (a) Our definition of the “midpoint” for quadratic Bézier curve, and (b-d) single step of our algorithm of curve merging: (b) we check that the widths are close, (c) we check that the curves are close, (d) we fit the endpoints and midpoints of the curves with least squares.

	IoU, %	Time	#P
CHD [9]	64	10 s	994
FvS [11]	74	17.5 m	433
PVF [3]	91	25 h	43k
Our, w/o final merging	68	35 s	2108
Our, w/o final merging	75	50 s	2106
Our, w/o final merging	91	5.5 m	1502
Our, w/o final merging, converged	92	12 m	1435
Our, with final merging	76	26 m	579

Table 4: IoU, computation time, and number of primitives for the results on the Globe (Figure 18) produced by the prior work, for intermediate results of our method with similar values of IoU, and for our final result.

E Prior work on patches

The main steps of our vectorization system, the primitive extraction network and refinement, operate on small patches of the image, while the methods that we compare with operate on whole images. To demonstrate that our method outperforms these ones not only because of this divide-and-merge strategy, in Figure 15 we show example outputs of these methods applied to small patches in comparison to the respective patches cut from the results on whole images.

The methods of [3,9] produce similar results on small patches and whole images, as expected since they use local operations. The method of [11] produces worse results on patches.

F Generalization to cartoon drawings

Figure 16 shows the results produced by our system on clean cartoon drawings. Here we used the version operating on curves, with the neural networks trained on technical drawings.

Our system produces reasonable results, although the predictions of the primitive extraction network are qualitatively less accurate than in case of technical drawings that we focused on. A proper extension of our system to a different kind of drawings would require (1) the corresponding training dataset for the primitive extraction network, and (2) in case of rough sketches, either a proper training dataset with clean targets for the preprocessing cleaning step, or significant changes of the refinement step.

G Additional results

In this section, we show more qualitative comparisons on test set for both PFP in Figure 17 and ABC in Figure 18 datasets and on real data in Figure 19.

H Qualitative ablation study

In this section, we show qualitative results obtained using our system with the (a) full model without refinement and post-processing steps, (b) full model without post-processing, (c) full model. You can see this comparison on ABC dataset in Figure 20 and Figure 21.

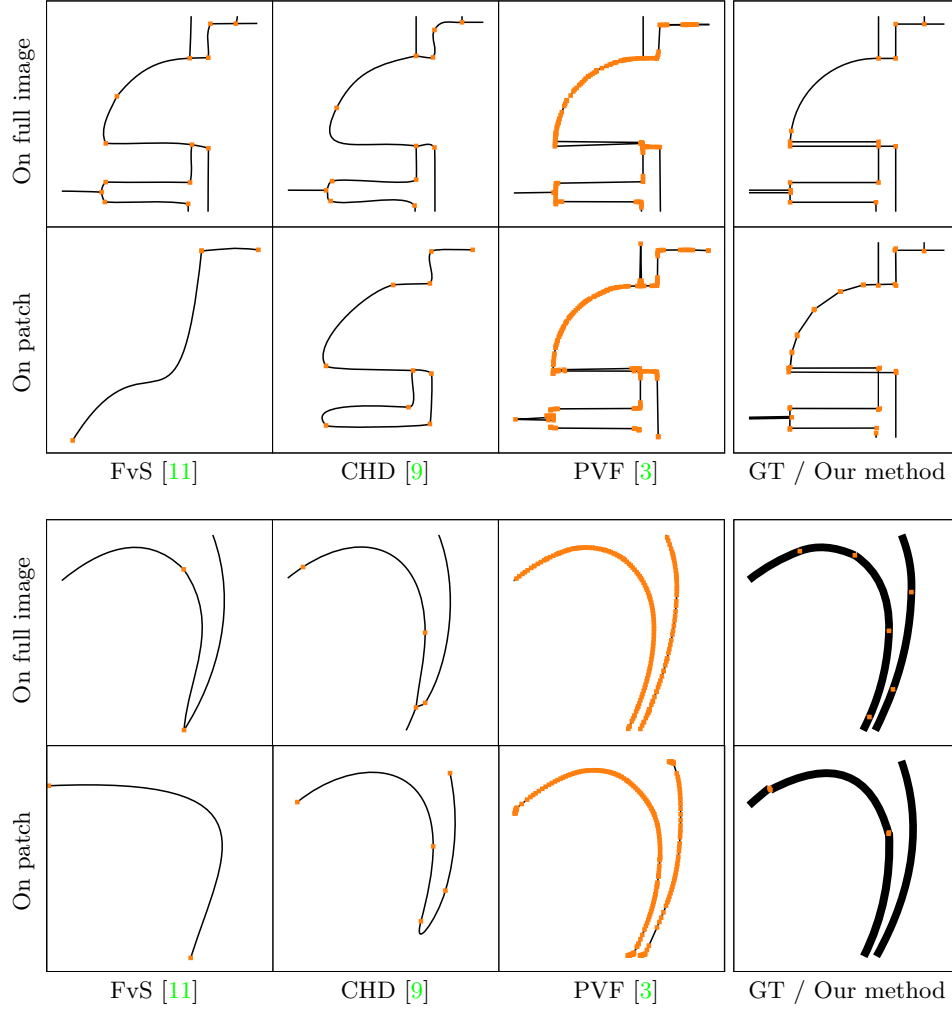


Fig. 15: Results of the prior work on small patches and the respective patches cut from the results on whole images. Endpoints of the primitives are shown in orange. The whole images are shown at the top of Figure 17 and in Figure 6 from the main text.

