Deep Sketch Vectorization via Implicit Surface Extraction

CHUAN YAN, George Mason University, U.S.A
YONG LI, South China University of Technology, China and George Mason University, U.S.A
DEEPALI ANEJA, Adobe Inc., U.S.A
MATTHEW FISHER, Adobe Inc., U.S.A
EDGAR SIMO-SERRA, Waseda University, Japan
YOTAM GINGOLD, George Mason University, U.S.A

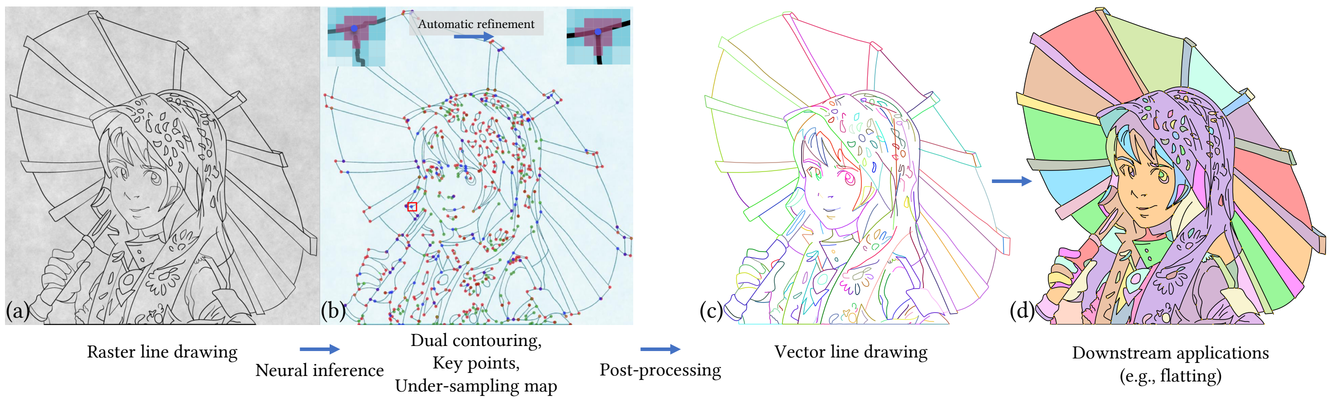

Fig. 1. We propose a fast and accurate vectorization method that can turn clean raster line drawings with complex topology (a) into high quality vector graphics (c). Our results are sufficiently detailed for automatic downstream processing (e.g., [Yin et al. 2022] (d)). Neural networks predict dual contouring inputs, key point locations, and under-sampled regions (b). The key points and under-sampled regions are used to automatically refine the initial vectorization produced by dual contouring, correcting incorrect topology (b, top). Our method is also capable of handling high valence star-junctions. Girl image ©David Revoy CC-BY-4.0.

We introduce an algorithm for sketch vectorization with state-of-the-art accuracy and capable of handling complex sketches. We approach sketch vectorization as a surface extraction task from an unsigned distance field, which is implemented using a two-stage neural network and a dual contouring domain post processing algorithm. The first stage consists of extracting unsigned distance fields from an input raster image. The second stage consists of an improved neural dual contouring network more robust to noisy input and more sensitive to line geometry. To address the issue of under-sampling inherent in grid-based surface extraction approaches, we explicitly predict undersampling and keypoint maps. These are used in our post-processing algorithm to resolve sharp features and multi-way junctions. The keypoint and undersampling maps are naturally controllable, which we demonstrate in an interactive topology refinement interface. Our proposed approach produces far more accurate vectorizations on complex input than previous approaches with efficient running time.

CCS Concepts: • **Computing methodologies** → *Image processing*; *Shape analysis.*

Authors' addresses: Chuan Yan, cyan3@gmu.edu, George Mason University, U.S.A; Yong Li, pressure36@gmail.com, South China University of Technology, China and George Mason University, U.S.A; Deepali Aneja, aneja@adobe.com, Adobe Inc., U.S.A; Matthew Fisher, matfishe@adobe.com, Adobe Inc., U.S.A; Edgar Simo-Serra, ess@waseda.jp, Waseda University, Japan; Yotam Gingold, ygingold@gmu.edu, George Mason University, U.S.A.

