Deep Vectorization of Technical Drawings

Vage Egiazarian*[1], Oleg Voynov*[1], Alexey Artemov[1], Denis Volkhonskiy[1], Aleksandr Safin[1], Maria Taktasheva[1], Denis Zorin[2,1], and Evgeny Burnaev[1]

[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.voinov, a.artemov, denis.volkhonskiy, aleksandr.safin, maria.taktasheva }@skoltech.ru, dzorin@cs.nyu.edu, e.burnaev@skoltech.ru

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 often constructed, 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 (such as missing parts of line segments and noise) in the images;
- 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, our goal is to develop a set of algorithms for geometrically precise and compact reconstruction of vector representations of technical drawings in a fully automatic way and performing significantly better, according to a number of metrics, than existing state-of-the-art algorithms. 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 lines approximating a single line) other than imaging flaws.

* Equal contribution

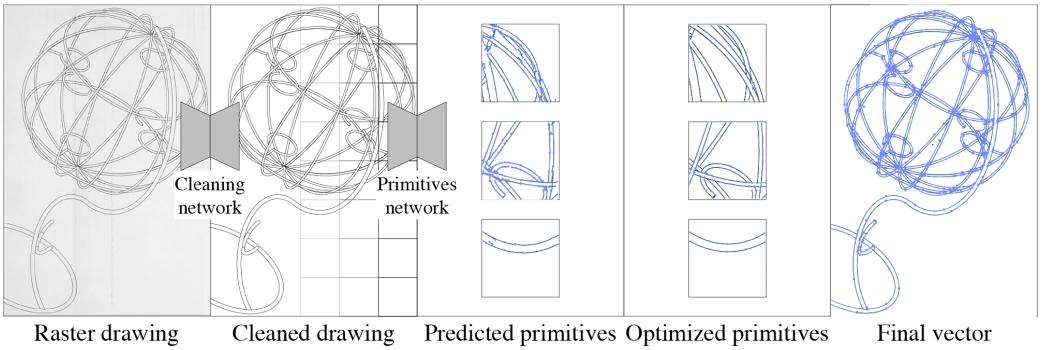


Fig. 1: An overview of our vectorization method. We clean the input image using a deep CNN splitting the input image into patches, we predict primitive placement with a deep CNN; we further refine the predictions using an optimization method, merging them together into the final vectorization.

Our vectorization pipeline includes components addressing vectorization goals listed above. The central element of the pipeline is a deep-learning accelerated optimization method matching geometric primitives to the raster image. This component addresses the key goal of finding a compact representation of a part of the rasterized image (a *patch*) with few vector graphics primitives. It is preceded by a learning-based image cleaning stage, removing background and noise as well as performing infill of missing parts of the image. It is followed by a postprocessing stage, aimed at further reducing the number of primitives by merging primitives discovered 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 a Potrace [34]. 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 [28] and [21] use Potrace as a stage in their algorithms.

Methods based on curve network topology extraction and cleanup include [6,11,5,29,17]. We compare to [3] and [11] which we found to be the best-performing methods in this class.

A recent sketch vectorization method [3] uses a polyvector field (crossfield) to guide the orientation of primitives. The method applies a sophisticated set of tuneable heuristics for curve network construction and topology cleanup, which are difficult to tune to produce clean vectorizations of technical drawings with a low number of primitives.

Neural network-based vectorization. Neural networks were applied in the context of vectorization in a variety of ways.

[25] is a method relying on neural networks for generating vectorized semantically annotated floor plans from raster images. At vectorization level, this method works by detecting a limited set of axis-aligned junctions and postprocessing to merge them, which is an approach 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. The paper does not aim to capture the geometry of primitives faithfully and is restricted to a class of relatively simple diagrams.

An interesting recent direction is the generation of sketches using neural networks that learn a latent model representation for sketch images [13,41,18]. 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.

The algorithm of [12] has similarities to our method: it uses a neural network-based initialization for a more precise geometry fit for Bezier 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.

In [40] algorithms for generating collections of color strokes approximating an input photo are described. While this is a task 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 [24,36]. 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 obtain a raster image by rendering the vector. The latter approach does not necessarily produce realistic raster images, as degradation suffered by real-world documents are known to be challenging to model [20].

As a result, existing vectorization-related datasets either lack vector-graphics annotation (*e.g.*, CVC-FP [16], Rent3D [26], SydneyHouse [7], and Raster-to-Vector [25] all provide semantic segmentation masks for raster images but not the vector ground truth) or are synthetic (*e.g.*, SESYD [8], ROBIN [35], and FPLAN-POLY [32]).

Image cleaning. The initial image cleaning is a crucial stage in our algorithm. Both [36] and related work [33], describe a GAN to convert hand-drawn raster images of sketches to clean raster line drawings. This method can be used as a preprocessing stage for vectorization analogous to our cleaning stage. We compare to this method in Section 4. Similar to our method, these approaches rely on training on synthetic data. The method of [24] uses a neural network to extract structural lines (*e.g.*, curves separating image regions) in manga cartoon images.

Other related work. Methods solving other vectorization problems include, e.g., [39] and [19], which approximates an image with adaptively refined constant color regions with piecewise-linear boundaries. [27] extracts a vector representation of road networks from aerial photographs; [4] also 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 framework

We first outline the entire framework and then discuss each aspect in detail.

Our framework (see Fig. 1) takes a raster technical drawing cleared of text as an input. It produces a collection of graphical primitives, namely line segments and quadratic Beźier curves

defined by the control points and width. Our overall processing pipeline consists of the following steps:

1. We clean the input image from noise, adjust the contrast and fill in missing parts;
2. We split the cleaned image into patches. For each patch we estimate initial primitive placement;
3. Refine the predictions;
4. Merge refined predictions from all patches and snap close endpoints of different primitives.

3.1 Preprocessing of input raster images

The goal of the cleaning stage is to produce a raster image that has a more clearly visible line structure compared to the input unprocessed raw data, eliminate noise, and remove gaps in lines, when these are imaging artifacts. All background/non-ink areas should get set to white.

Raster-cleaning task can be viewed as a semantic image segmentation in that the pixels of the input image are assigned one of several classes (in our case, background, *i.e.* whitespace, and foreground, *i.e.* ink). Inspired by [24] we used fully-convolutional network U-net [31]. It is considered to be the state-of-the-art in segmentation tasks, as it provide a way to take into account the image structure at multiple resolutions. Our raster-cleaning model is implemented as a U-net with skip-connections trained in an image-to-image mode. We train our raster cleaning models using a mixed dataset consisting of synthesized and real-world images described in Section 4.1.

3.2 Estimation of primitives with neural network

To vectorize a clean raster technical drawing, we split it into square fixed-size patches. For each patch we independently estimate the primitives with a feed-forward CNN. The division into small patches increases efficiency, as the patches are processed in parallel. It also increases the 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 [14] feature extractor $X^{\text{im}} = \text{ResNet}\,(I_p)$. Then we decode it tensors representing feature embeddings of the primitives X_i^{pr} using a sequence of n_{dec} Transformer blocks [38]

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

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 primitives to be found in the patch. The maximal number of primitives is set by choosing the size of the initial embedding X_0^{pr}, the values of which are initialized with positional encoding, as described in [38]. While the exact number of primitives in a patch is not known *a priori*, more than 97% of patches in our data contain no more than 10 primitives. Therefore, we fix the maximal 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. It extracts the coordinates of the control points and the width of the primitives $\Theta = \{\boldsymbol{\theta}_k = (x_{k,1}, y_{k,1}, \ldots, w_k)\}_{k=1}^{n_{\text{prim}}}$, and the confidence values $\boldsymbol{p} \in [0,1]^{n_{\text{prim}}}$ that indicate that the primitive should be discarded if the value is lower than 0.5.

Loss function. To train the primitive extraction network we utilize a multi-task loss function. It is composed of binary cross-entropy of the confidence and a weighted sum of L_1 and L_2 deviations of the parameters

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

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

$$L_{\text{loc}}\left(\boldsymbol{\theta}_k, \hat{\boldsymbol{\theta}}_k\right) = (1-\lambda)\left\|\boldsymbol{\theta}_k - \hat{\boldsymbol{\theta}}_k\right\|_1 + \lambda\|\boldsymbol{\theta}_k - \hat{\boldsymbol{\theta}}_k\|_2^2, \tag{4}$$

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

3.3 Refinement of estimations with optimization

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 optimize it w.r.t. the primitive parameters

$$\Theta^{\text{ref}} = \operatorname*{argmin}_{\Theta} E\left(\Theta, I_p\right). \tag{5}$$

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

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

where $\Theta^{\text{pos}} = \{\boldsymbol{\theta}_k^{\text{pos}}\}_{k=1}^{n_{\text{prim}}}$ are the position parameters of the primitives, and $\Theta^{\text{size}} = \left\{\boldsymbol{\theta}_k^{\text{size}}\right\}_{k=1}^{n_{\text{prim}}}$ are the size parameters, and $\boldsymbol{\theta}_k = (\boldsymbol{\theta}_k^{\text{pos}}, \boldsymbol{\theta}_k^{\text{size}})$.

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 $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, defined as a sum of close- and far-range potentials

$$\varphi\left(\boldsymbol{r}_1, \boldsymbol{r}_2\right) = e^{-\frac{\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|^2}{R_{\text{c}}^2}} + \lambda_{\text{f}} e^{-\frac{\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|^2}{R_{\text{f}}^2}}, \tag{7}$$

parameters of which we choose experimentally $R_{\text{c}} = 1\,\text{px}$, $R_{\text{f}} = 32\,\text{px}$, $\lambda_{\text{f}} = 0.02$. The energy of interaction of the uniform positively charged area of the k^{th} primitive Ω_k and a grid of point charges $\boldsymbol{q} = \{q_i\}_{i=1}^{n_{\text{pix}}}$ at the pixel centers $\boldsymbol{r}_i$ is then defined by the following equation, that we integrate analytically for lines

$$E_k\left(\boldsymbol{q}\right) = \sum_{i=1}^{n_{\text{pix}}} q_i \iint\limits_{\Omega_k} \varphi\left(\boldsymbol{r}, \boldsymbol{r}_i\right) dr^2. \tag{8}$$

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

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

Energy terms. In our description of energy terms, we use $\odot$ to denote the componentwise product of vectors, and $\mathbf{1}$ denotes the vector of ones of an appropriate size.

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

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

The weighting $c_{k,i} \in \{0, 1\}$ enforces coverage of a continuous raster region following the form and orientation of the primitive. Specifically, we set $c_{k,i}$ to 1 inside the largest region aligned with the primitive with only shaded pixels of the raster. For example, for a line segment, this region is a rectangle centered at the center of the segment and aligned with the segment.

The second term is responsible for alignment of fixed size primitives

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

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 $\lambda_{\text{f}} = 0$:

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

where $\boldsymbol{l}_{k,i}$ is the direction of the primitive at its closest point to the i^{th} pixel, $\boldsymbol{m}_{k,i} = \sum_{j \neq k} \boldsymbol{l}_{j,i} q_{j,i}$ 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 charge interactions, we can use well-known approximations used in physics to solve many-body interactions (mean field theory). This translates into the observation that one can obtain an approximation of the solution by viewing interactions of each primitive with the rest as interactions with a static set of charges, i.e., viewing each energy term $E_k^{\text{pos}}, E_k^{\text{size}}, E_k^{\text{rdn}}$ as depending only on the parameters of the primitive k. This enables very efficient gradient computation for our functional, as one needs to differentiate each terms with respect to a small number of parameters only.

We optimize the functional (26) by Adam, using a gradient computed in the following way. 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 estimations from all patches with straightforward heuristic algorithms. For lines, we build a graph with the primitives from the whole image as nodes and link two nodes if the respective primitives are close and collinear enough, but not if they are almost parallel. After that, we replace each group of primitives, forming a connected component of the graph, with a single primitive which is a least-squares fit to the endpoints of the original primitives. 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 quadratic Bezier curves, for each pair of close primitives we estimate a replacement quadratic Bezier curve via least-squares and leave this curve instead of the former pair if the fit is close enough. We repeat this operation for the whole image until no more pairs allow a close fit.

4 Experimental evaluation

4.1 Datasets

To train our networks and evaluate them on a wide variety of images, we have developed four distinct datasets: (1) synthetic raster cleaning datasets; (2) real-world raster for cleaning fine-tuning

and evaluation; (3) real-world vector datasets containing floor-plans (PFP); (4) real-world vector dataset containing mechanical drawings: rendering of ABC models.