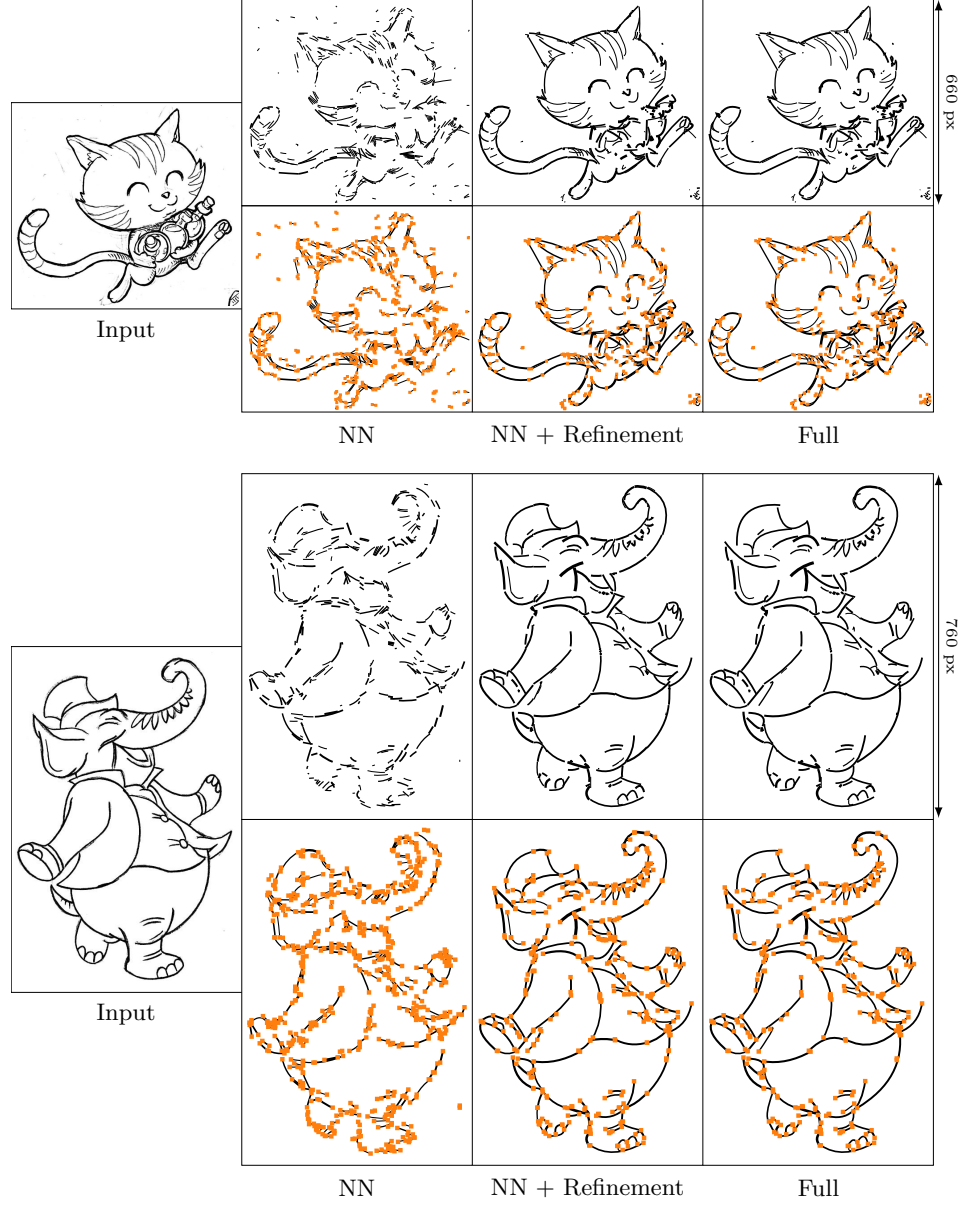


Fig. 16: Qualitative results of our system on clean cartoon drawings. Endpoints of primitives are shown in orange. The input image on the top is copyrighted by David Revoy www.davidrevoy.com under CC-by 4.0 license and on the bottom from www.easy-drawings-and-sketches.com, © Ivan Huska.

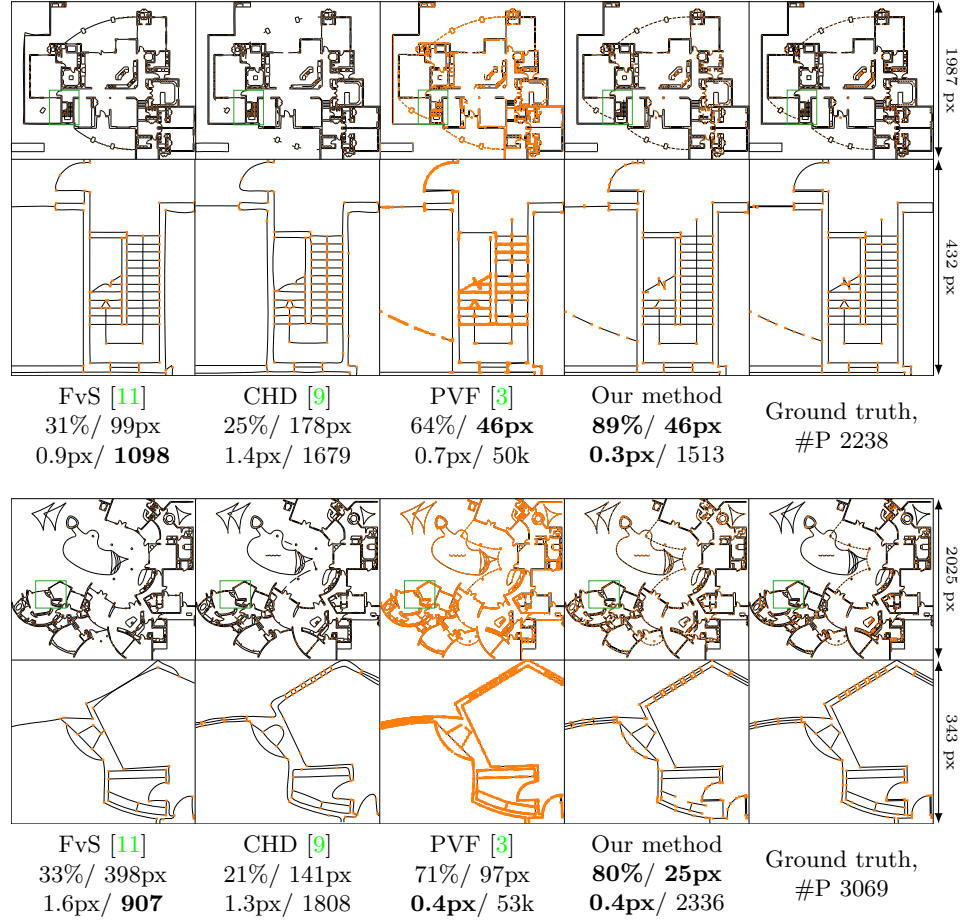


Fig. 17: Qualitative comparison on PFP images, and values of metrics IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.

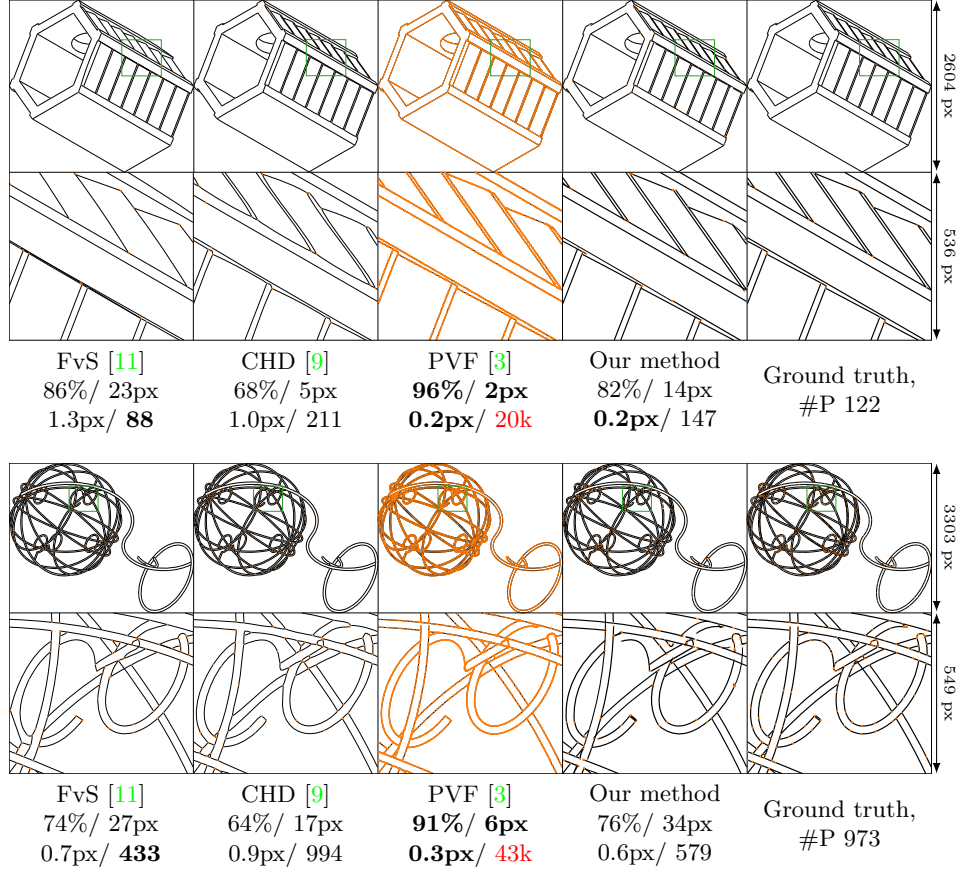


Fig. 18: Qualitative comparison on ABC images, and values of IoU / d_H / d_M / #P metrics, with the best result in boldface. The endpoints of primitives are shown in orange.