Additional Key Words and Phrases: vectorization, raster, sketch, drawing

ACM Reference Format:
Chuan Yan, Yong Li, Deepali Aneja, Matthew Fisher, Edgar Simo-Serra, and Yotam Gingold. 2024. Deep Sketch Vectorization via Implicit Surface Extraction. *ACM Trans. Graph.* 43, 4, Article 37 (July 2024), 13 pages. https://doi.org/10.1145/3658197

1 INTRODUCTION

Vector graphics offer numerous advantages compared to raster images, including enhanced precision, scalability, and editability. Many algorithms have been proposed for downstream processing of raw vector sketchs, e.g., [Gryaditskaya et al. 2020; Kaplan and Cohen 2006; Liu et al. 2018; Shao et al. 2012; Whited et al. 2010; Yang et al. 2018; Yin et al. 2022]. However, the world's sketches are often locked away in raster formats [Yan et al. 2020]. They may be digitized from the real world, drawn in a raster graphics program, or rasterized for other reasons.

For line drawings such as sketches, general image vectorization approaches are unsuitable. They output colorful regions rather than paths for stroke centerlines [Dominici et al. 2020; Lai et al. 2009; Zhu et al. 2022]. Vectorizing line drawings, particularly hand-drawn ones with high image resolution and complex line topology, remains a challenging task. Techniques for line drawing vectorization typically address this problem by breaking it down into optimization sub-tasks such as line extraction and topology refinement [Bessmeltsev and Solomon 2019; Favreau et al. 2016; Gutan et al. 2023; Noris et al. 2013; Puhachov et al. 2021]. However, these methods are computationally complex, and their processing time increases significantly when applied to line drawings encountered in real-world scenarios [Yan et al. 2020]. Several recent deep learning techniques have been proposed to convert raster input into centerlines for individual sketch strokes [Carlier et al. 2020; Das et al. 2021; Egiazarian et al. 2020; Ha and Eck 2017; Liu et al. 2022; Lopes et al. 2019; Mo et al. 2021]. However, these methods typically adopt a fully end-to-end approach, where the network learns to extract lines and fit vector parameters. These approaches don't scale to complex inputs, often omitting lines.

We propose an approach to sketch vectorization that generates far more accurate output, from more complex sketches. Our key insight is treating centerline extraction as the problem of implicit surface extraction from unsigned distance fields, allowing us to take advantage of and improve upon recent work on Neural Dual Contouring [Chen et al. 2022]. For topology reconstruction, we also predict an undersampling map, which identifies where the neural network's raster sampling rate is unable to capture stroke detail. Motivated by the observation that topology is typically local, we extend the fully convolutional neural network of Puhachov et al. [2021] for predicting key points such as end points, sharp turns, T or Y junctions, etc. Together, this information leads to a post-processing algorithm that overcomes the accuracy and resolution limits of dual contouring approaches alone. Furthermore, the architecture of our approach, which predicts an undersampling map (indicating regions with a sampling rate too low to recover accurate geometry) and keypoint locations, lends itself to interactive topology editing (Section 8.4). Other than Section 8.4, all results shown were generated automatically with no user interaction.

Sketch consolidation and cleanup are outside the scope of our research. Our work aims to faithfully vectorize all strokes in clean or messy raster line drawings, including physical artifacts like paper texture and lighting variation. Numerous techniques could be used for pre-processing (e.g. [Simo-Serra et al. 2018]) or post-processing (e.g. [Liu et al. 2018]).

2 RELATED WORK

Converting raster images to vector graphics representations has been extensively studied for decades. These studies can be broadly categorized into two groups: extraction of vector regions (bounded by closed curves) and strokes (typically open).

Region vectorization. Many approaches are based on outputting vector graphics representations of colorful regions bounded by closed curves (e.g. photographs or clip art). Output formats include regions filled with opaque linear or radial gradients [Dominici et al. 2020; Lecot and Levy 2006], translucent layers [Du et al. 2023; Favreau et al. 2017; Reddy et al. 2021], gradient meshes [Lai et al. 2009; Zhu et al. 2022], and diffusion curves [Orzan et al. 2008; Zhao et al. 2018]. These methods are unsuitable for vectorizing line drawings, which are collections of (often open) curves. At best, they produce long, thin regions conforming to the outlines of strokes.

Sketch vectorization. Other approaches, like, ours, aim to output a set of open or closed curves in the plane that correspond to the centerlines of strokes in a raster image. (Sketch vectorization is also related to simplifying or consolidating a rough sketch given in raster or vector format. See Yan et al. [2020] for an overview and recent benchmark.) Early approaches employed techniques such as thresholding (binarization) and thinning to extract a 1-pixel width center line [Hilaire and Tombre 2006]. However, these methods often produce inaccurate or messy vector strokes, particularly for complex, noisy, or anti-aliased drawings. Donati et al. [2018] showed that adaptive thresholding and skeletonization can heuristically reduce these artifacts. Parakkat et al. [2018] relied on simple thresholding but insights related to the shape of triangles in a Delaunay Triangulation of the shape to obtain higher quality skeletons. Zhang et al. [2022] introduced an algorithm for replacing unreliable skeletonizations of thick strokes with information obtained from the stroke boundaries. Noris et al. [2013] proposed to use image gradients to determine stroke topology and centerlines. However, this method still produces unreliable vector strokes due to insufficient local information provided by the gradient field alone. In contrast, we use a convolutional neural network (CNN) to accurately predict stroke centerlines and auxiliary information as distance fields for topological reconstruction.