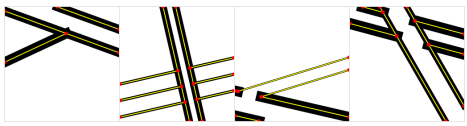


Fig. 2: Examples of vector synthetic local patches that we use for training our deep vectorization model.

Synthetic datasets. We create a synthetic dataset for the cleaning step. We take into account the restriction that the visual content of synthetic data is similar to the real-world images.

To create the synthetic line drawing, we use graphical primitives (straight lines, curved lines, and circular arcs) and objects (parallelepipeds, circles, and triangles). We use 40 realistic photographed and scanned paper backgrounds, randomly selected from images available online. Our synthetic dataset consists of 20000 images at a resolution of 512×512.

In addition for cleaning we create synthetic data 64×64, by mimicing real data patch distribution, to train vectorization network (see Figure 2).

Degraded line drawings(DLD). We collected 81 photographed or scanned floor plan images. The resolutions of the images, on average, is 1300×1000 pixels. We manually remove text and clean the images using the Photoshop software. We obtain their versions (a) without background, (b) the same having improved lines (where we have inpainted gaps and sharpened edges). We provide sample images from the created dataset demonstrating these versions in Figure 3.

PFP floor plan vector dataset. To obtain a vector dataset for training the model, we downloaded 1554 PDF documents containing real-world architectural floor plans from a commercial website [2]. We transformed them into a scalable vector-graphics format, perform extensive augmentations in terms of both configurations (rotations, translations, and random crops) and parameterizations (line widths) of primitives. We use 1364 floor plan images for training and leave 150 images for validation. As PFP dataset mostly contains configurations of straight lines of varying complexity. We use it to demonstrate the performance of our method with lines only.

ABC mechanical parts vector dataset. We use $\approx$ 10000 parametric CAD models from the ABC dataset [23]. They have been designed to model mechanical parts with sharp edges and well defined surfaces. To create a vector image, we use the boundary representation of the CAD model, perform an arbitrary 3D rotation, and render it with a parallel projection using the open-source software Open Cascade [1].

We split both vector datasets into 64×64 patches and create training targets required by the neural network, as described in Section 3.2.

4.2 Evaluation setup

For image cleaning, we use synthetic training data along with real images from the DLD dataset. We randomly split real cleaned images to train/val/test as 51/15/15 images correspondingly. We use the train and validation set during training and test set for evaluating models.

Due to the dominance of the number of lines on floorplan-based datasets we use lines on it and quadratic curves on ABC dataset to show our pipeline performance on both lines and curves.

On both PFP and ABC we randomly divided our datasets into train-val-test by original big images to prevent similar patch appearance in those sets. As test we used 40 images from PFP and 50 images in ABC, with resolutions of the images $\approx 2000 \times 3000$ pixels.

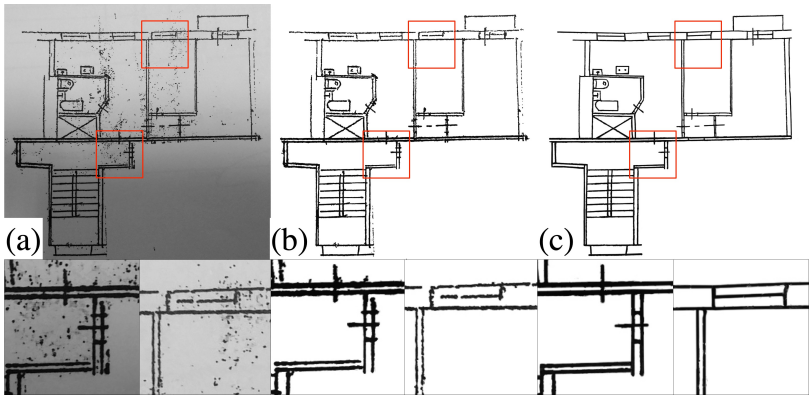


Fig. 3: Our real cleaning dataset: (a) displays an example of a original line drawing; (b) a cleaned version of the drawing (note in lines gaps that are subject to infilling), and (c) a target version with infilled lines.

4.3 Quality measures

Vectorization performance is naturally characterized using Intersection-over-Union (IoU) measure. It reflects deviations in two raster shapes R_1 and R_2 via $\mathrm{IoU}(R_1, R_2) = \frac{R_1 \cap R_2}{R_1 \cup R_2}$. IoU can be used with both raster and vector drawings (the latter requires computing $R_i = \mathrm{Render}(V_i)$ where Render is a rendering operator).

Almost all methods do not take into account the primitives width for constructing a vector representation. Thus we put the primitives width equal to the average width for fair method comparison by IoU. In case, when there isn't ground truth vector, we set the width as the sum of all nonzero pixels divided by the length of the predicted primitives.

IoU does not capture deviations in graphical primitives that have similar shapes but are slightly offset from each other, see Figure 4. We additionally compute the difference in skeleton structures of the two vector images X and Y using Hausdorff distance, defined by

$$d_{\mathrm{H}}(X, Y) = \max\left\{ \sup_{x \in X} \inf_{y \in Y} d(x, y),\ \sup_{y \in Y} \inf_{x \in X} d(x, y) \right\}, \tag{12}$$

and mean minimal deviation, defined by

$$d_{\mathrm{M}}(X, Y) = \left(\widetilde{d_{\mathrm{M}}}(X \to Y) + \widetilde{d_{\mathrm{M}}}(Y \to X) \right) \Big/ 2\,, \tag{13a}$$

$$\widetilde{d_{\mathrm{M}}}(X \to Y) = \int\limits_{x \in X} \inf_{y \in Y} d(x, y) dX \Big/ \int\limits_{x \in X} dX\,, \tag{13b}$$

where $d(x, y)$ denotes the Euclidean distance between a pair of points on skeletons. In practice, we densely sample the skeletons of the two vector images and approximate these metrics on a pair of point clouds.

Lastly, we quantify complexity of the vector drawing by the number of primitives constituting the vector image (denoted #P).

4.4 Comparative studies

In this section, we assess quantitatively and qualitatively the quality of the proposed vectorization approach alongside the existing approaches.

Methods. We compare against Fidelity vs Simplicity (FvS) [11], Complete Hand-Drawn Sketch Vectorization Framework (CHD) [9], and Vectorization of Line Drawings via PolyVector Fields (PVF) [3] methods on vectorization tasks, and against Mastering Sketching [36] on cleaning tasks.

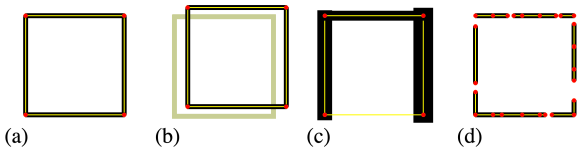


Fig. 4: Vectorization defects diagnosed by our quality measures: (a) ground-truth (desired) vector image, (b) skeleton structure deviations, captured by (12)–(13) (pale green primitives denote ground-truth), (c) shape deviations, captured by IoU, (d) overparameterization.

Table 1: Quantitative comparison on our datasets. In IOU metric on our model the left one with average width, and right one with predicted.

	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

Clean line drawings. The goal of this experiment is to characterize the maximum achievable geometric precision using a set of clean raster images with precisely known vector ground-truth (renderings of precisely known vector drawings).

We display values of numerical performance measures across methods on PFP and ABC datasets in Table 1.

There is always a trade-off between the number of primitives and the precision. Thus the comparison of some methods with different number of output primitives is not to fair. Our method outperforms all quality metrics on PFP dataset, but loses in primitive count to FVS. On ABC dataset PVF method has IoU equal to 89. In our case, we obtained IoU equal to 91. However, after post processing IoU decreased since it is very sensitive to small movements of the objects. Also PVF resulted in 7818 primitives which is much bigger then our 97.

We provide side-by-side images of vectorization results using various methods, along with close-ups, highlighting difference in vectorization across methods on PFP in Figure 5 .

Degraded line drawings. We would like to establish the effect of noise on the vectorization performance. For this purpose we characterize the typical anticipated level of geometric deviations using a representative set of real-world raster images (without vector ground-truth). We use the real images with low PSNR values in the dataset, according to their cleaned versions.

We display values of numerical performance measures across methods and datasets in the right part of the Table 1. Only CHD can work with noise images; we compare with this method.

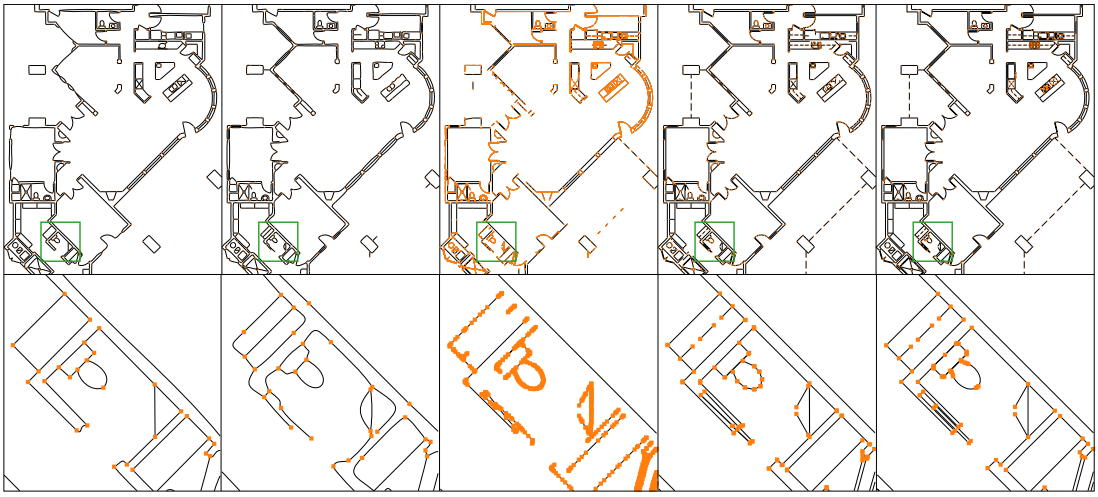
We provide side-by-side images of vectorization results using various methods, along with close-ups, highlighting differences in vectorizations across methods in Figure 6.

One may conclude that cleaning prior to the vectorization step is necessary, when dealing with real-world drawings, to ensure geometric fidelity of the predictions.

Cleaning results. We separately evaluate our cleaning procedure, comparing it with a recent cleaning method MS [36]. We display values of numerical performance measures in Table 2 and an example of cleaning results along with close-up fragments in Figure 7. We note that our cleaning procedure keeps straight and repeated lines commonly found in technical drawing while MS outputs wavy strokes and tends to join repeated straights, thus harming the structure of the drawing.

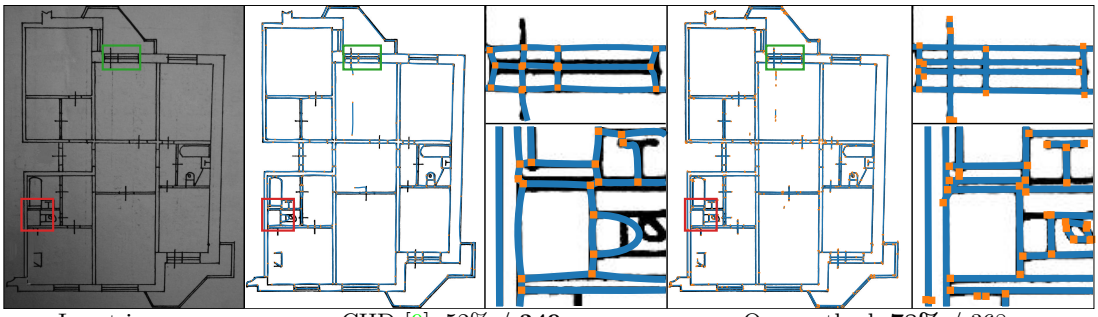
4.5 Ablation studies

In this section, we evaluate our proposed vectorization approach to understand the impact of each of our components. We perform forward feature selection technique on our pipeline modules. To this



FvS [11]	CHD [9]	PVF [3]	Our method	Ground truth,
29% / 415px	21% / 215px	64% / 140px	**89%** / **28px**	#P 1634
4.2px / **615**	1.9px / 1192	0.9px / 35k	**0.2px** / 1286	

Fig. 5: 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.