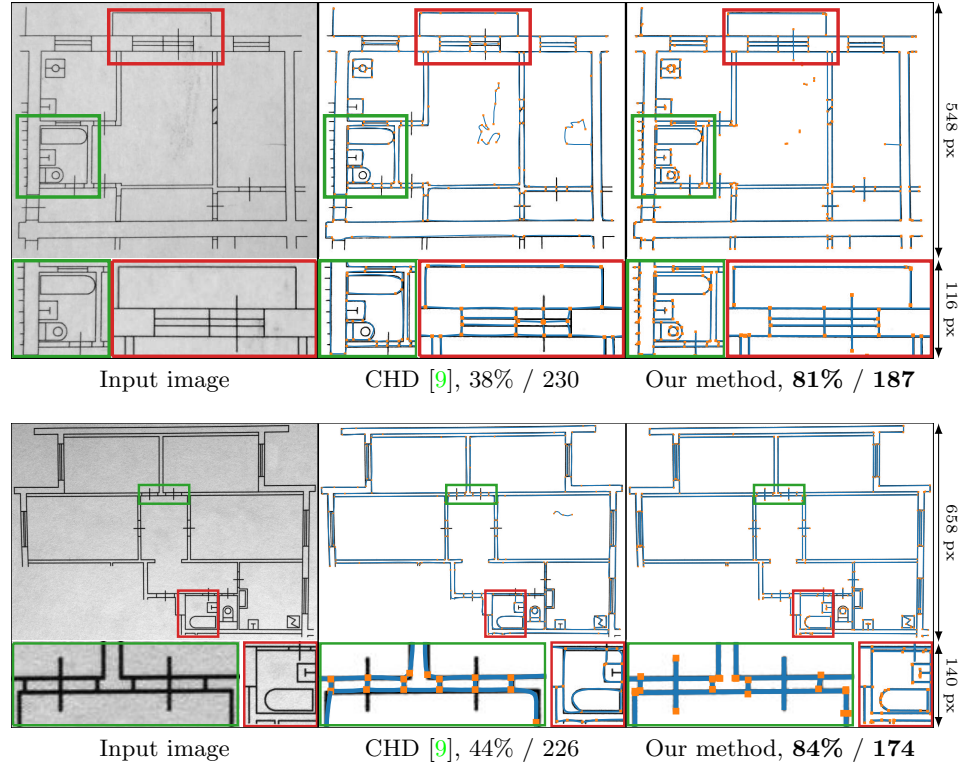


Fig. 19: Qualitative comparison on real noisy images, and values of metric IoU / #P with best in bold. Primitives are shown in blue with the endpoints in orange on top of the cleaned raster image.

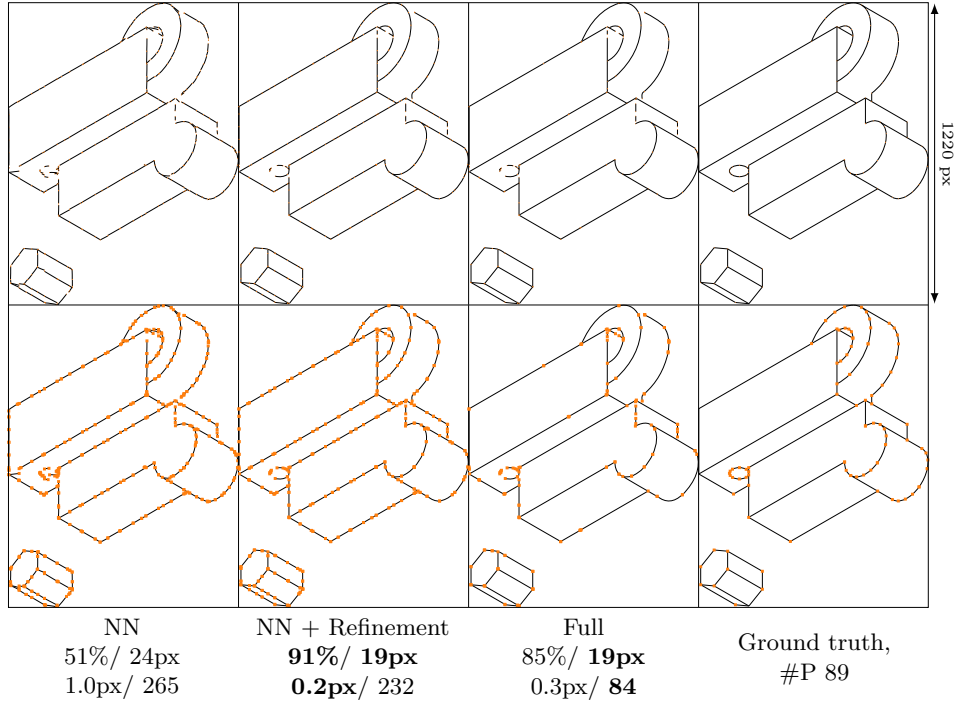


Fig. 20: Qualitative comparison on ABC images, and values of IoU / d_H / d_M / #P metrics, with the best results shown in boldface. The endpoints of primitives are shown in orange.