Favreau et al. [2016] used a simple approach to find 1-pixel skeletons by thresholding (for open strokes) and computing boundary pixels of trapped-ball regions (closed strokes). The focus of their work was on finding a simple curve network with accurate junctions. Bessmeltsev and Solomon [2019] proposed to treat the problem of curve smoothness and junction topology as one of finding a smooth polyvector field. This approach, and the numerous follow-up works [Bessmeltsev and Solomon 2019; Gutan et al. 2023; Puhachov et al. 2021; Stanko et al. 2020] produce high quality curve and junction shapes, but have two primary drawbacks: (1) they rely on thresholding to identify the narrow region around strokes, and (2) they are slow. Of particular relevance to our work, Puhachov et al. [2021] trained a CNN to predict the location and type of key points; we base our distance field prediction network on theirs (Section 5). Bao and Fu [2023] combined tangent fields and gradient fields to achieve good results on clean sketches. In contrast to these approaches, we treat the stroke geometry extraction problem as one of extracting the zero level set of an unsigned distance field. We recover junctions by predicting the undersampling area around junctions, and gluing appropriate topology based on the junction's boundary.

Other techniques for line drawing vectorization have approached it as a sequence generation problem and used recurrent neural networks (RNN) [Carlier et al. 2020; Das et al. 2021; Ha and Eck 2017; Liu et al. 2022; Lopes et al. 2019; Mo et al. 2021]. These approaches typically project the raster sketch into a latent space and then use a generative decoder to produce a sequence of vector graphics primitives such as points or Bezier curve parameters. As revealed in our evaluation, they fail to capture details in complex input. Guo et al. [2019] introduced a CNN-based approach to output an image with all stroke centerlines and an image with all junctions. The junction information is used to separate individual strokes with the help of a second network. This is reminiscent of our distance field network's output. However, by outputting in the distance field domain, we can

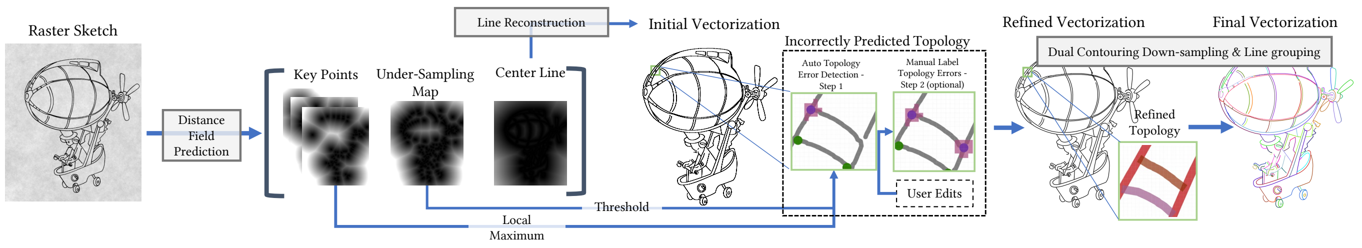
Raster Sketch
Distance Field Prediction
Key Points
Under-Sampling Map
Center Line
Line Reconstruction
Initial Vectorization
Incorrectly Predicted Topology
Auto Topology Error Detection - Step 1
Manual Label Topology Errors - Step 2 (optional)
User Edits
Threshold
Local Maximum
Refined Vectorization
Final Vectorization
Dual Contouring Down-sampling & Line grouping
Refined Topology

Fig. 2. The overall pipeline of our sketch vectorization approach. First, a Distance Field Prediction network outputs implicit stroke centerlines, key point locations, and regions likely undersampled. Next, a Line Reconstruction network produces an initial vectorization. The undersampled regions and keypoints are used to refine the vectorization. Finally, we extract long, connected chains of edges. Balloon image ©David Revoy CC-BY-4.0.

directly obtain vector strokes via dual contouring, which allows our approach to scale to more complex input.

