Deep Vectorization of Technical Drawings

Vage Egiazarian[1*] [0000−0003−4444−9769], Oleg Voynov[1*] [0000−0002−3666−9166], Alexey Artemov[1] [0000−0001−5451−7492], Denis Volkhonskiy[1] [0000−0002−9463−7883], Aleksandr Safin[1] [0000−0002−5453−1101], Maria Taktasheva[1] [0000−0001−6907−5922], Denis Zorin[2,1], and Evgeny Burnaev[1] [0000−0001−8424−0690]

[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
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

* Equal contribution

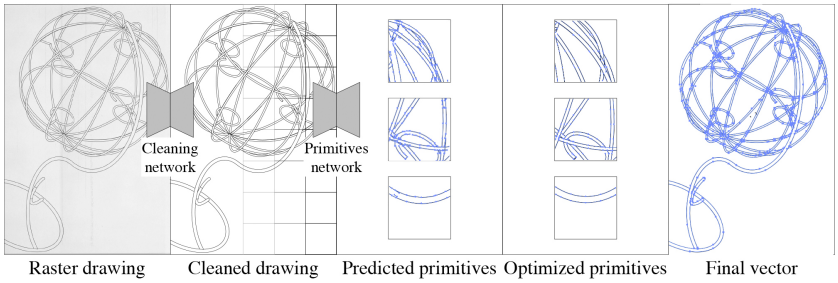

Fig. 1: An overview of our vectorization method. First, the input image is cleaned with a deep CNN. Then, the clean result is split into patches, and primitive placement in each patch is estimated with with a deep neural network. After that, the primitives in each patch are refined via iterative optimization.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

vector representation of a drawing document is unavailable, it is reconstructed, 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 [33]. 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 [27,21] use Potrace as a stage in their algorithms.

Another widely used approach is based on curve network extraction and topology cleanup [11,3,28,6,5,29,17]. 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 [28] 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 [24] 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 [14,39,18]. 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. In [38] 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 [23,35]. 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 [20]. As a result, existing vectorization-related datasets either lack vector annotation (*e.g.*, CVC-FP [16], Rent3D [25], SydneyHouse [7], and Raster-to-Vector [24] all provide semantic segmentation masks for raster images but not the vector ground truth) or are synthetic (*e.g.*, SESYD [8], ROBIN [34], and FPLAN-POLY [31]).

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 [35,32] use convolutional neural networks trained on synthetic data. The authors of [23] 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 [23,35].

Other related work. Methods solving other vectorization problems include, *e.g.*, [37,19], which approximate an image with adaptively refined constant color regions with piecewise-linear boundaries; [26] 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 [23,35], we preprocess the input image with U-net [30] 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 [36]

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

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 output primitives per patch. The maximum number of primitives is set with the size of the 0^{th} embedding $X_0^{\text{pr}} \in \mathbb{R}^{n_{\text{prim}} \times d_{\text{emb}}}$, initialized with positional encoding, as described in [36]. 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 $\Theta = \{\boldsymbol{\theta}_k = (x_{k,1}, y_{k,1}, \ldots, w_k)\}_{k=1}^{n_{\text{prim}}}$, and the confidence values $\boldsymbol{p} \in [0,1]^{n_{\text{prim}}}$. 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

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

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

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

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. 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

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

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

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 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 $\boldsymbol{q} = \{q_i\}_{i=1}^{n_{\text{pix}}}$ 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 three different charge grids, encoded as vectors of length n_{pix}: $\hat{\boldsymbol{q}}$ represents the raster image with charge magnitudes set to intensities of the pixels, $\boldsymbol{q}_k$ represents the rendering of the k^{th} primitive with its current values of parameters, and $\boldsymbol{q}$ represents the rendering of all the primitives in the patch. The charge grids $\boldsymbol{q}_k$ and $\boldsymbol{q}$ 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:

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

The weighting $c_{k,i} \in \{0, 1\}$ enforces coverage of a continuous raster region following the form and orientation of the primitive. 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

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

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

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

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 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 (6) 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

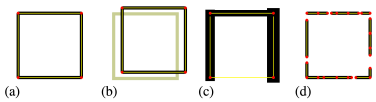

Fig. 2: (a) Ground-truth vector image, and artefacts w.r.t. which we evaluate the vectorization performance (b) skeleton structure deviation, (c) shape deviation, (d) overparameterization.