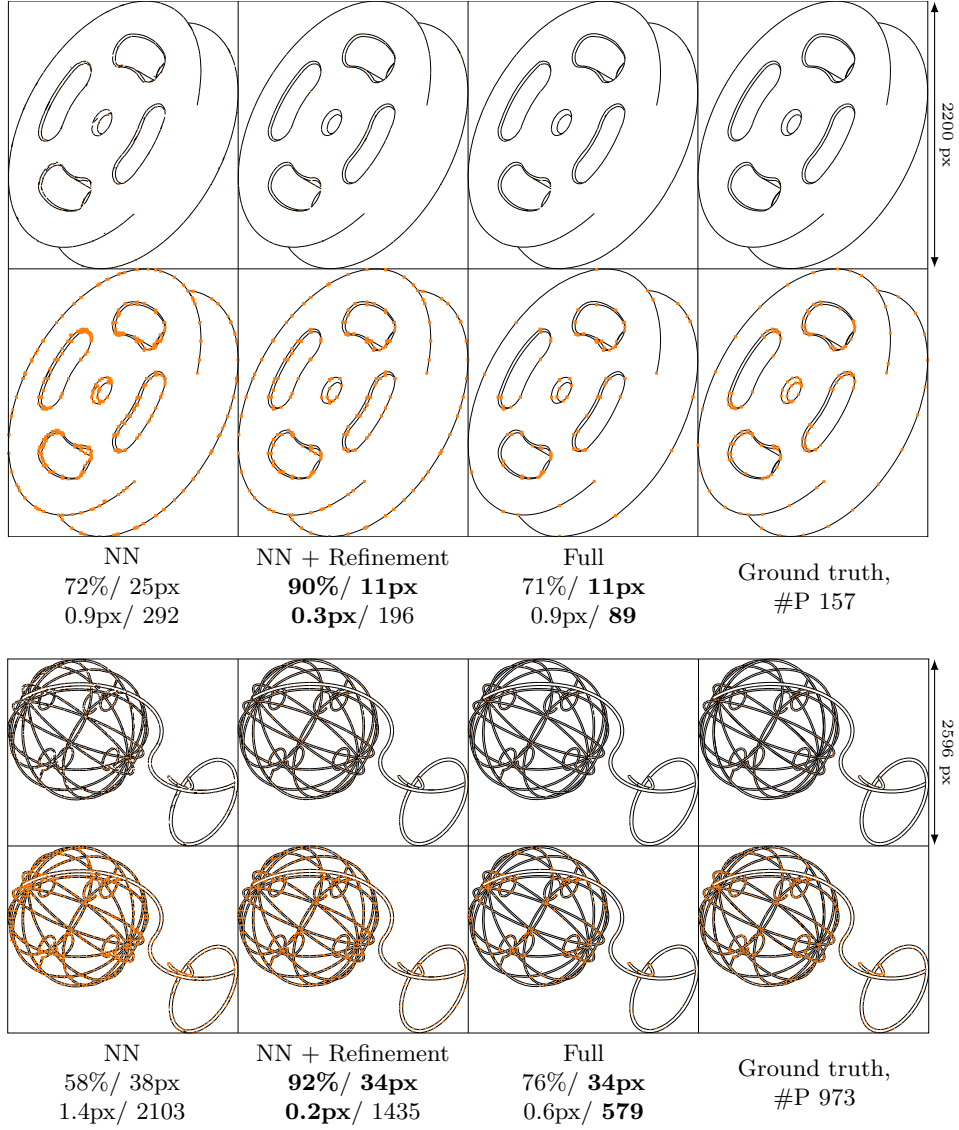


Fig. 21: Qualitative comparison on ABC images, and values of metrics IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.

I Details on refinement algorithm

I.1 Overall idea

The underlying idea in our approach is to use interaction potentials, qualitatively similar, *e.g.*, to electrostatic interaction, to construct our optimization functionals. Fixed charges are associated with filled pixels, and moving charges to the points on primitives. Primitives and filled pixels of the raster image are assigned charges of different signs: negative for pixels and positive for primitives. As a consequence, primitives and filled pixels are attracted, and primitives repulse other primitives. Internal charges push primitives to expand, because their internal charges are repulsing each other. A number of modifications need to be made to this general approach to avoid undesirable minima.

The interaction energy of two charges at points $\mathbf{r}_1, \mathbf{r}_2$ is given by

$$q_1 q_2 \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|), \quad (16)$$

where q_1, q_2 are signed charges, and $\varphi(r)$ is the interaction potential of two charges at the distance r from each other. The standard 3D electrostatic potential is $\frac{1}{r}$; we replace it by an exponentially decaying potential as explained at the end of this section. The total energy is obtained by summation/integration over all charge pairs.

Energy. We split our energy into three parts: primitive-pixel interactions, interactions between distinct primitives and interaction between charges inside the same primitive. As the charges at pixels do not move, their interactions with each other can be ignored.

$$E = \sum_{k_{\text{prim}}, i_{\text{pix}}} E_{k_{\text{prim}}, i_{\text{pix}}}^{\text{prim, pix}} + \sum_{k_{\text{prim}} < j_{\text{prim}}} E_{k_{\text{prim}}, j_{\text{prim}}}^{\text{prim, prim}} + \sum_{k_{\text{prim}}} E_{k_{\text{prim}}}^{\text{prim}}. \quad (17)$$

Three parts of the energy have the following form:

$$E_{k_{\text{prim}}, i_{\text{pix}}}^{\text{prim, pix}} = -\hat{q}_{i_{\text{pix}}} \iint_{\Omega_{k_{\text{prim}}}} \varphi(\|\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}\|) d\mathbf{r}^2, \quad (18)$$

where $\hat{q}_{i_{\text{pix}}}$ is the pixel intensity, $\mathbf{r}_{i_{\text{pix}}}$ and $\Omega_{k_{\text{prim}}}$ domain covered by the primitive;

$$E_{k_{\text{prim}}, j_{\text{prim}}}^{\text{prim, prim}} = \iint_{\Omega_{k_{\text{prim}}}} \iint_{\Omega_{j_{\text{prim}}}} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) d\mathbf{r}_1^2 d\mathbf{r}_2^2; \quad (19)$$

and

$$E_{k_{\text{prim}}}^{\text{prim}} = \frac{1}{2} E_{k_{\text{prim}}, k_{\text{prim}}}^{\text{prim, prim}} = \frac{1}{2} \iint_{\Omega_{k_{\text{prim}}}} \iint_{\Omega_{k_{\text{prim}}}} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) d\mathbf{r}_1^2 d\mathbf{r}_2^2. \quad (20)$$

Energy properties. Observe that pixel-primitive interaction is negative and decays (increases in magnitude) as primitive get close to a pixel, and also decreases as a primitive increase in size (more coverage is good). Primitive-primitive interaction energy is positive, decreases as the primitives move apart and also as the size of the primitives decreases. Finally, the self-interaction energy of a primitive is positive, does not depend on the primitive position and decreases if the primitive shrinks.

I.2 Mean-field-based optimization

For optimizing the energy efficiently, we use an approach based on a standard approach in the *mean-field* theory: the interactions between particles are viewed as individual interactions with a mean field, which is then updated using updated particle positions.

The basic gradient descent update of α^{th} parameter of k^{th} primitive is:

$$\theta_{k,\alpha} \leftarrow \theta_{k,\alpha} - \lambda \frac{\partial E}{\partial \theta_{k,\alpha}}. \quad (21)$$

We split the primitive parameters into size and position parameters and spell out the derivatives explicitly in each case, highlighting in blue the parts of the expressions that depend on the primitive.

$$\begin{aligned} \frac{\partial}{\partial \theta_{k,\alpha}^{\{\text{pos}, \text{size}\}}} \sum_{k_{\text{prim}}, i_{\text{pix}}} E_{k_{\text{prim}}, i_{\text{pix}}}^{\text{prim}, \text{pix}} &= \partial \sum_{i_{\text{pix}}} E_{k, i_{\text{pix}}}^{\text{prim}, \text{pix}} = \\ &= - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \partial \iint_{\Omega_k} \varphi(\|\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}\|) dr^2, \end{aligned} \quad (22)$$

$$\begin{aligned} \frac{\partial}{\partial \theta_{k,\alpha}^{\{\text{pos}, \text{size}\}}} \sum_{k_{\text{prim}} < j_{\text{prim}}} E_{k_{\text{prim}}, j_{\text{prim}}}^{\text{prim}, \text{prim}} &= \partial \sum_{j_{\text{prim}} \neq k} E_{k, j_{\text{prim}}}^{\text{prim}, \text{prim}} = \\ &= \partial \iint_{\Omega_k} \sum_{j_{\text{prim}} \neq k} \iint_{\Omega_{j_{\text{prim}}}} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) dr_1^2 dr_2^2, \end{aligned} \quad (23)$$

$$\frac{\partial}{\partial \theta_{k,\alpha}^{\text{pos}}} \sum_{k_{\text{prim}}} E_{k_{\text{prim}}}^{\text{prim}} = 0, \quad (24)$$

$$\begin{aligned} \frac{\partial}{\partial \theta_{k,\alpha}^{\text{size}}} \sum_{k_{\text{prim}}} E_{k_{\text{prim}}}^{\text{prim}} &= \partial E_k^{\text{prim}} = \\ \frac{1}{2} \partial \iint_{\Omega_k} \iint_{\Omega_k} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) dr_1^2 dr_2^2 &+ \frac{1}{2} \partial \iint_{\Omega_k} \iint_{\Omega_k} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) dr_1^2 dr_2^2 = \\ \partial \iint_{\Omega_k} \iint_{\Omega_k} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) dr_1^2 dr_2^2, \end{aligned} \quad (25)$$