Dual contouring isosurface extraction. Dual contouring (DC) [Ju et al. 2002] is an elegant approach for isosurface/isocurve extraction from signed distance fields. In dual contouring, a grid is overlaid on the domain. All grid edges identified as intersecting the isosurface (via a sign change at either end), and all grid cells incident to intersecting edges, produce a geometric dual in the output. Grid cells produce dual vertices in the output and grid edges produce dual edges (in 2D) or faces (in 3D) connecting those vertices. Dual vertex positions are chosen as the minimizer of an objective function using distance field gradients. Recently, Chen et al. [2022] proposed Neural Dual Contouring, in which a neural network outputs edge flags and vertex positions for a given distance field. Importantly, Neural Dual Contouring enables dual contouring on *unsigned* distance fields for the first time, which are needed for stroke centerline distance maps.

3 OVERVIEW

Figure 2 summarizes the architecture of our method. Our architecture consists of two sub-networks (**Distance Field Prediction** and **Line Reconstruction**), followed by post-processing algorithms tailored for the dual contouring domain.

The first sub-network performs **Distance Field Prediction** given an input raster sketch (Section 5). This network is a combination of image-to-image translation and super-resolution. The network outputs a set of Unsigned Distance Fields (UDF) with double the input resolution. One of the distance fields encodes stroke centerlines. Super-resolution greatly reduces vectorization errors and limits them to multi-way junctions and sharp corners. The other distance fields encode such likely under-sampled regions and keypoint locations. This allows our post-processing to recover precise geometry and connectivity.

The centerline UDF is then passed to the **Line Reconstruction** network (Section 6). This network is based on Neural Dual Contouring [Chen et al. 2022] for zero isocurve extraction. The output takes the form of three maps: two with flags for horizontal and vertical grid edge crossings, and one with the grid cell dual positions. (See Section 2 for a brief summary of dual contouring.)

The raw network outputs are combined and post-processed with methods tailored to the dual contouring domain (Section 7). These methods can automatically refine the topology at sharp turn and multi-way junctions, remove redundant lines, connect broken strokes, and output long, continuous strokes.

Details about an additional layer in each sub-network to address the conversion of data between the raster and UDF domain can be found in the supplementary materials.

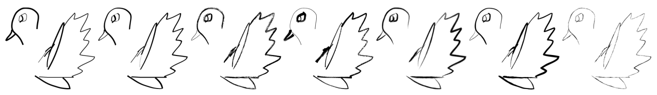

Fig. 3. Example of brush augmentations. We augment each sketch with 7 different brush styles, and we randomly set the every stroke width during each augmentation.

4 TRAINING DATASET

We used a similar methodology to Puhachov et al. [2021] to construct our training dataset. We did not use Puhachov et al. [2021]'s dataset directly, since it primarily focuses on key point detection, resulting in certain brush styles being too thick for vectorization. Instead, we created a dataset with synthetic brushes more akin to hand-drawn sketches encountered in the wild. We sampled 10,000 and 53,000 vector sketches from the Quick Draw![1] and Creative Sketch [Ge et al. 2021] datasets, respectively.

Raster input. Each sketch was rasterized using 7 different brush styles, with randomly varying stroke widths (Figure 3). This resulted in a total of 441,000 raster sketches for our training input. During training, each raster sketch was also composited onto a randomly generated paper texture, as shown in Figure 1 (a). The testing set, discussed in Section 8, consists of a disjoint set of 369 professionally cleaned vector sketches from Yan et al. [2020]'s sketch cleanup benchmark.

UDF & Dual Contouring Ground Truth. We extract several components to serve as targets for network training: (a) **keypoint**, (b) **center line**, (c) **edge flags**, (d) **vertex map**, and (e) **under sampling map**. Components (a), (b), and (e) are encoded as Unsigned Distance Fields (UDF) and function as the ground truth for our Distance Field Prediction network, while components (c) and (d) form the basis of the ground truth for our Line Reconstruction network. Details are provided in Sections 5 and 6, respectively.

[1]https://github.com/googlecreativelab/quickdraw-dataset

5 DISTANCE FIELD PREDICTION

For a given raster input sketch, the Distance Field Prediction network encodes centerline and keypoint information using six UDFs: $U_{centerline}$ encodes the distance to strokes' centerlines; U_{USM} encodes the Under Sampling Map (USM), or distance from under-sampled grid points in the dual contouring domain; and, similar to Puhachov et al. [2021], $U_{end,sharp,junc}$ encode the distance to keypoints (endpoints, sharp corners, and junctions, respectively) and U_{all} redundantly encodes the distance to all three types of keypoints.