Input image | CHD [9], 52% / **349** | Our method, **78%** / 368

Fig. 6: Qualitative comparison on real noisy image, and values of metrics IoU / #P with best in bold. Primitives are shown in blue with endpoints in orange on top of the cleaned raster image.

Table 2: Comparison of our cleaning method with Mastering Sketching (MS). Comparison is performed on noisy images.

	IoU,%	PSNR
MS [36]	49	15.7
Our	**92**	**25.5**

end, we compare vectorization 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. We show Ablation study results on the ABC test set in Table 3.

As you can see adding Refinement improve all metrics. So always using this module is justified. On the other hand adding postprocess decrease other metrics due to the trade-off between number of primitives and precision.

5 Conclusion

In this paper, we have described a system for vectorization of technical line drawings. Two critical steps in our pipeline (initial cleaning, and the initial placements of primitives) are using neural net-

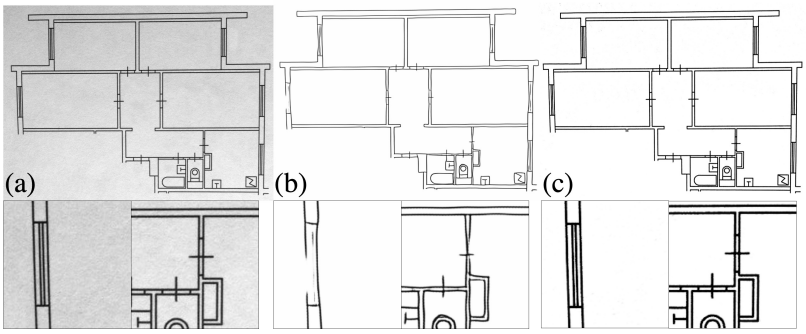


Fig. 7: Cleaning results: (a) input photo of a floorplan, (b) a cleaning result by MS [36]; (c) a cleaned version produced by our model. Note the tendency of MS to unite closely neighbouring straights in closeups.

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

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

works trained on a combination of synthetic and real data. A neural network can place the primitives approximately in the right location most of the time. However, its geometric accuracy is generally inadequate. Thus we add an optimization stage to the pipeline, adjusting primitive parameters to improve the fit. The evaluation shows that our algorithm, in general, performs significantly better compared to a number of recent vectorization algorithms.

Our algorithm can be easily extended in a number of ways. For example, integration with OCR systems will allow to separate and enhance the text annotations. One can also hope that training the method end-to-end, including the smooth part of the optimization stage, will improve the overall system performance.

6 Acknowledgement

The authors thank CDISE MSc. students 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. The authors acknowledge the usage of the Skoltech CDISE HPC cluster Zhores for obtaining the results presented in this paper. The work was supported by The Ministry of Education and Science of Russian Federation, grant No. 14.615.21.0004, grant code: RFMEFI61518X0004.

References

1. Open CASCADE Technology OCCT. https://www.opencascade.com/, accessed: 2020-03-05 7
2. PrecisionFloorplan. http://precisionfloorplan.com, accessed: 2020-03-05 7
3. Bessmeltsev, M., Solomon, J.: Vectorization of line drawings via polyvector fields. ACM Transactions on Graphics (TOG) **38**(1), 9 (2019) 2, 8, 9, 10, 19, 20
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) 3
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) 2
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) 2
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) 3
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) 3
9. Donati, L., Cesano, S., Prati, A.: A complete hand-drawn sketch vectorization framework. Multimedia Tools and Applications **78**(14), 19083–19113 (2019) 3, 8, 9, 10, 19, 20, 21
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) 3
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) 2, 3, 8, 9, 10, 19, 20
12. Gao, J., Tang, C., Ganapathi-Subramanian, V., Huang, J., Su, H., Guibas, L.J.: Deepspline: Data-driven reconstruction of parametric curves and surfaces. arXiv preprint arXiv:1901.03781 (2019) 3
13. Ha, D., Eck, D.: A neural representation of sketch drawings (2018) 3
14. 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) 4
15. 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) 14
16. 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) 3
17. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. IEEE Transactions on Pattern Analysis & Machine Intelligence (6), 890–904 (2006) 2
18. Kaiyrbekov, K., Sezgin, M.: Stroke-based sketched symbol reconstruction and segmentation. arXiv preprint arXiv:1901.03427 (2019) 3
19. Kansal, R., Kumar, S.: A vectorization framework for constant and linear gradient filled regions. The Visual Computer **31**(5), 717–732 (2015) 3
20. Kanungo, T., Haralick, R.M., Baird, H.S., Stuezle, 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) 3
21. 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) 2
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 14
23. 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) 7
24. Li, C., Liu, X., Wong, T.T.: Deep extraction of manga structural lines. ACM Transactions on Graphics (TOG) **36**(4), 117 (2017) 3, 4
25. 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
26. 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) 3
27. Máttyus, 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) 3
28. Munusamy Kabilan, V., Morris, B., Nguyen, A.: Vectordefense: Vectorization as a defense to adversarial examples. arXiv preprint arXiv:1804.08529 (2018) 2
29. 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) 2

30. 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 14
31. 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) 4, 14
32. 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) 3
33. Sasaki, K., Iizuka, S., Simo-Serra, E., Ishikawa, H.: Learning to restore deteriorated line drawing. The Visual Computer **34**(6-8), 1077–1085 (2018) 3
34. Selinger, P.: Potrace: a polygon-based tracing algorithm. Potrace (online), http://potrace.sourceforge.net/potrace.pdf (2009-07-01) (2003) 2
35. 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) 3
36. Simo-Serra, E., Iizuka, S., Ishikawa, H.: Mastering sketching: adversarial augmentation for structured prediction. ACM Transactions on Graphics (TOG) **37**(1), 11 (2018) 3, 8, 9, 10, 11
37. 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 14
38. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017) 4, 14
39. 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) 3
40. Zheng, N., Jiang, Y., Huang, D.: Strokenet: Aneural painting environment 3
41. 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) 3

A Introduction

We provide additional qualitative results and details about our pipeline components. In Section B, we describe the architectures of our neural networks. In Section C we detail on training of the models. We show more qualitative comparisons for ablation study in Section D and more comparisons of our method to others in Section E. In Section F we detail on our refinement algorithm.

B Neural Networks architectures

For image cleaning, we used the U-net (see [31]) architecture. It consists of blocks with convolutional layers, with batch normalization, and ReLU activation. There are seven blocks in the encoder, and seven blocks in the decoder. We used MaxPool, downsampling the nearest neighbors upsampling, and skip connections, as in original U-net paper [31].

For the vectorization neural network, we used ResNet18 blocks [15] for feature extraction and Transformer model [38] for primitive parameters estimation. The pipeline of this model is shown in Figure 8.

For both lines and second-degree Bezier curves, we use the same neural networks with the same hyper-parameters. The only difference is in primitives parameters m, which is equal to 6 for lines and 8 for curves. We use one ResNet18 block and eight transformer layers. In the ResNet, we use 64 channels for each convolution, and in the transformer part, we use 4 heads of multi-head attention and 512 neurons in the fully-connected layer.

C Training details

We used Pytorch 1.2 for GPU computations and model training [30] and GNU Parallel to speed up the metric calculations [37] 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 [22] for optimization with a scheduler with the same hyperparameters as in the original Transformer paper [38]. 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.

In Figure 9 and Figure 10 we provide metrics on train and validation sets for patches with size 64×64 from ABC and PFP datasets correspondingly.

D 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 11 and Figure 12.

E Additional results

In this section, we show more qualitative comparisons on test set for both PFP in Figure 13 and ABC in Figure 14 datasets and on real data in Figure 15.

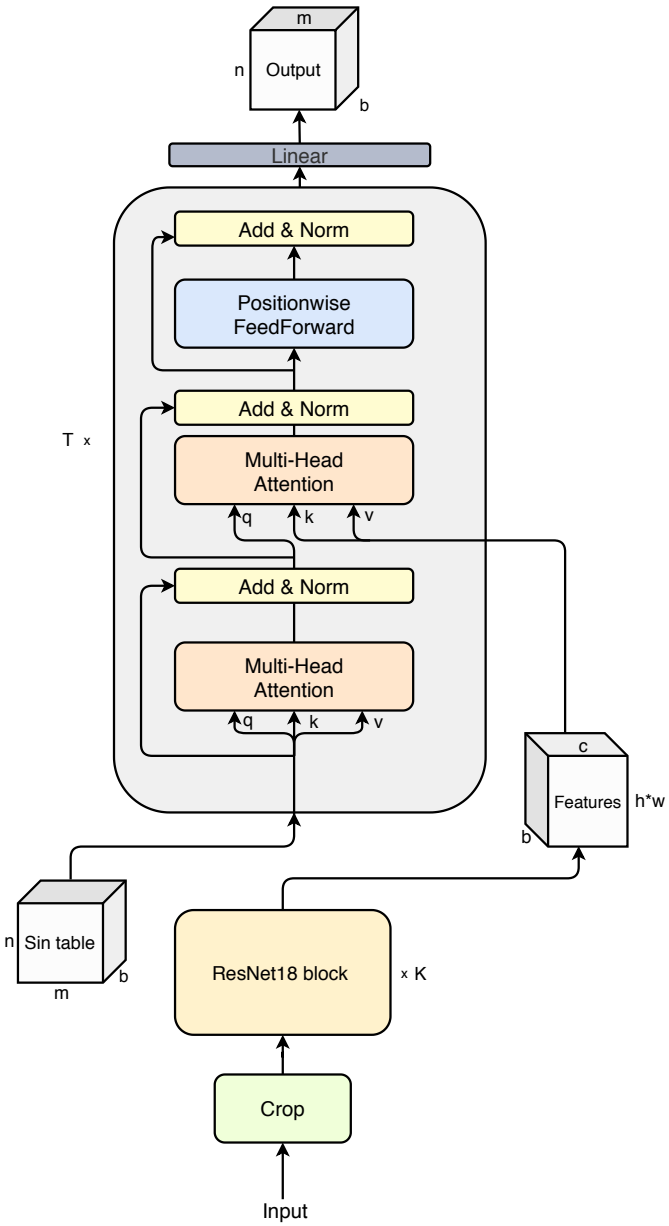


Fig. 8: Neural network architecture for estimation of primitives. The input for the model is a batch of greyscale images. Here n is the number of primitives in a patch, b is a batch size, h and w are height and width of a ResNet block output feature map ,and m is primitive parameters. K and T are ResNet and Transformer block repetition numbers.

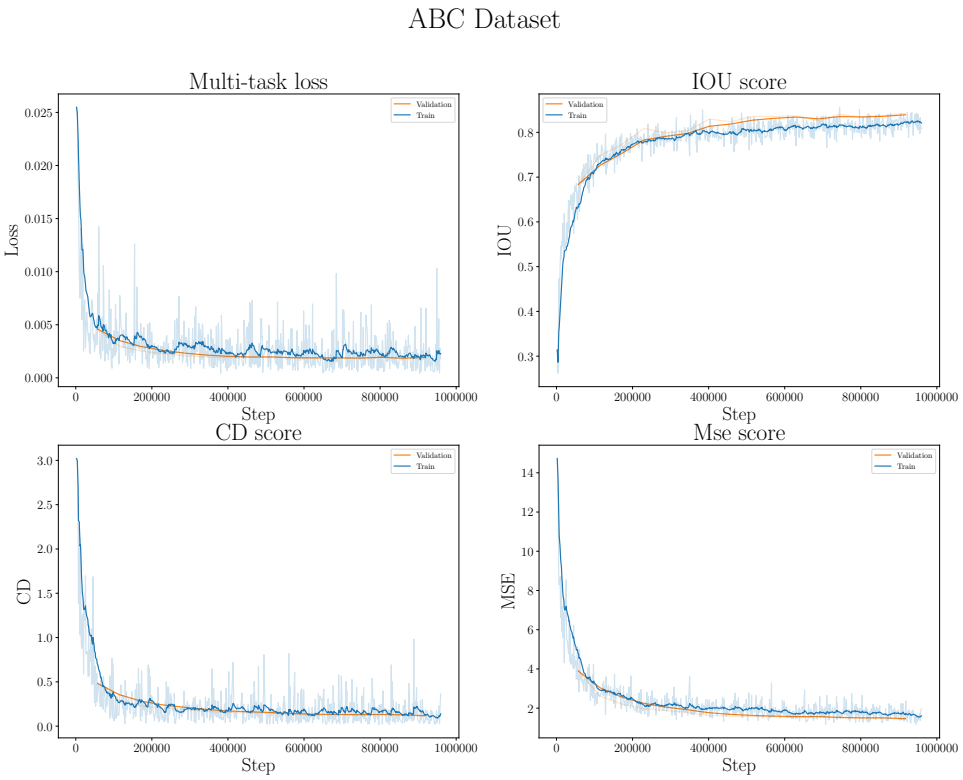


Fig. 9: Metrics and loss function for train and validation on ABC dataset. One step of X-axis represents calculations on a single batch.