The complete expressions for the energy derivatives with respect to positional parameters are:

$$\begin{aligned} \frac{\partial E}{\partial \theta_{k,\alpha}^{\text{pos}}} &= \partial \sum_{i_{\text{pix}}} E_{k,i_{\text{pix}}}^{\text{prim,pix}} + \partial \sum_{j_{\text{prim}} \neq k} E_{k,j_{\text{prim}}}^{\text{prim,prim}} = \\ &\partial \iint_{\Omega_k} \left[\sum_{j_{\text{prim}} \neq k} \iint_{\Omega_{j_{\text{prim}}}} \varphi(\|\mathbf{r} - \mathbf{r}_1\|) dr_1^2 - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \varphi(\|\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}\|) \right] dr^2 \end{aligned} \quad (26)$$

For size parameters, we obtain the following expression

$$\begin{aligned} \frac{\partial E}{\partial \theta_{k,\alpha}^{\text{size}}} &= \partial \sum_{i_{\text{pix}}} E_{k,i_{\text{pix}}}^{\text{prim,pix}} + \partial \sum_{j_{\text{prim}}} E_{k,j_{\text{prim}}}^{\text{prim,prim}} = \\ &\partial \iint_{\Omega_k} \left[\sum_{j_{\text{prim}} \in \Omega_{j_{\text{prim}}}} \iint \varphi(\|\mathbf{r} - \mathbf{r}_1\|) dr_1^2 - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \varphi(\|\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}\|) \right] dr^2, \end{aligned} \quad (27)$$

where j_{prim} ranges over all primitives including k

We can interpret these derivatives as derivatives of a different function

$$E^* = \sum_k E_k^{\text{pos}} + E_k^{\text{size}}. \quad (28)$$

with terms defined below. Each term corresponds to particular parameters of one of the primitives, and can be viewed as the interaction energy of the primitive with a background charge distribution defined by all primitives at a given instance in time.

$$E_k(q) = \iint_S q(\mathbf{r}_1) \iint_{\Omega_k} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) dr_2^2 dr_1^2, \quad (29)$$

$$E_k^{\text{pos}} = E_k(q_k^{\text{pos}})|_{\theta_k^{\text{size}}=\text{const}}, \quad E_k^{\text{size}} = E_k(q_k^{\text{size}})|_{\theta_k^{\text{pos}}=\text{const}}, \quad (30)$$

$$q_k^{\text{pos}}(\mathbf{r}) = \sum_{j_{\text{prim}} \neq k} \mathbb{1}[\mathbf{r} \in \Omega_{j_{\text{prim}}}] - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \delta(\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}), \quad (31)$$

$$q_k^{\text{size}}(\mathbf{r}) = \sum_{j_{\text{prim}}} \mathbb{1}[\mathbf{r} \in \Omega_{j_{\text{prim}}}] - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \delta(\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}), \quad (32)$$

where $\mathbb{1}[\cdot]$ is the Iverson bracket, and δ is the delta-function.

Expressions (28)-(32) provide the physics-based foundation for our optimization: at every step, we use the new form of the energy terms to obtain the gradients using automatic differentiation; the “frozen” parts of each term are updated after parameter update at every step. In this initial form, the functional has a number of undesirable properties for our application; we make several modifications described in the next section.

I.3 Discretization and functional modifications

Discretization. While for simple primitives the integrals in (28)-(32) can be computed explicitly, we simplify the problem by using discrete charges instead of continuous distributions.

The expression (28) becomes equation (8) from the submission.

$$\iint_S q(\mathbf{r}_1) \iint_{\Omega_k} \varphi(\|\mathbf{r}_1 - \mathbf{r}_2\|) dr_2^2 dr_1^2 \longrightarrow \sum_{i_{\text{pix}}} q_{i_{\text{pix}}} \iint_{\Omega_k} \varphi(\|\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}\|) dr^2 \quad (33)$$

Expressions (31) and (32) become

$$\iint_S \mathbb{1}[\mathbf{r} \in \Omega_k] f(\mathbf{r}) dr^2 \longrightarrow \sum_{i_{\text{pix}}} q_{k,i_{\text{pix}}} f(\mathbf{r}_{i_{\text{pix}}}), \quad (34)$$

$$\iint_S \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \delta(\mathbf{r} - \mathbf{r}_{i_{\text{pix}}}) f(\mathbf{r}) dr^2 \longrightarrow \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} f(\mathbf{r}_{i_{\text{pix}}}), \quad (35)$$

$$q_k^{\text{pos}}(\mathbf{r}) \longrightarrow q_{k,i_{\text{pix}}}^{\text{pos}} = \sum_{j_{\text{prim}} \neq k} q_{j_{\text{prim}},i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \quad (36)$$

$$q_k^{\text{size}}(\mathbf{r}) \longrightarrow q_{k,i_{\text{pix}}}^{\text{size}} = \sum_{j_{\text{prim}}} q_{j_{\text{prim}},i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \quad (37)$$

where $\hat{q}_{i_{\text{pix}}}$ is the coverage of the $i_{\text{pix}}^{\text{th}}$ raster image pixel, and $q_{k_{\text{prim}},i_{\text{pix}}}$ is the coverage of the $k_{\text{prim}}^{\text{th}}$ primitive in $i_{\text{pix}}^{\text{th}}$ pixel.

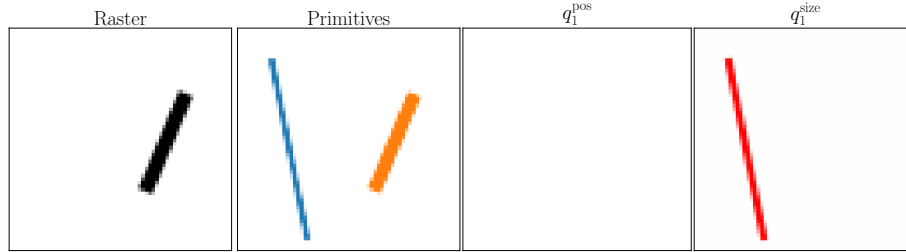


Fig. 22: Raster, primitives and charge grids of the first primitive. First primitive in blue, second in orange. In charge grids red represents excess charge.

Charge saturation. The charge distributions $\mathbf{q}_k^{\text{pos}} = \{q_{k,i_{\text{pix}}}^{\text{pos}}\}_{i_{\text{pix}}}$, $\mathbf{q}_k^{\text{size}} = \{q_{k,i_{\text{pix}}}^{\text{size}}\}_{i_{\text{pix}}}$ are excess or insufficient charges that need to be compensated by

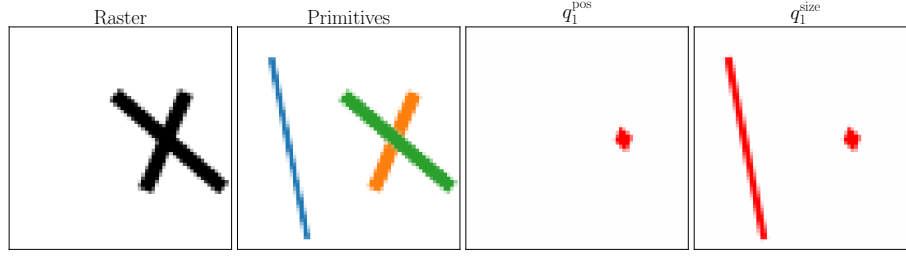


Fig. 23: Overlap affection on other primitives. First primitive in blue, second in orange. In charge grids red represents excess charge.

changing the k^{th} primitive. The energy terms corresponding to a primitive should not be affected by how many primitives cover a particular filled area. For example, in Figure 22, the second primitive covers the filled part of the raster perfectly, and this area does not affect the placement and size of the first primitive. Figure 23, the second and third primitives are covering the area equally well, but because of the overlap, the sum of their charges is higher than the negative charge of the raster image, and this creates a force acting on the first primitive.