Ground truth UDFs creation. Rather than pre-computing ground truth UDFs (Unsigned Distance Fields), we generate them dynamically. This is because augmentation methods such as random cropping and rotation do not commute with distance field creation. (Consider that cropping a vector sketch could eliminate the closest stroke to many pixels in the rasterized UDF. Rotating the rasterized UDF would introduce sampling artifacts.) Therefore, in order to obtain a precise UDF, we apply the random augmentation to the vector sketch and generate the UDF on the fly. To avoid a performance bottleneck, we use a spatial acceleration data structure to quickly find the closest point on the cropped vector paths to each pixel. Each line segment's axis-aligned bounding box (AABB) provides a lower bound on the point-to-line segment distance. In our experiments, a simple linear sweep across AABBs was as fast as building and querying an AABB tree.

Network Training and Objective Function. We modified the network architecture used by Puhachov et al. [2021] to improve its performance and add more prediction branches for the $U_{centerline}$ and U_{USM} outputs. We increased the network's inner channel from 128 to 256 and also increased the cardinality of the ResNext module from 3 to 8 with gradually increasing dilation steps for a larger receptive field. We train our Center Line Extraction network C by minimizing:

$$\theta^* = \arg\min_{\theta} \mathbb{E}\left[\Sigma_{i=1}^{6} L_{line}(UDF_i, C_i(I;\theta))\right] \quad (1)$$

where θ are the model parameters, C_i is the i-th prediction branch, $UDF_i \in \mathbb{R}^{(2W+1)\times(2H+1)}$ is the i-th UDF channel, and $I \in \mathbb{R}^{W\times H}$ is the input raster sketch.

Because pixels representing strokes are typically sparser than the background, our loss function L_{line} uses the masked l_1 distance:

$$L_{line}(UDF_i, \widehat{UDF}_i) = average(|UDF_i - \widehat{UDF}_i| \odot M_i) \quad (2)$$

where the mask $M_i \in \mathbb{R}^{(2W+1)\times(2H+1)}$ comes from the ground truth UDF_i binarized with distance threshold 4.5 pixel units and $\widehat{UDF}_i = C_i(I;\theta)$. The rationale for setting the threshold at 4.5 is to ensure that all branches within the subsequent **Line Reconstruction** network, which have a maximum receptive field of size 7×7 in their convolutional kernels, receive enough valid information. This setting aims to minimize the area that the NDC network needs to focus on, while still ensuring sufficient information acquisition.

The network training converged after approximately 72 GPU hours on a single Nvidia A100 GPU.

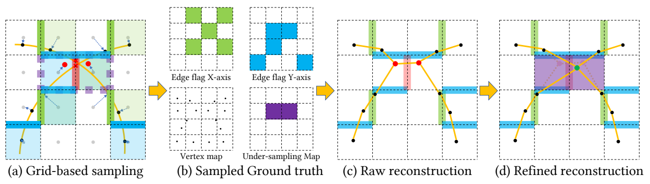

Fig. 4. The process of creating ground truth for the 2D Neural Dual Contouring network, and the process of reconstruction and refinement based on the key point and under sampling map. (a) Given a target vector graphic, we sample it with a fixed size grid. (b) We detect and count crossings of each grid edge and sample one vertex within each grid cell to extract edges flags, the vertex map, and the under-sampling map. (c) The edge flags and vertices alone are insufficient to correctly reconstruct junctions when a grid edge is crossed multiple times. (d) Simple surgery based on the under-sampling map can still recover the correct topology in many cases.

6 LINE RECONSTRUCTION

After obtaining the centerline UDF from the Distance Field Prediction network, we obtain our raw vectorization by extracting the zero isocurves via an improved Neural Dual Contouring network [Chen et al. 2022].

Ground Truth Creation. For each vector sketch, we follow Chen et al. [2022] to define its **vertex map** $V \in \mathbb{R}^{(W\times H\times 2)}$ and **edge flag map** $E \in \mathbb{R}^{(W\times H\times 2)}$. We further introduce the concept of an **Under-Sampling Map (USM)** $U \in \mathbb{R}^{(W\times H)}$. We first define a grid by picking a sampling rate (0.5-pixel in our case). For each cell containing a vector path, $V_{(x,y)}$ stores the point on the vector path (black dots in Figure 4a and b) closest to the grid center (gray dots in Figure 4a). Then, we find intersections between the vector paths and grid edges (Figure 4a). If an intersection is found with the grid edge of a cell at location (x, y), we label the corresponding position in the edge flag map $E_{(x,y)}$ (green and blue edges in Figure 4a and blocks in Figure 4b). If two or more intersections are detected with the same edge (red edge in Figure 4a), this indicates an under-sampling issue. Dual contouring reconstruction only allows for one vertex per grid cell and one crossing per edge. The signal cannot be correctly reconstructed (Figure 4c) in the under-sampled region (purple dotted line in Figure 4a). In this case, we assign labels to the cells on both sides of this edge in our USM $U_{(x,y)}, U_{(x-1,y)}$ for a vertical edge or $U_{(x,y)}, U_{(x,y-1)}$ for a horizontal edge (purple cells in Figure 4b and Figure 4d).