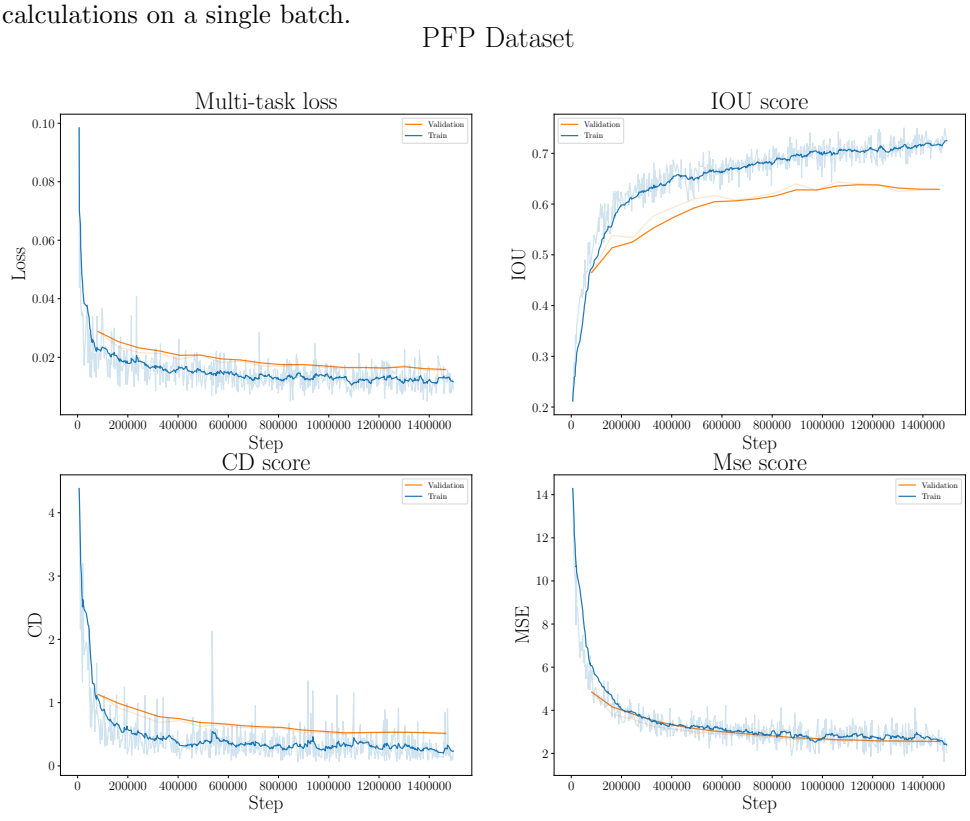
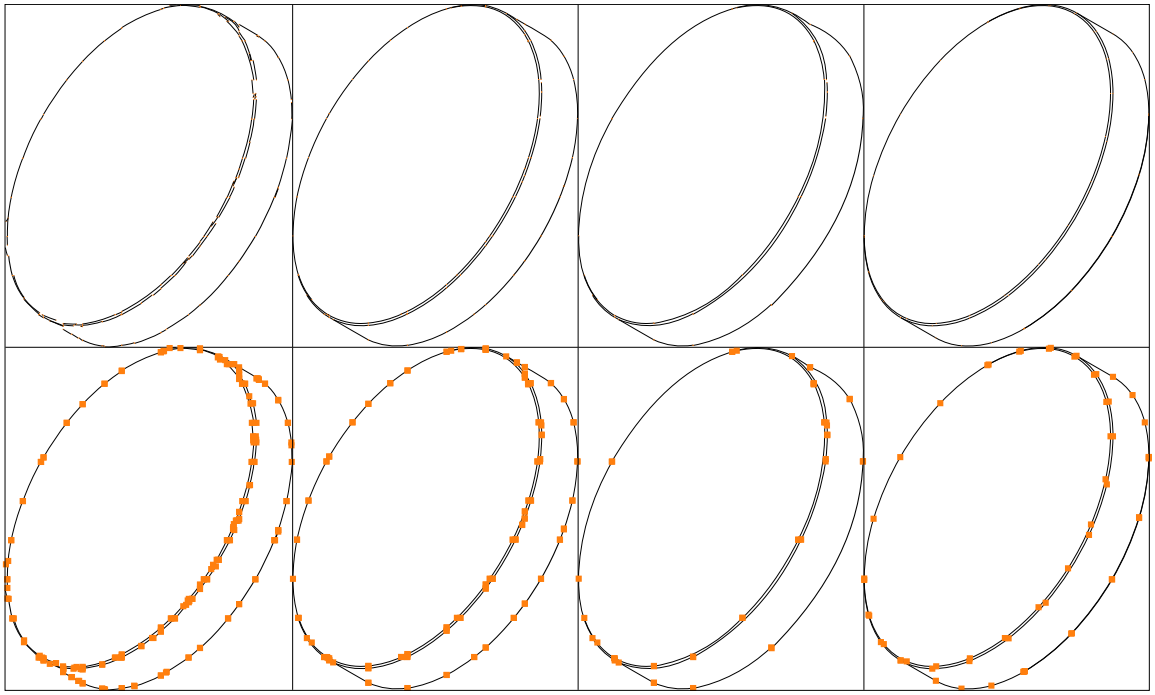


Fig. 10: Metrics and loss function for train and validation on PFP dataset. One step of the X-axis represents computations on a single batch.

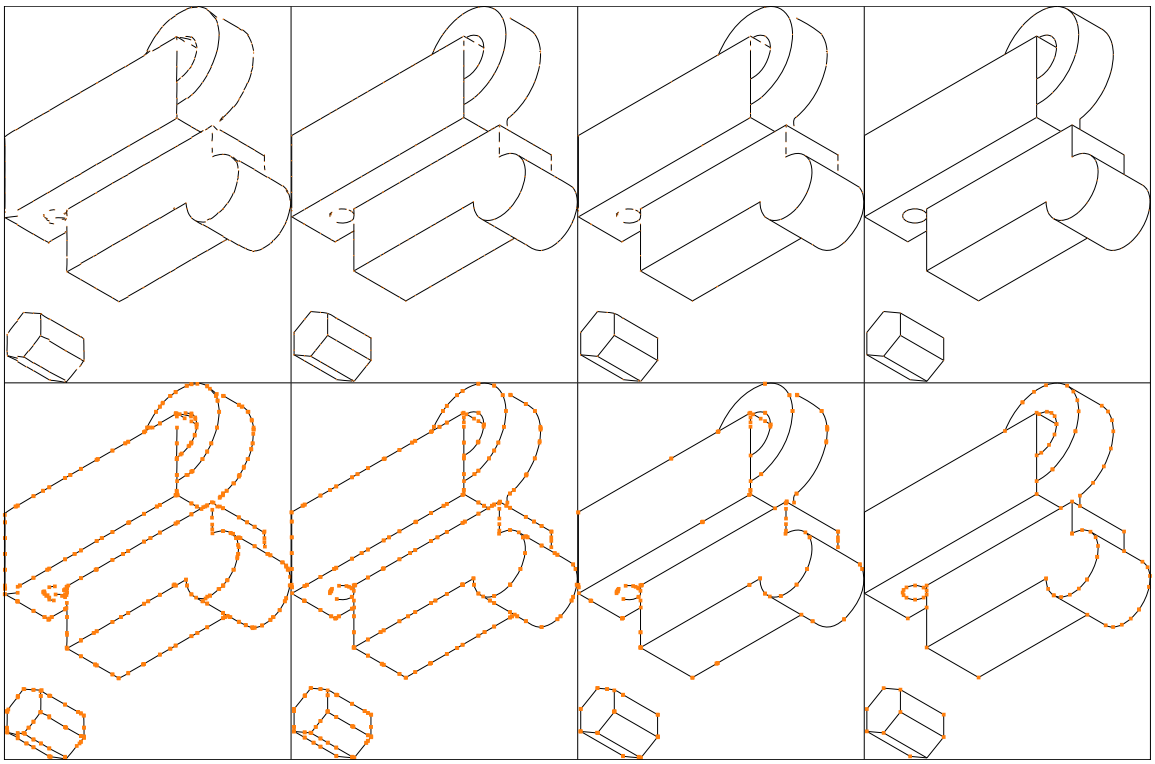


NN
54% / 11px
0.9px / 109

NN + Refinement
93% / **5px**
0.2px / 69

Full
67% / **5px**
0.5px / **25**

Ground truth,
#P 57



NN
51% / 24px
1.0px / 265

NN + Refinement
91% / **19px**
0.2px / 232

Full
85% / **19px**
0.3px / **84**

Ground truth,
#P 89

Fig. 11: 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.

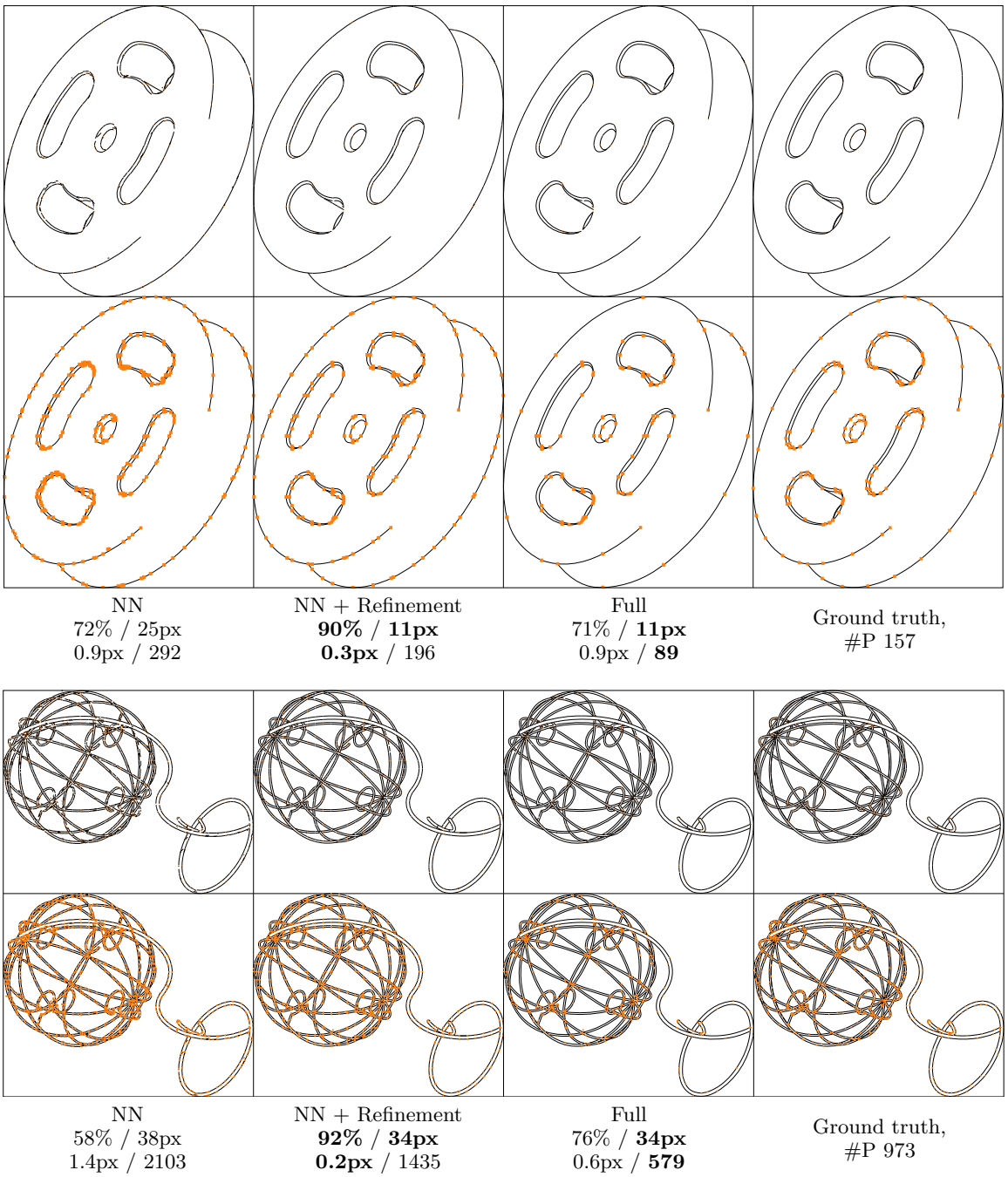


Fig. 12: Qualitative comparison on ABC images, and values of metrics IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.