To avoid the excess charge, we replace the sum of the charges with the maximum, leading to the following modification:

$$q_{k,i_{\text{pix}}}^{\text{pos}} = q_{-k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \quad (38)$$

$$q_{k,i_{\text{pix}}}^{\text{size}} = q_{i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \quad (39)$$

where $q_{i_{\text{pix}}}$ is the sum of coverages of $i_{\text{pix}}^{\text{th}}$ pixel for all primitives, and $q_{-k,i_{\text{pix}}}$ is the same sum with k^{th} primitive excluded.

Compared to (36), (37), modified charge distributions (38), (39) do not penalize overlaps.

Illustrative examples. Next, we consider several examples illustrating the behavior of the functional, which also help us to explain the modifications we make.

An isolated primitive.

For a single primitive (Figure 24) the position energy term is constant and does not depend on the parameters, so it would not move. The size energy term becomes lower with size: the primitive collapses to a point.

Primitive separated from the filled areas of the raster image. For a single primitive sufficiently far from the filled part of the image (Figure 25) the energy is decreasing if the primitive moves to the raster due to the second term in (38). If the primitive shrinks, the first term in (39) decreases and the second term increases. However, as we use fast-decaying potentials, we can neglect the interactions with distant charges, and overall the energy favors size reduction.

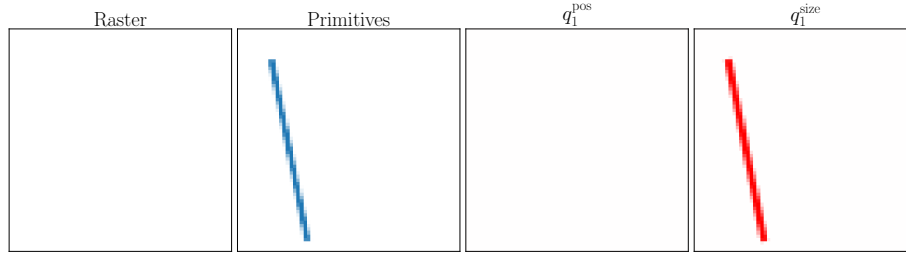


Fig. 24: Single primitive collapse. In charge grids red represents excess charge.

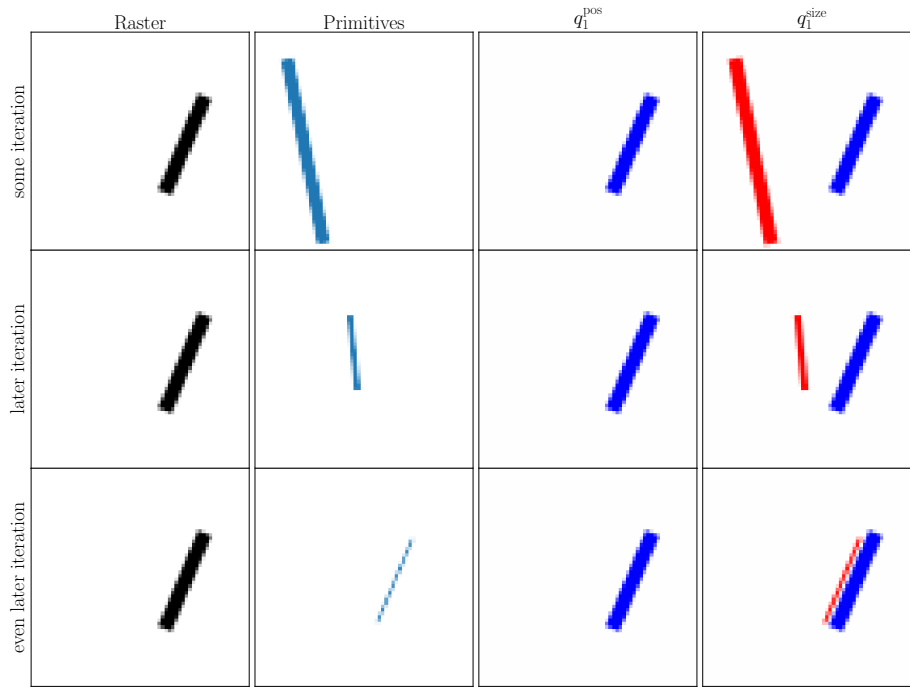


Fig. 25: Primitive interaction with far raster filled part. In charge grids red represents excess charge and blue represents uncovered raster.

A primitive close to a raster image. If a primitive is close to a filled part of the raster it will get aligned to the filled pixels, if these form a line and will increase in size until it covers the filled area (Figure 26). If we add additional filled pixels or other primitives at a distance, due to potential decay, they will have a minimal effect on the behavior.

Primitive aligned with a filled part of the primitive. In this case, the potentials from the raster image and the first primitive compensate each other, and the second primitive will not be affected by either, as in scenario of I.3.

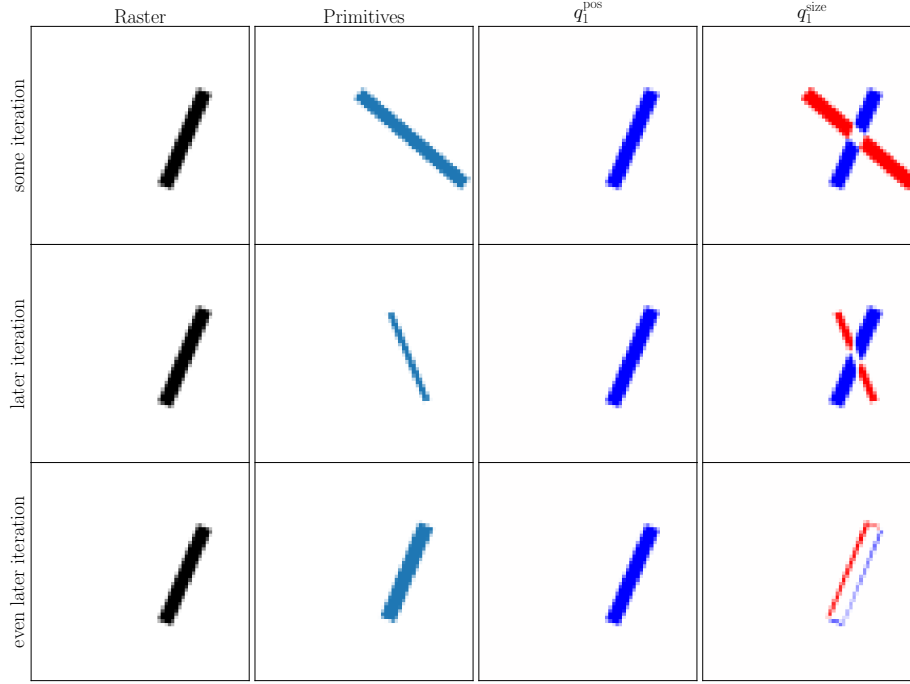


Fig. 26: Primitive interaction with close raster filled part. In charge grids red represents excess charge and blue represents uncovered raster.