Network Training and Objective Function. Our Line Reconstruction network is built on the Neural Dual Contouring (NDC) network introduced in [Chen et al. 2022] adapted to 2D. However, when applying the original NDC network to sketch vectorization, we encountered certain limitations. The original NDC network is sensitive to noisy signals due to the predicted UDF inherently containing noise. This sensitivity made it difficult to find clean and contiguous centerlines when strokes had varying thickness. Additionally, accurately reconstructing junction regions was challenging due to the undersampling issue, even when the edge flag map was accurately predicted.

We therefore proposed two improvements in our Line Reconstruction network. We first introduce **multi-resolution convolution**

branches comprising three consecutive residual convolution layers. This choice corresponds to employing 3×3, 5×5, and 7×7 grid sizes during Dual Contouring inference. Each branch follows the same design, with the only difference being the dilation size of the convolution kernel, which is set to 1, 2, and 3, respectively. Then we ask the network to predict an addition channel which encodes the 1-pixel width **sketch skeleton**. Together with our proposed skeleton loss function, these changes encourage the network to use context from a larger receptive field through the multi-resolution convolution branches and generate line segments more likely to be connected.

We formulate our Line Reconstruction network N prediction as a pixel-wise classification task, which we express as:

$$\theta^* = \arg\min \mathbb{E}\left[L_{Rec}(N(U_{centerline}; \theta), E, V, S)\right] \tag{3}$$

where $U_{centerline} = UDF_1$ is the input Unsigned Distance Field and E, V, and S are the ground truth edge flag map, vertex map, and skeleton map, respectively. We optimize for the network's parameters via a loss function L_{Rec}. The L_{Rec} function contains 3 different loss terms, which are edge map loss, vertex loss, and skeleton loss, respectively. We formulate it as:

$$L_{Rec} = L_{edge}(\widehat{E}, E) + 0.5 \cdot L_{vertex}(\widehat{V}, V) + 0.01 \cdot L_{skeleton}(\widehat{S}, S) \tag{4}$$

where $\widehat{E} = N_E(U_{centerline})$, $\widehat{V} = N_V(U_{centerline})$, $\widehat{S} = N_S(U_{centerline})$, $E \in \mathbb{R}^{W \times H \times 4}$, $V \in \mathbb{R}^{W \times H \times 2}$, and $S \in \mathbb{R}^{W \times H}$. N_E, N_V, and N_S represent the branches predicting the edge map, vertex map and skeleton map in our Line Reconstruction network, respectively.

L_{edge} is a masked cross entropy loss similar to L_{line} (Eq. 2), L_{vertex} is an L_2 loss, and $L_{skeleton}$ is a binary cross entropy loss. Specifically, we define the edge flag prediction as a 4-channel output, with each channel representing the None flag, X-axis flag, Y-axis flag, and both-axis flag, respectively. We then convert the ground truth edge flag map into a 4-channel one-hot vector map.

The convergence of the network required approximately 24 GPU hours on a single Nvidia A100 GPU.

End-to-End training. After the two sub-networks have been properly trained, we continue fine-tuning the end-to-end pipeline with an additional 24 GPU hours. The joint training pipeline can be expressed as:

$$\theta_N^*, \theta_C^* = \arg\min \mathbb{E}\left[L_{Rec}(N(C(I; \theta_C); \theta_N), E, V, S) + \Sigma_{i=1}^{6} L_{\text{line}}(UDF_i, C_i(I; \theta_C))\right] \tag{5}$$

7 POST-PROCESSING

The raw output of our vectorization pipeline contains several types of artifacts: incorrect reconstructions due to under-sampling (Figure 4); and parallel edges (Figure 5a), gaps (Figure 6), and jagged lines (Figure 10 (a3)) due to redundant or missing edge flags. We propose post-processing algorithms to resolve them. (Chen et al. [2022] proposed a different, 3D post-process to address gaps in their output.)