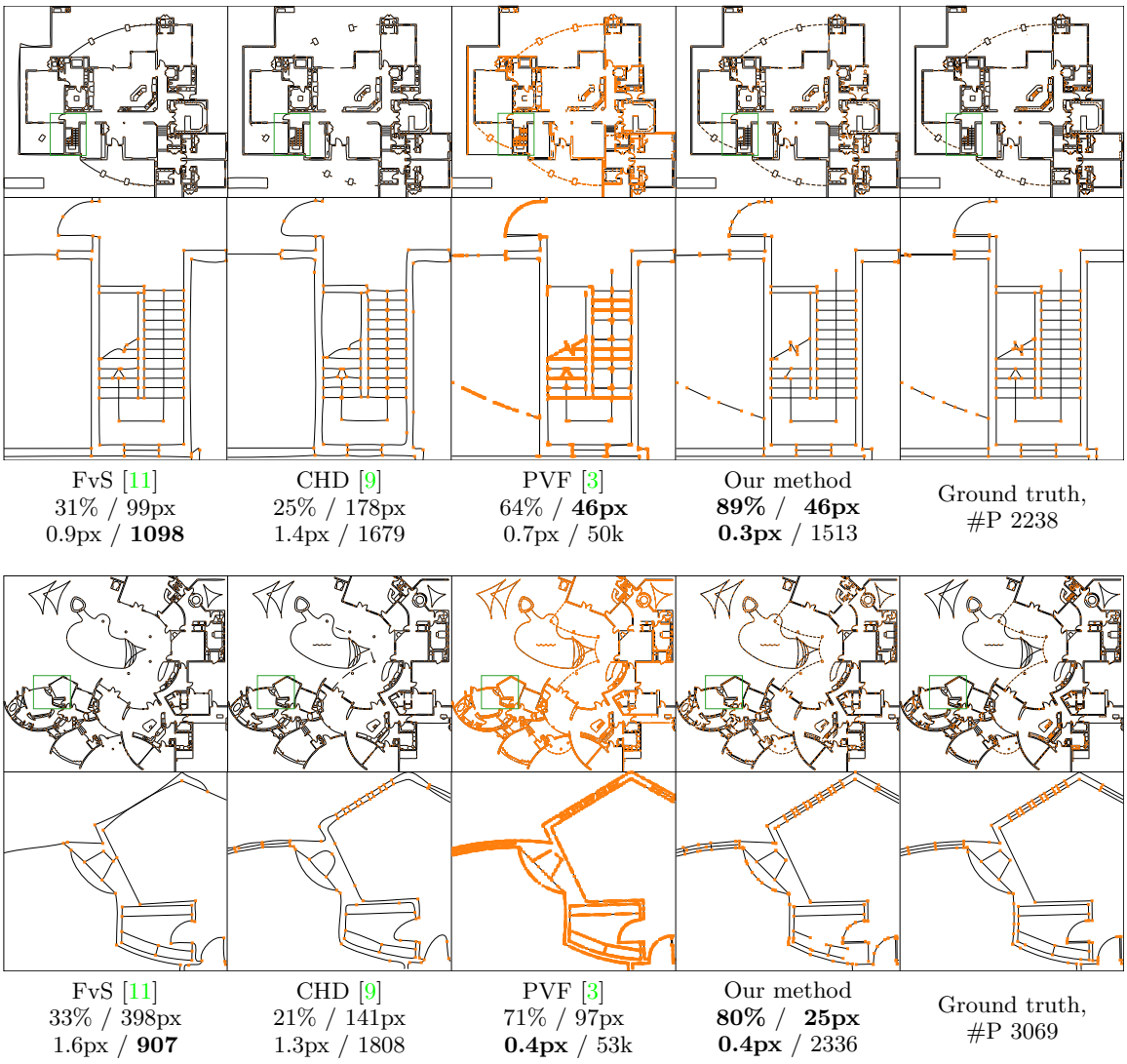
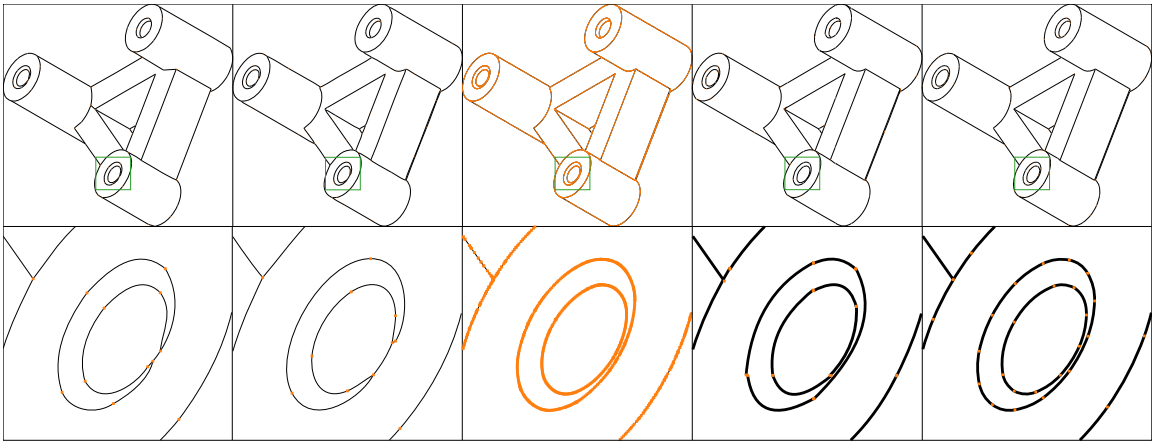
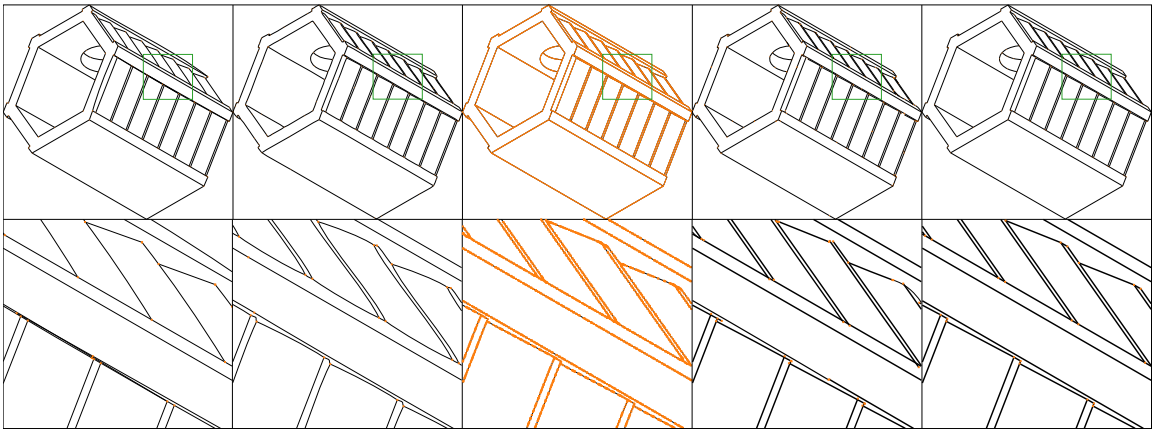


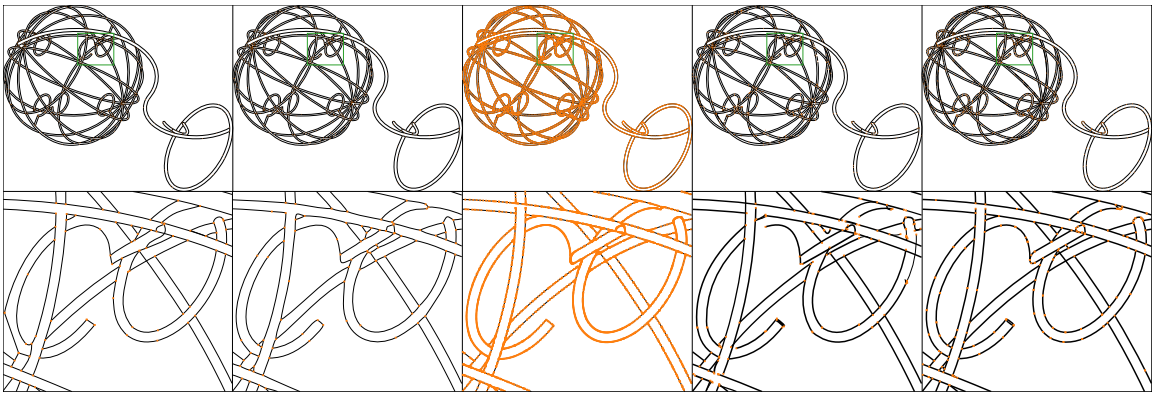
Fig. 13: Qualitative comparison on PFP images, and values of metrics IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.



FvS [11] 67% / 32px 1.1px / **79** | CHD [9] 67% / 7px 1.0px / 108 | PVF [3] **95%** / **4px** **0.2px** / 9.5k | Our method 86% / 5px 0.4px / 139 | Ground truth, #P 139

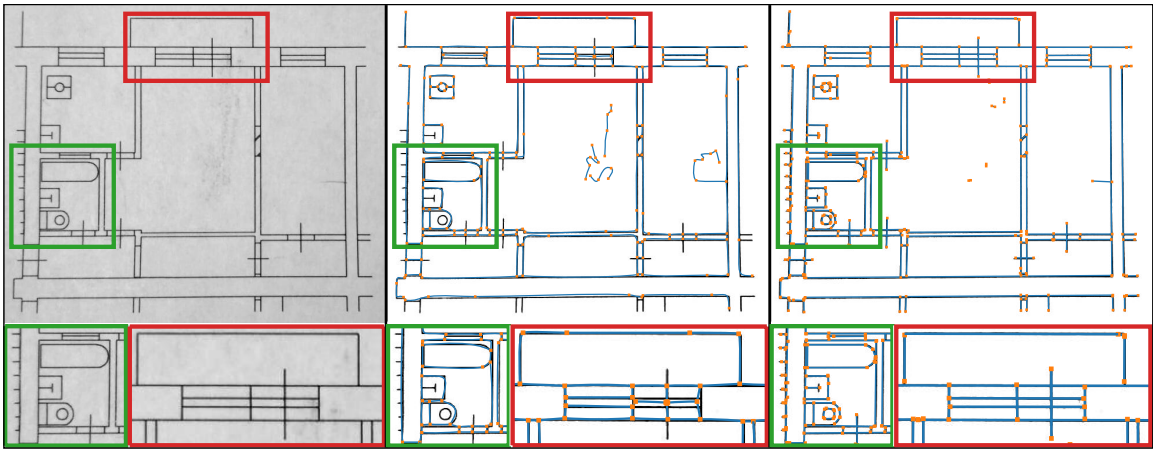


FvS [11] 86% / 23px 1.3px / **88** | CHD [9] 68% / 5px 1.0px / 211 | PVF [3] **96%** / **2px** **0.2px** / 20k | Our method 82% / 14px **0.2px** / 147 | Ground truth, #P 122

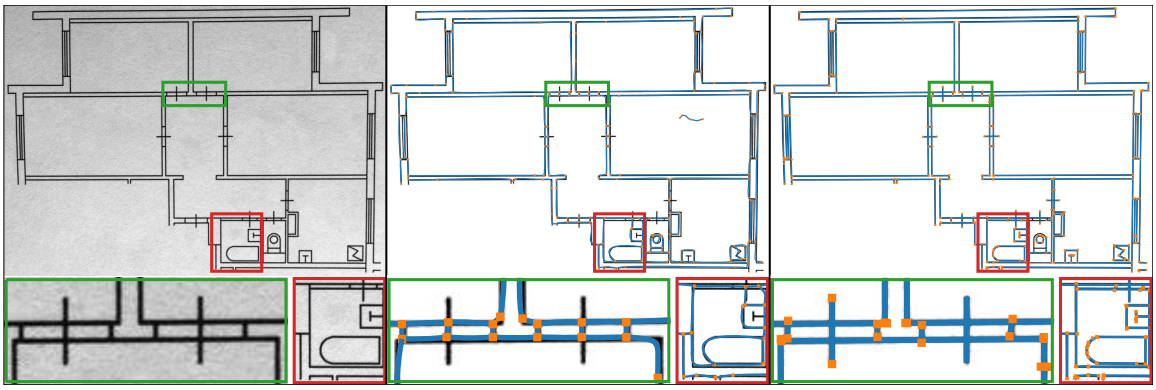


FvS [11] 74% / 27px 0.7px / **433** | CHD [9] 64% / 17px 0.9px / 994 | PVF [3] **91%** / **6px** **0.3px** / 43k | Our method 76% / 34px 0.6px / 579 | Ground truth, #P 973

Fig. 14: 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.