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 $\mathrm{IoU}(R_1, R_2) = \frac{R_1 \cap R_2}{R_1 \cup R_2}$. It does not capture deviations in graphical primitives that have similar shapes but are slightly offset from each other.

Hausdorff distance

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

and **Mean Minimal Deviation**

$$d_{\mathrm{M}}(X, Y) = \frac{1}{2} \left(\widetilde{d_{\mathrm{M}}}(X \to Y) + \widetilde{d_{\mathrm{M}}}(Y \to X) \right), \tag{13a}$$

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

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 [22]. 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].

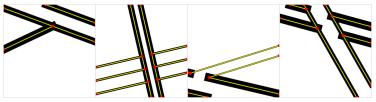

Fig. 3: Examples of synthetic training data for our primitive extraction network.

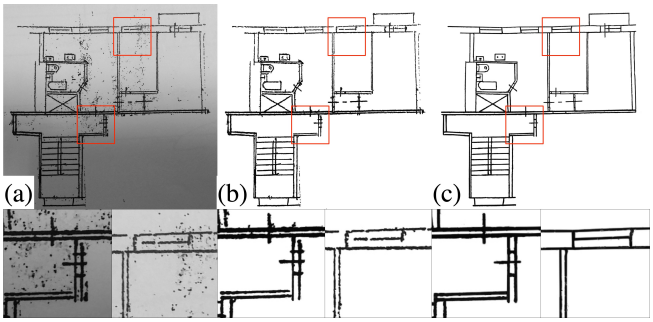

Fig. 4: Sample from DLD dataset: (a) raw input image, (b) the image cleaned from background and noise, (c) final target with infilled lines.

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 precision, so the comparison of the results with different number primitives is not to 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.

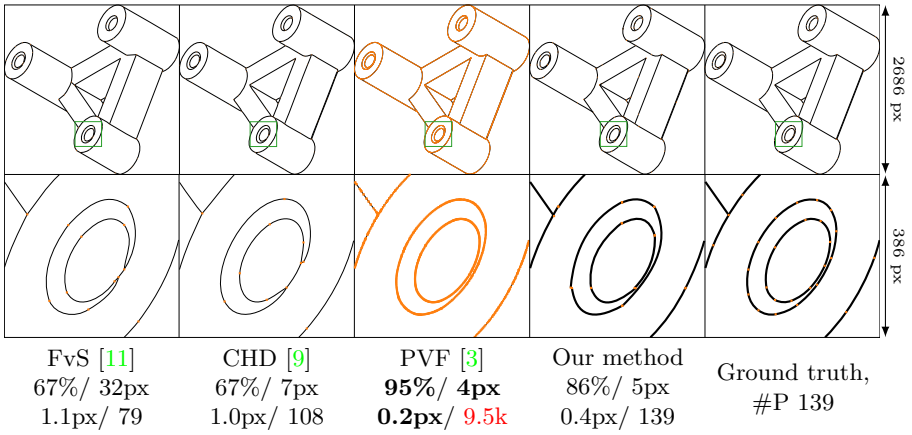

Fig. 6: Qualitative comparison on an ABC image, and values of IoU / d_H / d_M / #P with best in bold. Endpoints of the primitives are shown in orange.

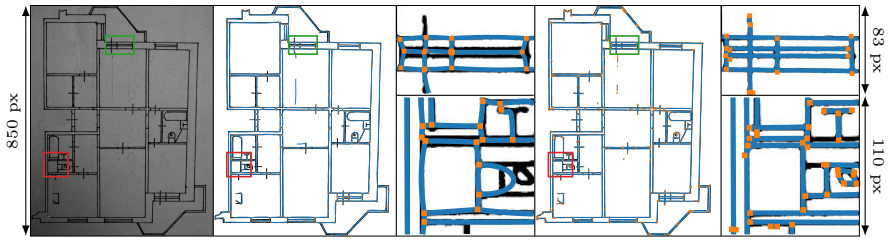

Fig. 7: Qualitative comparison on a real noisy image, and values of IoU / #P with best in bold. Primitives are shown in blue with endpoints in orange on top of the cleaned raster image.

	IoU,%	PSNR
MS [35]	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 [35]. 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.