Edge flag refinement. A raw prediction may contain redundant edge flags (Figure 5a) or broken lines (Figure 6). We propose two simple morphological image operations to address this. To address redundant edges, we first obtain the occupancy map as a bit-wise

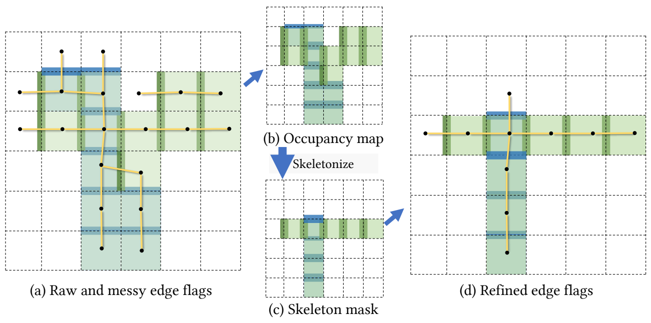

Fig. 5. Edge flag refinement. The raw edge flags from our Line Reconstruction network may contain parallel repetitive messy lines (a). The combined X-axis and Y-axis edge flags could be interpreted as a "imperfect" skeleton (b) because the ideal skeleton map of a sketch should always be 1 pixel width (c). We therefore could apply a skeletonize algorithm to remove this type of redundant edge flags(d).

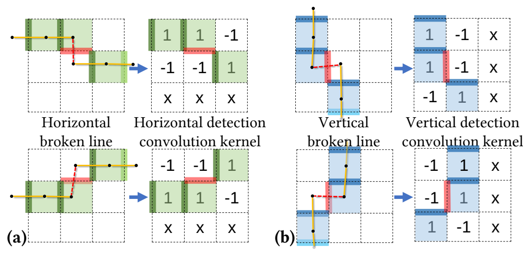

Fig. 6. Example of (a) horizontal and (b) vertical broken lines in the edge flag map. The convolution kernels which defined by the edge flag position (green or blue edge flag on the edges) detect broken edges (red flag on the edges) that should be flagged. X's represents ignored values in the convolution kernel.

logical OR operation of the X-axis and Y-axis edge flags (Figure 5b). We then skeletonize it [Zhang and Suen 1984] to obtain a 1-pixel width skeleton mask (Figure 5c) and eliminate all edge flags outside of the mask (Figure 5d).

We observed four common broken line patterns in the edge flag maps (Figure 6). To address them, we designed four $3{\times}3$ convolution kernels to locate positions that need to restore missing edge flags. This operation is efficient and can also be easily extended to refine other types of topology errors in the future.

Topology refinement. We propose a topological surgery algorithm based on the **under-sampling map** and **key point** locations. Our observation is that we can recover precise topology by removing all edge flags within the under-sampling map (Figure 7b) and reconnecting the truncated edges at the boundary (Figure 7c). The valence can be determined by counting the number of truncated edges. If the valence of the under-sampling map is 4 and no key points are present, pairs of truncated edges are connected to each other based on matching tangent directions (Figure 7c). Otherwise, if a keypoint is present within the under-sampling map, all truncated edges are directly connected to it. If no keypoint is present, the edges are connected to the average position of the removed grid

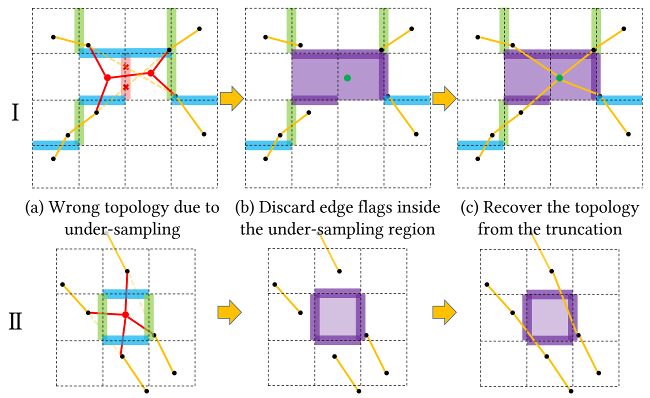
I
II
(a) Wrong topology due to under-sampling
(b) Discard edge flags inside the under-sampling region
(c) Recover the topology from the truncation

Fig. 7. The general methodology for topology recovery in an under-sampling map. An under-sampling map can arise in two scenarios: junctions or parallel lines that are in close proximity to each other (row I and II, respectively). When the grid edge intersects multiple vector paths (I.(a)), or when all four edges of a grid exhibit intersection points (II.(a)), we classify that particular grid cell as under-sampled (purple cells), as seen in (b). To refine the topology within this region, we first extract each individual under-sampling map and compute its valence by counting the number of truncated edges (the purple edge in (b)), as demonstrated in (c). We then proceed to select and remove all edge labels and center points contained within this region, as shown in (b). Subsequently, we retrieve all key points located within this region and perform reconstruction based on the retrieval results. If a key point is found within this region, we reconstruct the junction by reconnecting all the truncated points to this key point (I.(c)). Alternatively, if no key point is detected and the region's valence is 4, we reconstruct the parallel line by connecting the pairs of truncated points that possess the closest tangent direction (II.(c)).