Input image | CHD [9], 38% / 230 | Our method, **81%** / **187**



Input image | CHD [9], 44% / 226 | Our method, **84%** / **174**

Fig. 15: 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.

F Details on refinement algorithm

F.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 and 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 minimima.

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

$$q_1 q_2 \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right), \tag{14}$$

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.

$$E = \sum_{k_{\text{prim}}, i_{\text{pix}}} E^{\text{prim,pix}}_{k_{\text{prim}}, i_{\text{pix}}} + \sum_{k_{\text{prim}} < j_{\text{prim}}} E^{\text{prim,prim}}_{k_{\text{prim}}, j_{\text{prim}}} + \sum_{k_{\text{prim}}} E^{\text{prim}}_{k_{\text{prim}}}. \tag{15}$$

Three parts of the energy have the following form:

$$E^{\text{prim,pix}}_{k_{\text{prim}}, i_{\text{pix}}} = -\hat{q}_{i_{\text{pix}}} \iint\limits_{\Omega_{k_{\text{prim}}}} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\|\right) dr^2_{,} \tag{16}$$

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

$$E^{\text{prim,prim}}_{k_{\text{prim}}, j_{\text{prim}}} = \iint\limits_{\Omega_{k_{\text{prim}}}} \iint\limits_{\Omega_{j_{\text{prim}}}} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr^2_1 dr^2_2; \tag{17}$$

and

$$E^{\text{prim}}_{k_{\text{prim}}} = \frac{1}{2} E^{\text{prim,prim}}_{k_{\text{prim}}, k_{\text{prim}}} = \frac{1}{2} \iint\limits_{\Omega_{k_{\text{prim}}}} \iint\limits_{\Omega_{k_{\text{prim}}}} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr^2_1 dr^2_2. \tag{18}$$

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.

F.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:

$$\theta_{k,\alpha} \leftarrow \theta_{k,\alpha} - \lambda \frac{\partial E}{\partial \theta_{k,\alpha}}. \tag{19}$$

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.

$$\begin{gathered}\frac{\partial}{\partial \theta_{k,\alpha}^{\{\text{pos,size}\}}} \sum_{k_{\text{prim}}, i_{\text{pix}}} E_{k_{\text{prim}}, i_{\text{pix}}}^{\text{prim,pix}} = \partial \sum_{i_{\text{pix}}} E_{k, i_{\text{pix}}}^{\text{prim,pix}} = \\ -\sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \partial \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\|\right) dr^2,\end{gathered} \tag{20}$$

$$\begin{gathered}\frac{\partial}{\partial \theta_{k,\alpha}^{\{\text{pos,size}\}}} \sum_{k_{\text{prim}} < j_{\text{prim}}} E_{k_{\text{prim}}, j_{\text{prim}}}^{\text{prim,prim}} = \partial \sum_{j_{\text{prim}} \neq k} E_{k, j_{\text{prim}}}^{\text{prim,prim}} = \\ \partial \iint\limits_{\Omega_k} \sum_{j_{\text{prim}} \neq k} \iint\limits_{\Omega_{j_{\text{prim}}}} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr_1^2 dr_2^2,\end{gathered} \tag{21}$$

$$\frac{\partial}{\partial \theta_{k,\alpha}^{\text{pos}}} \sum_{k_{\text{prim}}} E_{k_{\text{prim}}}^{\text{prim}} = 0, \tag{22}$$

$$\begin{gathered}\frac{\partial}{\partial \theta_{k,\alpha}^{\text{size}}} \sum_{k_{\text{prim}}} E_{k_{\text{prim}}}^{\text{prim}} = \partial E_k^{\text{prim}} = \\ \frac{1}{2} \partial \iint\limits_{\Omega_k} \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr_1^2 dr_2^2 + \frac{1}{2} \partial \iint\limits_{\Omega_k} \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr_1^2 dr_2^2 = \\ \partial \iint\limits_{\Omega_k} \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr_1^2 dr_2^2,\end{gathered} \tag{23}$$

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

$$\begin{gathered}\frac{\partial E}{\partial \theta_{k,\alpha}^{\text{pos}}} = \partial \sum_{i_{\text{pix}}} E_{k, i_{\text{pix}}}^{\text{prim,pix}} + \partial \sum_{j_{\text{prim}} \neq k} E_{k, j_{\text{prim}}}^{\text{prim,prim}} = \\ \partial \iint\limits_{\Omega_k} \left[\sum_{j_{\text{prim}} \neq k} \iint\limits_{\Omega_{j_{\text{prim}}}} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_1\|\right) dr_1^2 - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\|\right) \right] dr^2\end{gathered} \tag{24}$$

For size parameters, we obtain the following expression

$$\begin{gathered}\frac{\partial E}{\partial \theta_{k,\alpha}^{\text{size}}} = \partial \sum_{i_{\text{pix}}} E_{k, i_{\text{pix}}}^{\text{prim,pix}} + \partial \sum_{j_{\text{prim}}} E_{k, j_{\text{prim}}}^{\text{prim,prim}} = \\ \partial \iint\limits_{\Omega_k} \left[\sum_{j_{\text{prim}}} \iint\limits_{\Omega_{j_{\text{prim}}}} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_1\|\right) dr_1^2 - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\|\right) \right] dr^2,\end{gathered} \tag{25}$$

where j_{prim} ranges over all primitives including k

We can interpret these derivatives as derivatives of a different function

$$E^* = \sum_k E_k^{\text{pos}} + E_k^{\text{size}}. \tag{26}$$

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.

$$E_k\left(q\right) = \iint\limits_{S} q\left(\boldsymbol{r}_1\right) \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr_2^2 dr_1^2, \tag{27}$$

$$E_k^{\text{pos}} = E_k\left(q_k^{\text{pos}}\right)\big|_{\boldsymbol{\theta}_k^{\text{size}}=\text{const}}, \quad E_k^{\text{size}} = E_k\left(q_k^{\text{size}}\right)\big|_{\boldsymbol{\theta}_k^{\text{pos}}=\text{const}}, \tag{28}$$

$$q_k^{\text{pos}}\left(\boldsymbol{r}\right) = \sum_{j_{\text{prim}} \neq k} \mathbb{1}\left[\boldsymbol{r} \in \Omega_{j_{\text{prim}}}\right] - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \delta\left(\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\right), \tag{29}$$

$$q_k^{\text{size}}\left(\boldsymbol{r}\right) = \sum_{j_{\text{prim}}} \mathbb{1}\left[\boldsymbol{r} \in \Omega_{j_{\text{prim}}}\right] - \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \delta\left(\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\right), \tag{30}$$

where $\mathbb{1}\left[\cdot\right]$ is equal to one if the condition it depends on is satisfied, and zero otherwise, and δ is the delta-function.

Expressions (26)-(30) 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.

F.3 Discretization and functional modifications

Discretization. While for simple primitives the integrals in (26)-(30) can be computed explicitly, we simplify the problem by using discrete charges instead of continuous distributions.

The expression (26) becomes equation (8) from the submission.

$$\iint\limits_{S} q\left(\boldsymbol{r}_1\right) \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|\right) dr_2^2 dr_1^2 \longrightarrow \sum_{i_{\text{pix}}} q_{i_{\text{pix}}} \iint\limits_{\Omega_k} \varphi\left(\|\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\|\right) dr^2 \tag{31}$$

Expressions (29) and (30) become

$$\iint\limits_{S} \mathbb{1}\left[\boldsymbol{r} \in \Omega_k\right] f\left(\boldsymbol{r}\right) dr^2 \longrightarrow \sum_{i_{\text{pix}}} q_{k,i_{\text{pix}}} f\left(\boldsymbol{r}_{i_{\text{pix}}}\right), \tag{32}$$

$$\iint\limits_{S} \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} \delta\left(\boldsymbol{r} - \boldsymbol{r}_{i_{\text{pix}}}\right) f\left(\boldsymbol{r}\right) dr^2 \longrightarrow \sum_{i_{\text{pix}}} \hat{q}_{i_{\text{pix}}} f\left(\boldsymbol{r}_{i_{\text{pix}}}\right), \tag{33}$$

$$q_k^{\text{pos}}\left(\boldsymbol{r}\right) \longrightarrow q_{k,i_{\text{pix}}}^{\text{pos}} = \sum_{j_{\text{prim}} \neq k} q_{j_{\text{prim}},i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \tag{34}$$

$$q_k^{\text{size}}\left(\boldsymbol{r}\right) \longrightarrow q_{k,i_{\text{pix}}}^{\text{size}} = \sum_{j_{\text{prim}}} q_{j_{\text{prim}},i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \tag{35}$$

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.

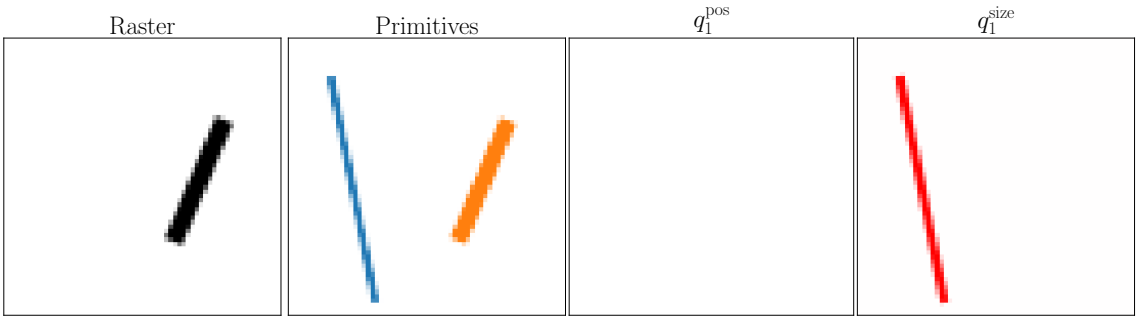


Fig. 16: Raster, primitives and charge grids of the first primitive. First primitive in blue, second in orange. In charge grids red represents excess charge.

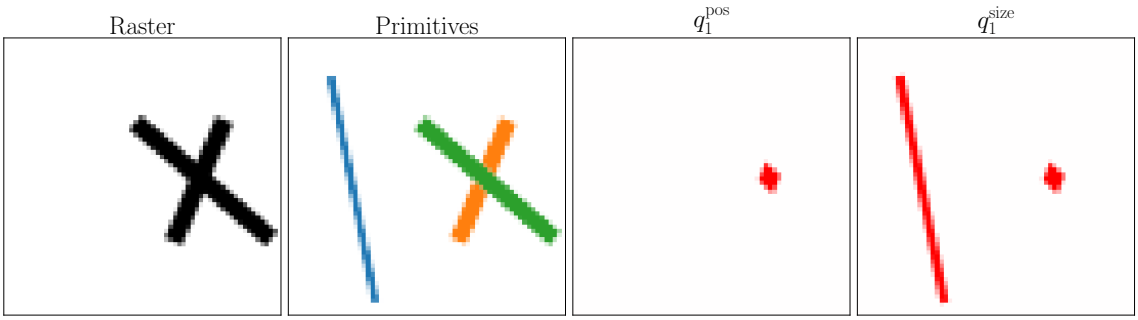


Fig. 17: Overlap affection on other primitives. First primitive in blue, second in orange. In charge grids red represents excess charge.