Enabling primitive intersections. Consider the case in Figure 27. In this case, it would be desirable for the second primitive to cover the entire horizontal line, but this will not happen, as the second primitive, with the original energy formulation will remain close to its initial state. Primitive 1 instead of expanding will not change much either, as primitive 2 prevents its expansion.

To achieve the desired effect, i.e., expansion of the second primitive, we modify q_k^{pos} as follows:

$$q_{k,i_{\text{pix}}}^{\text{pos}} = q_{i_{\text{pix}}} - q_{k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}. \quad (40)$$

With this definition, the primitive interacts with pixels covered by other primitives, which allows it to expand across already filled areas, as in Figure 28

Penalty for overlapping collinear primitives E_k^{rdn} . By itself, (40) allows not just transversal intersections, but also aligned primitives covering the same area (Figure 29). We add an additional penalty E_k^{rdn} to avoid this. This term is also based on the interaction of the primitive with a background charge excess/deficiency, in this case, created by nearby primitives with tangents close to its tangent at close points. We define this term for segments first, and then generalize to curves.

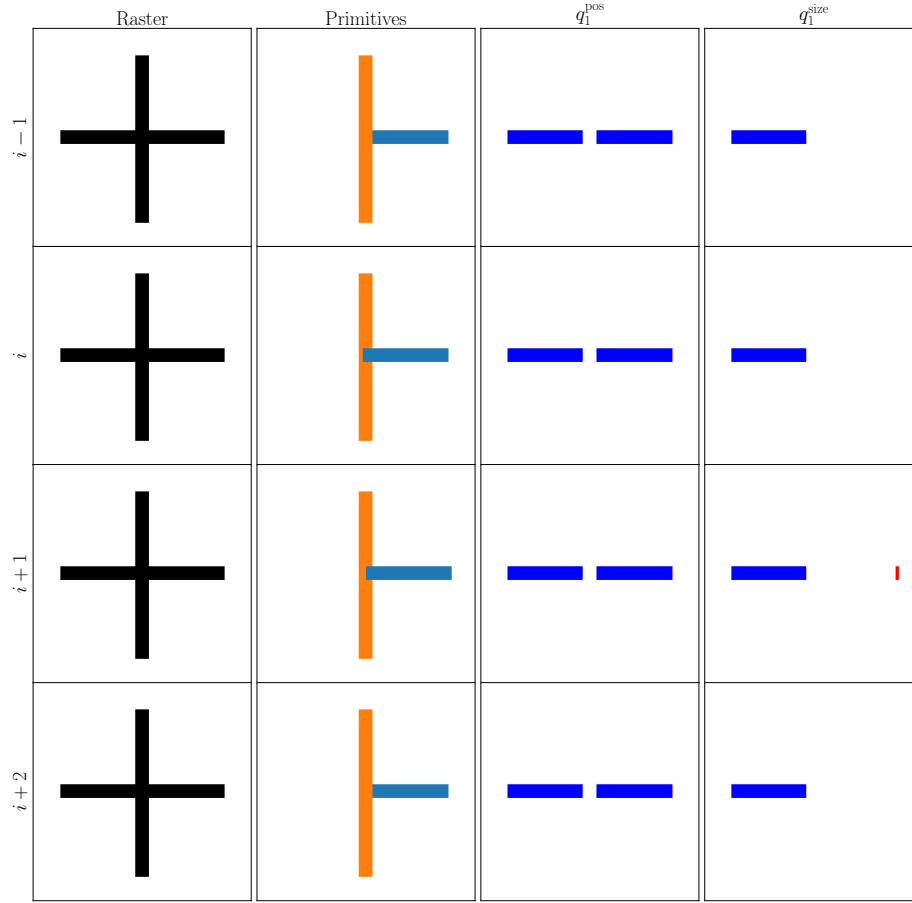


Fig. 27: Primitives interaction with disable primitive intersections. In charge grids red represents excess charge and blue represents uncovered raster.

Collinear penalty for line segments. For a system of two segments k and j we define it as

$$E_k^{\text{rdn}} = E_k(q_k^{\text{rdn}}) \Big|_{\text{close } \varphi(r), \theta_k^{\text{pos}} = \text{const}}, \quad (41)$$

$$q_{k,i_{\text{pix}}}^{\text{rdn}} = q_{j,i_{\text{pix}}} \exp \left(- \frac{(|\mathbf{l}_k \cdot \mathbf{l}_j| - 1)^2}{(|\cos \alpha_{\text{col}}| - 1)^2} \right), \quad (42)$$

where $\text{close } \varphi(r)$ means that we truncate the interaction at a fixed radius $\varphi(r|r > r^*) = 0$, as explained below, and charges $q_{k,i_{\text{pix}}}^{\text{rdn}}$ are defined by j^{th} primitive weighted by the cosine of the angle between segment directions, $\mathbf{l}_k, \mathbf{l}_j$, and α_{col} is a threshold angle for collinearity detection.

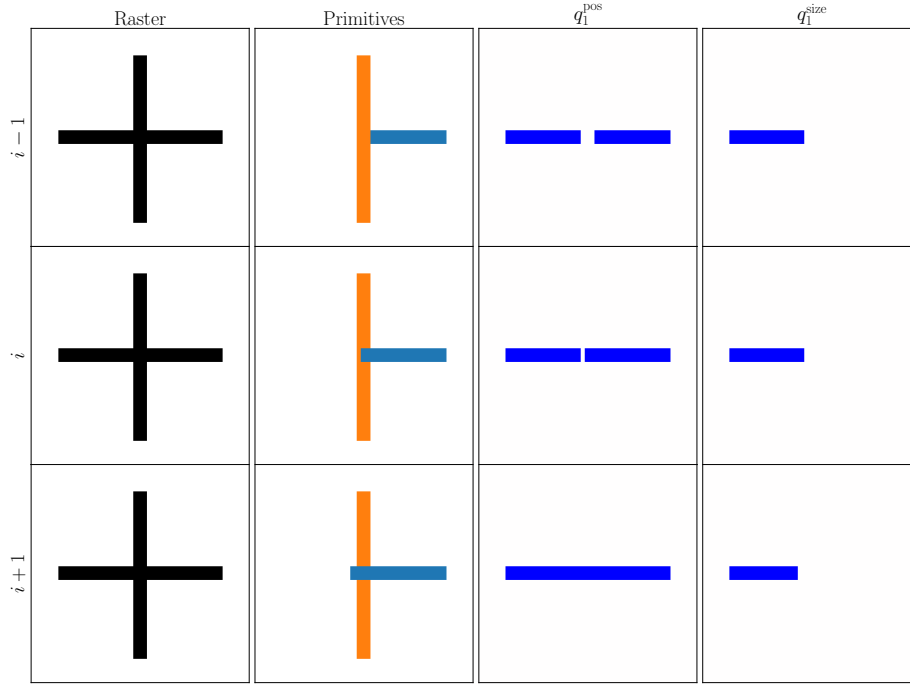


Fig. 28: Primitives interaction with enabled primitive intersections. In charge grids red represents excess charge and blue represents uncovered raster.

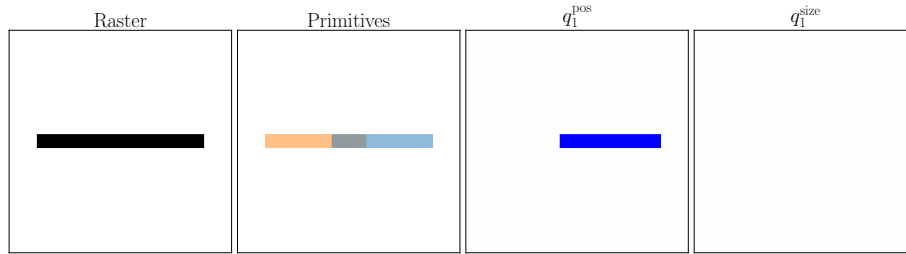


Fig. 29: Primitives covering the same area.