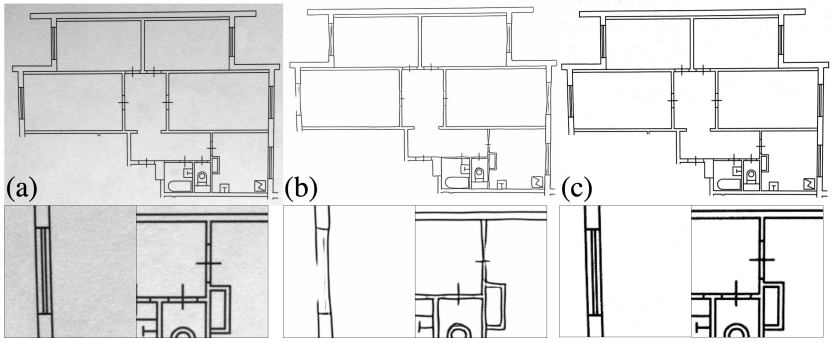

Fig. 8: Example of preprocessing results: (a) raw input image, (b) output of MS [35], (c) output of our preprocessing network. Note the tendency of MS to combine close parallel lines.

	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 9.

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

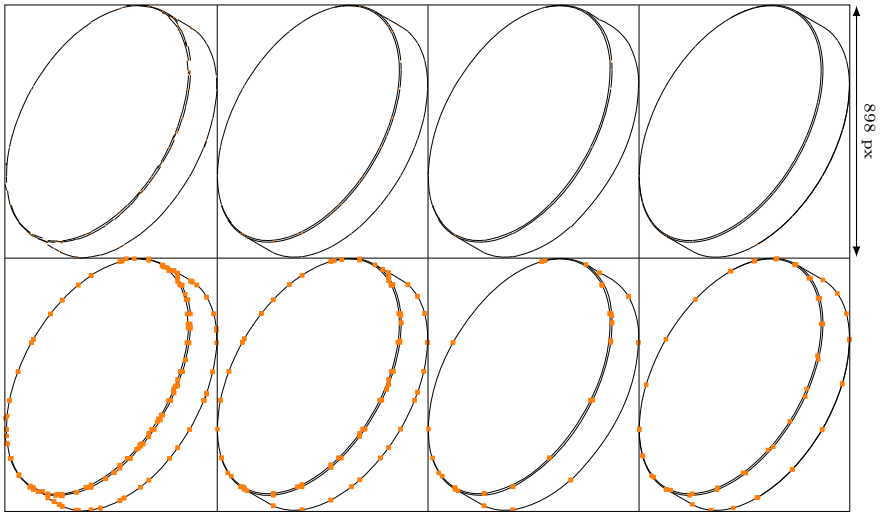

Fig. 9: Results of our method on an ABC image with and without refinement and postprocessing, and values of IoU / d_H / d_M / #P with best in bold. The endpoints of primitives are shown in orange.

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
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
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
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) 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. 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
17. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. IEEE Transactions on Pattern Analysis & Machine Intelligence (6), 890–904 (2006) 3
18. Kaiyrbekov, K., Sezgin, M.: Stroke-based sketched symbol reconstruction and segmentation. arXiv preprint arXiv:1901.03427 (2019) 4

19. Kansal, R., Kumar, S.: A vectorization framework for constant and linear gradient filled regions. The Visual Computer **31**(5), 717–732 (2015) 5
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) 4
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) 3
22. 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
23. Li, C., Liu, X., Wong, T.T.: Deep extraction of manga structural lines. ACM Transactions on Graphics (TOG) **36**(4), 117 (2017) 4, 5
24. 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
25. 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
26. 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) 5
27. Munusamy Kabilan, V., Morris, B., Nguyen, A.: Vectordefense: Vectorization as a defense to adversarial examples. arXiv preprint arXiv:1804.08529 (2018) 3
28. Najgebauer, P., Scherer, R.: Inertia-based fast vectorization of line drawings. Comput. Graph. Forum **38**, 203–213 (2019) 3
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) 3
30. 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
31. 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
32. 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
33. Selinger, P.: Potrace: a polygon-based tracing algorithm. Potrace (online), http://potrace.sourceforge.net/potrace.pdf (2009-07-01) (2003) 3
34. 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
35. 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
36. 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) 5, 6

37. 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
38. Zheng, N., Jiang, Y., Huang, D.: Strokenet: Aneural painting environment 4
39. 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