Charge saturation. The charge distributions $\boldsymbol{q}_k^{\text{pos}} = \left\{q_{k,i_{\text{pix}}}^{\text{pos}}\right\}_{i_{\text{pix}}}$, $\boldsymbol{q}_k^{\text{size}} = \left\{q_{k,i_{\text{pix}}}^{\text{size}}\right\}_{i_{\text{pix}}}$ are excess or insufficient charges that need to be compensated by 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 16, 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 17, 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:

$$q_{k,i_{\text{pix}}}^{\text{pos}} = q_{-k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \tag{36}$$

$$q_{k,i_{\text{pix}}}^{\text{size}} = q_{i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}, \tag{37}$$

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 (34), (35), modified charge distributions (36), (37) 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 18) 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 19) the energy is decreasing if the primitive moves to the

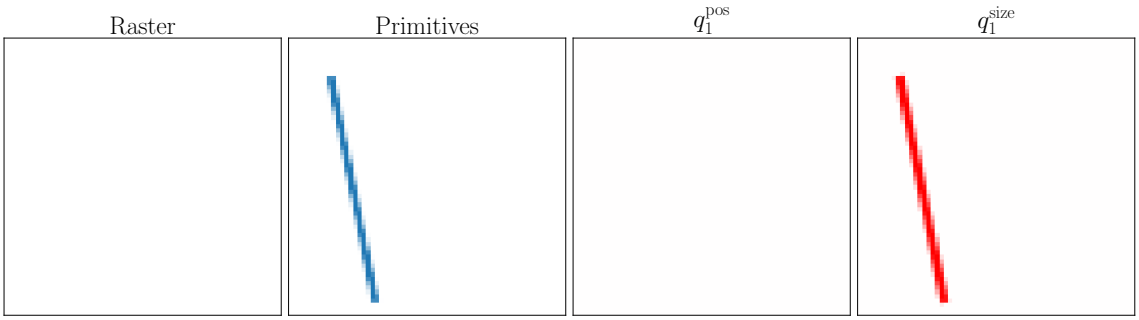


Fig. 18: Single primitive collapse. In charge grids red represents excess charge.

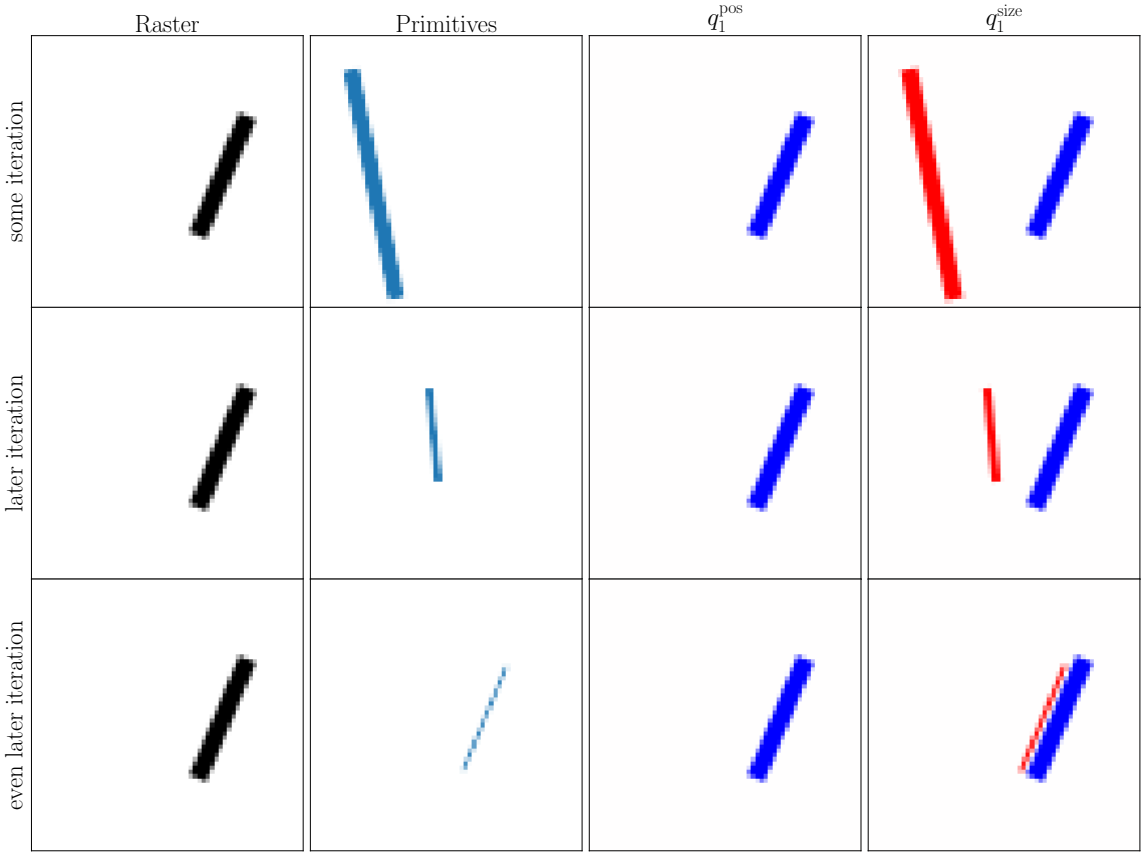


Fig. 19: Primitive interaction with far raster filled part. In charge grids red represents excess charge and blue represents uncovered raster.

raster due to the second term in (36). If the primitive shrinks, the first term in (37) 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.

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 20) 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 F.3.

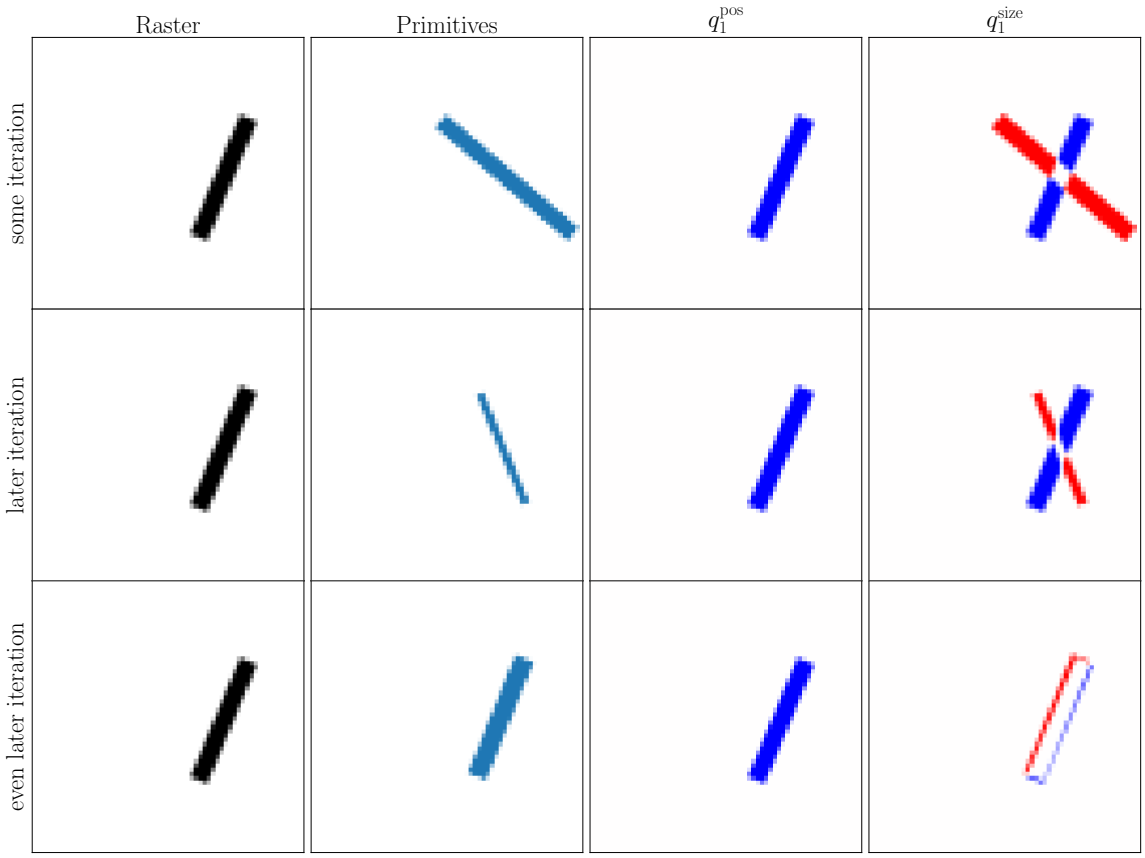


Fig. 20: Primitive interaction with close raster filled part. In charge grids red represents excess charge and blue represents uncovered raster.

Enabling primitive intersections. Consider the case in Figure 21. 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 $\boldsymbol{q}_k^{\text{pos}}$ as follows:

$$q_{k,i_{\text{pix}}}^{\text{pos}} = q_{i_{\text{pix}}} - q_{k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}}. \tag{38}$$

With this definition, the primitive interacts with pixels covered by other primitives, which allows it to expand across already filled areas, as in Figure 22

Penalty for overlapping collinear primitives $\boldsymbol{E_k^{\text{rdn}}}$. By itself, (38) allows not just transversal intersections, but also aligned primitives covering the same area (Figure 23). 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.

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

$$E_k^{\text{rdn}} = E_k\left(\boldsymbol{q}_k^{\text{rdn}}\right)\Big|_{\text{close } \varphi(r),\boldsymbol{\theta}_k^{\text{pos}}=\text{const}}, \tag{39}$$

$$q_{k,i_{\text{pix}}}^{\text{rdn}} = q_{j,i_{\text{pix}}} \exp\left(-\frac{\left(|\boldsymbol{l}_k \cdot \boldsymbol{l}_j| - 1\right)^2}{\left(|\cos\alpha_{\text{col}}| - 1\right)^2}\right), \tag{40}$$

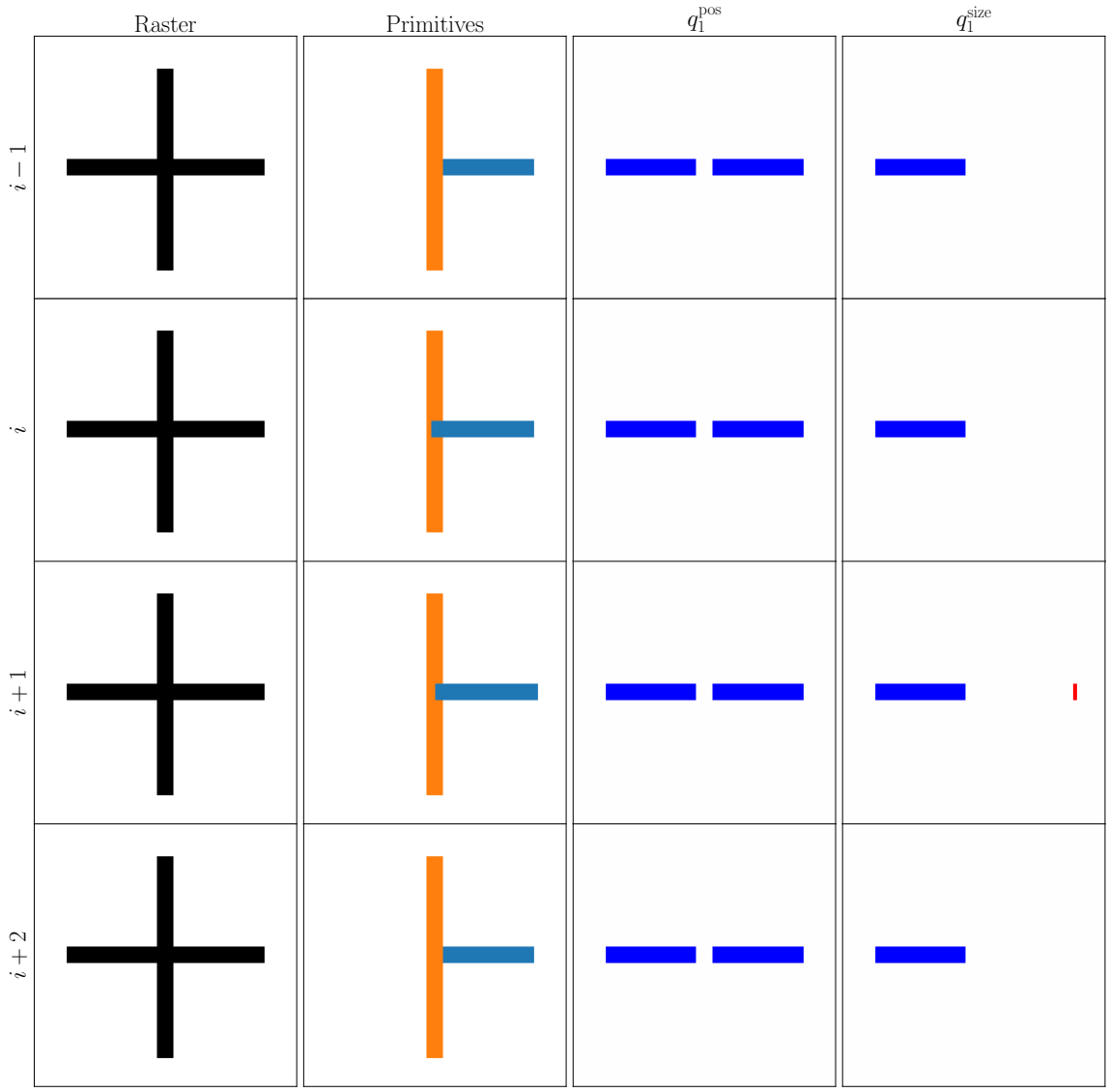


Fig. 21: Primitives interaction with disable primitive intersections. In charge grids red represents excess charge and blue represents uncovered raster.