For many segments, we define the charges as

$$q_{k,i_{\text{pix}}}^{\text{rdn}} = \|\mathbf{m}_{k,i_{\text{pix}}}\| \exp \left(-\frac{(|\mathbf{l}_k \cdot \mathbf{m}_{k,i_{\text{pix}}}| - 1)^2}{(|\cos \alpha_{\text{col}}| - 1)^2} \right), \quad (43)$$

where $\mathbf{m}_{k,i_{\text{pix}}} = \sum_{j \neq k} \mathbf{l}_j q_{j,i_{\text{pix}}}$ is the sum of directions of all the other primitives weighted w.r.t. the mean direction of other primitives.

Collinearity penalization for curved segments. For curves, we use a similar idea, but need to use a different direction for every pixel, $\mathbf{l}_{k,i}$

$$q_{k,i_{\text{pix}}}^{\text{rdn}} = \|\mathbf{m}_{k,i_{\text{pix}}}\| \exp\left(-\frac{(\|\mathbf{l}_{k,i_{\text{pix}}}\| \cdot \|\mathbf{m}_{k,i_{\text{pix}}}\| - 1)^2}{(|\cos \alpha_{\text{col}}| - 1)^2}\right), \quad (44)$$

$$\mathbf{m}_{k,i_{\text{pix}}} = \sum_{j \neq k} \mathbf{l}_{j,i_{\text{pix}}} q_{j,i_{\text{pix}}}.$$

This definition reduces to the definition for segments if the curve is straight.

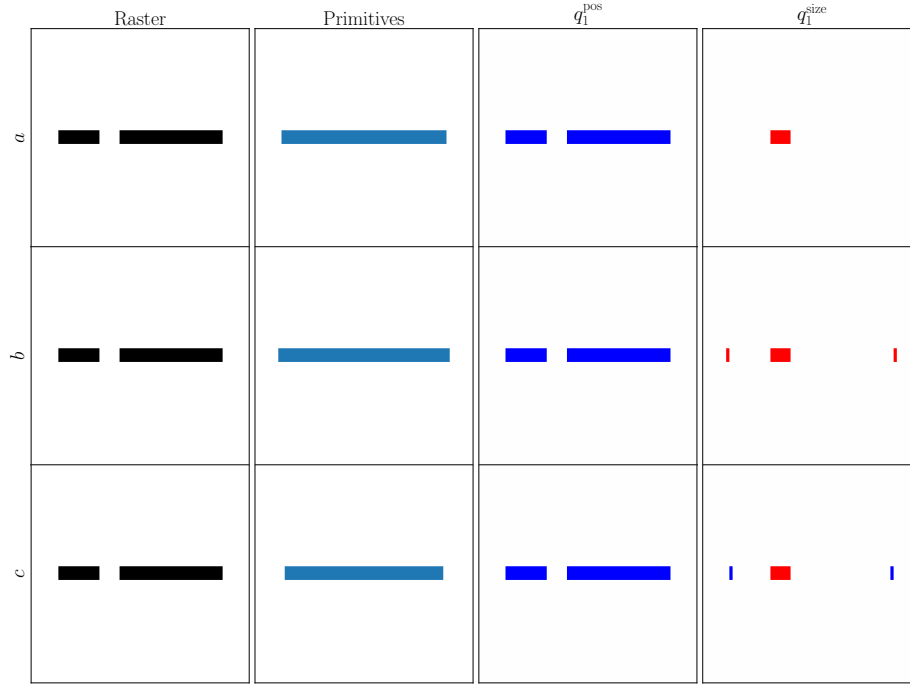


Fig. 30: Undesirable local minimum in which a primitive covers two disconnected raster parts.

Connected area mask. Consider the example in Figure 30 (a). Clearly the position and width of the primitive cannot be changed so that the energy decreases. Same is true for length: If the length increases or decreases (Figure 30 (b,c)), a counteracting force immediately appears because of the charge excess or deficit. We conclude that this is a local minimum, but clearly not a desirable solution. To reduce the chances of a primitive getting stuck in such a minimum, we limit the interaction of the primitive with the raster to the part that it can cover for

its current position and orientation, but arbitrary size. To be more precise, for line segments (Figure 31), for a fixed position and orientation, we find unfilled pixels along the line of the segment closest to its center, determining the length of the area. Once this is determined, then we find unfilled pixels closest to the line of the segment on two sides. This determines the rectangle for which the mask coefficient $c_{k,i_{\text{pix}}}$ is set to one, and for the rest of the image to zero.

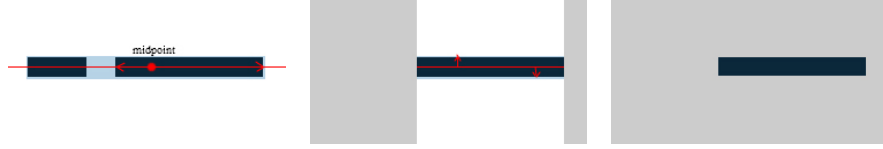


Fig. 31: Stages of $c_{k,i_{\text{pix}}}$ calculation.

For second order Bézier curves, we use a similar definition. Instead of a line we use a parabola containing the Bézier segment, and the distance along the parabola instead of the Euclidean distance.

The charge distribution for the size terms of the energy is redefined as follows

$$q_{k,i_{\text{pix}}}^{\text{size}} = \begin{cases} q_{i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}} & \text{if } c_{k,i_{\text{pix}}} = 1, \\ q_{k,i_{\text{pix}}} & \text{if } c_{k,i_{\text{pix}}} = 0. \end{cases} \quad (45)$$

Connected area mask for positional terms. If we use the same masks for the charges used for the positional terms $q_{k,i_{\text{pix}}}^{\text{pos}}$ eliminating the influence of everything outside the area where $c_{k,i_{\text{pix}}}$ is positive, then the primitive will stay within the filled area they initially overlap, which may be undesirable if the initial position is inaccurate, or multiple primitives initially cluster in the same place. For this reason, we only amplify the charge in the masked area leaving it the same outside:

$$q_{k,i_{\text{pix}}}^{\text{pos}} = \begin{cases} \lambda_{\text{pos}} (q_{i_{\text{pix}}} - q_{k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}) & \text{if } c_{k,i_{\text{pix}}} = 1, \\ q_{i_{\text{pix}}} - q_{k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}} & \text{if } c_{k,i_{\text{pix}}} = 0, \end{cases} \quad (46)$$

The amplification coefficient λ_{pos} is chosen empirically.

The choice of the potential function. The main property of the potential function $\varphi(r)$ used in our algorithm is rapid monotonic decrease with distance. We use the function

$$\varphi(r) = e^{-\frac{r^2}{R_c^2}} + \lambda_f e^{-\frac{r^2}{R_f^2}}, \quad (47)$$

which allows us to control interactions at close range $\sim R_c$ independently from interactions at far range $\sim R_f$. We choose these ranges and the weight experimentally $R_c = 1 \text{ px}$, $R_f = 32 \text{ px}$, $\lambda_f = 0.02$, and in E_k^{rdn} disable the far interactions by setting $\lambda_f = 0$.