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

For many segments, we define the charges as

$$q^{\text{rdn}}_{k,i_{\text{pix}}} = \|\boldsymbol{m}_{k,i_{\text{pix}}}\| \exp\left(-\frac{\left(|\boldsymbol{l}_k \cdot \boldsymbol{m}_{k,i_{\text{pix}}}| - 1\right)^2}{\left(|\cos\alpha_{\text{col}}| - 1\right)^2}\right), \tag{41}$$

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

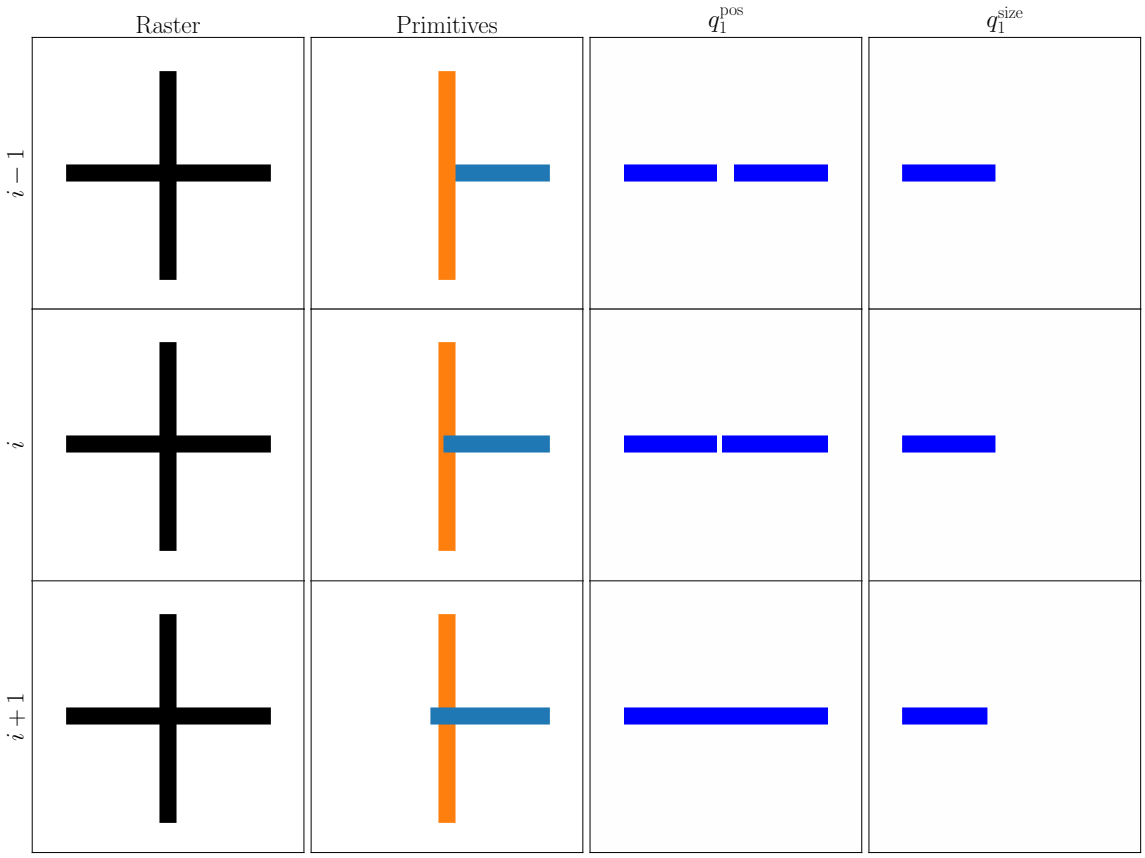


Fig. 22: Primitives interaction with enabled primitive intersections. In charge grids red represents excess charge and blue represents uncovered raster.

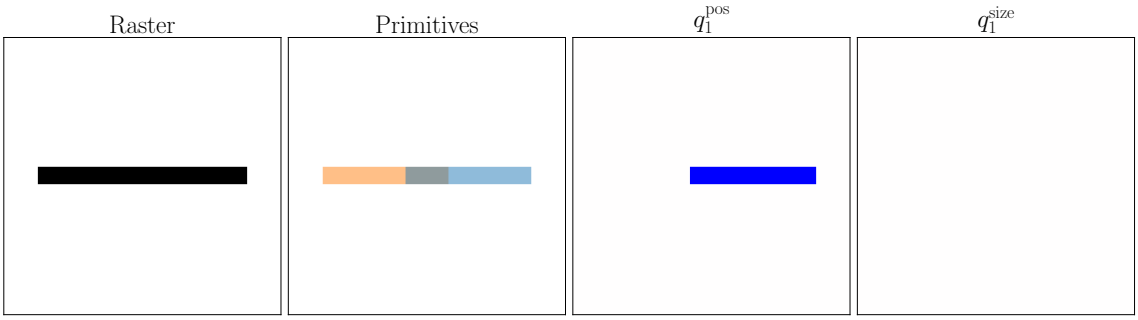


Fig. 23: Primitives covering the same area.

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

$$
\begin{aligned}
q^{\text{rdn}}_{k,i_{\text{pix}}} &= \|\boldsymbol{m}_{k,i_{\text{pix}}}\| \exp\left(-\frac{\left(|\boldsymbol{l}_{k,i_{\text{pix}}} \cdot \boldsymbol{m}_{k,i_{\text{pix}}}| - 1\right)^2}{\left(|\cos\alpha_{\text{col}}| - 1\right)^2}\right), \\
\boldsymbol{m}_{k,i_{\text{pix}}} &= \sum_{j \neq k} \boldsymbol{l}_{k,i_{\text{pix}}} q_{j,i_{\text{pix}}}.
\end{aligned}
\tag{42}
$$

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

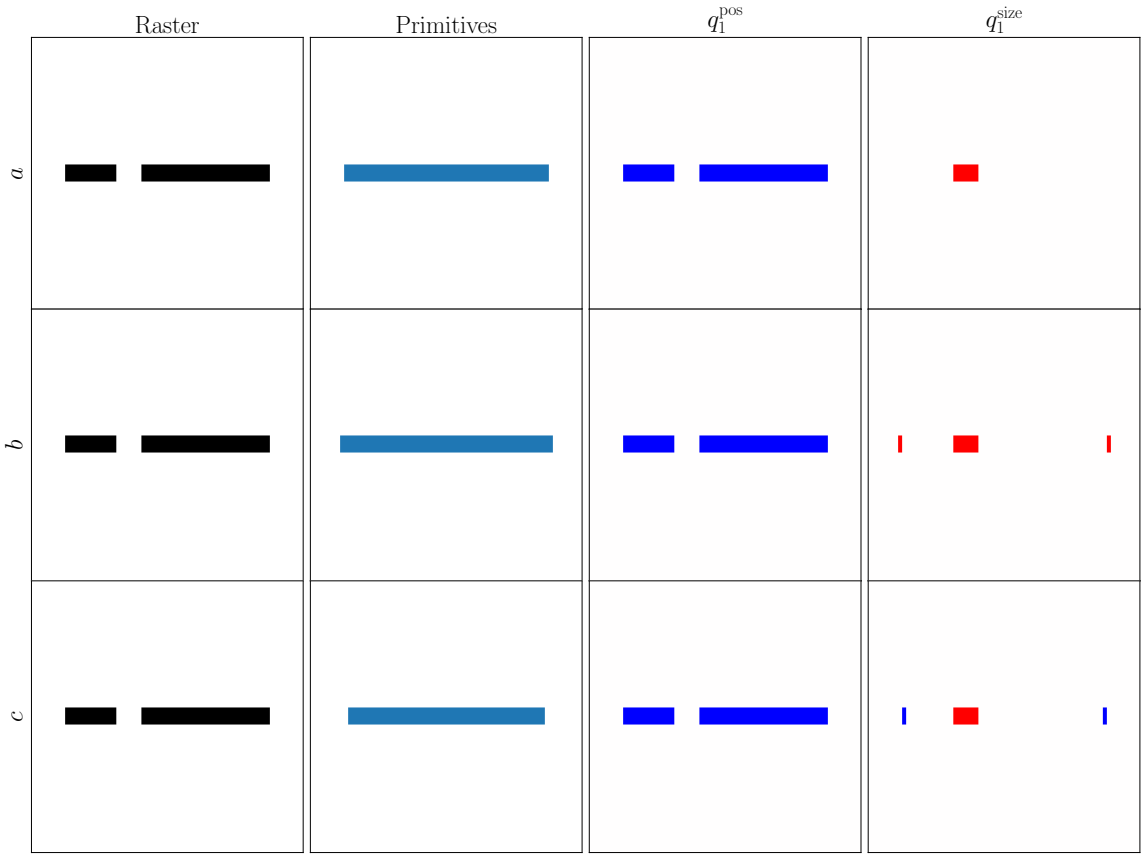


Fig. 24: Undesirable local minimum in which a primitive covers two disconnected raster parts.

Connected area mask. Consider the example in Figure 24 (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 24 (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 25), 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.

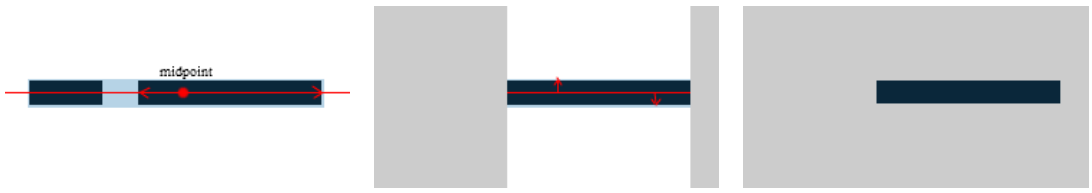


Fig. 25: Stages of $c_{k,i_{\text{pix}}}$ calculation.

For second order Bezier curves, we use a similar definition. Instead of a line we use a parabola containing the Bezier 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

$$q^{\text{size}}_{k,i_{\text{pix}}} = \begin{cases} q_{i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}} & \text{if } c_{k,i_{\text{pix}}} = 1, \\ q_{k,i_{\text{pix}}} & \text{if } c_{k,i_{\text{pix}}} = 0. \end{cases} \tag{43}$$

Connected area mask for positional terms. If we use the same masks for the charges used for the positional terms $q^{\text{pos}}_{k,i_{\text{pix}}}$ 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:

$$q^{\text{pos}}_{k,i_{\text{pix}}} = \begin{cases} \lambda_{\text{pos}} \left(q_{i_{\text{pix}}} - q_{k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}} \right) & \text{if } c_{k,i_{\text{pix}}} = 1, \\ q_{i_{\text{pix}}} - q_{k,i_{\text{pix}}} - \hat{q}_{i_{\text{pix}}} & \text{if } c_{k,i_{\text{pix}}} = 0, \end{cases} \tag{44}$$

The amplification coefficient λ_{pos} is chosen empirically.

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

$$\varphi(r) = e^{-\frac{r^2}{R_{\text{c}}^2}} + \lambda_{\text{f}} e^{-\frac{r^2}{R_{\text{f}}^2}}, \tag{45}$$

This choice allows us to control close interactions at a distance R_{c} independently from far interactions, beyond the characteristic distance R_{f}, for which we choose a lower rate of decay.

Recall that we disable the far interactions in E^{rdn}_k, by setting $\lambda_{\text{f}} = 0$.