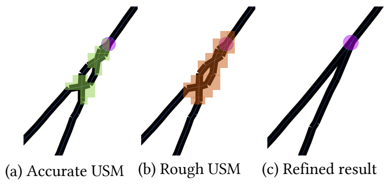

Fig. 8. Refinement based on under-sampling maps with varying accuracy. Both (a) accurate and easier-to-generate (b) approximate under-sampling maps produce virtually the same refined result (c).

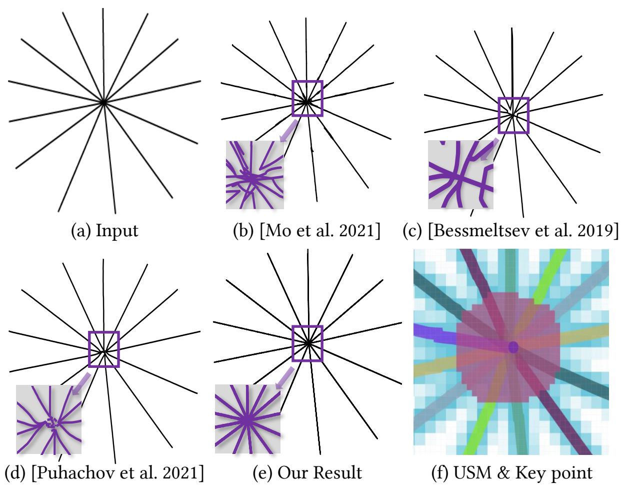

Fig. 9. Resolving a star junction in a raster line drawing. (a) Input image. (b), (c), (d) Comparison results obtained from other methods, with zoom in of the line structure at bottom left. (e) Our output. (f) Detected key point, under-sampling map and zoom in reconstructed topology.

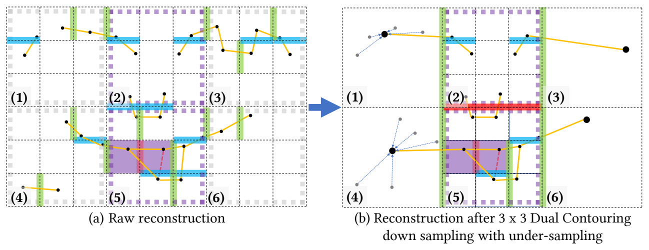
(1)
(2)
(3)
(4)
(5)
(6)
(a) Raw reconstruction
(b) Reconstruction after 3 x 3 Dual Contouring down sampling with under-sampling

Fig. 10. An example of 3×3 Dual Contouring edge flag down-sampling with the under-sampling map. Our down-sampling algorithm labels a 3×3 block as under-sampled if it contains predicted under-sampling (a5) or its edges contain multiple crossings (a2). Then, only the geometry in un-labeled blocks (b1, b3, b4, b6) is down-sampled. This process naturally connects broken lines (b1), smooths jagged lines (b3), and removes noisy stray lines (b4).

vertices. Our topology refinement method is capable of reconstructing intricate topology when provided with reasonably accurate key points and an USM (Figure 8). Our approach successfully recovers star-junction topology, which has historically posed challenges for previous methods (Figure 9).

Dual contouring downsampling. Our 2× super-resolution reconstruction is beneficial for complex structures, such as sharp turns or multi-way junction regions, where a dense sampling rate is necessary. However, it is wasteful to use so many line segments for simple curves. We propose a dual-contouring domain downsampling algorithm to simplify undesirable complexity.

Without loss of generality, we describe our approach for 3× downsampling. We first group each 3×3 block of grid cells into a coarser grid (Figure 10a). Then we detect crossings of coarse grid edges to create coarse edge flags. If the coarse grid cell contains exactly one open endpoint, the endpoint is used as the coarse cell's vertex (Figure 10 (b3, b6)). Otherwise, the average of grid vertices on the finer path inside is used (Figure 10 (b1, b4)). If multiple crossings are detected across a single coarse edge (Figure 10 (a2)) or one 3×3 block contains predicted under-sampling map (Figure 10 (a5)), we label the coarse grid cell and its neighbor across the edge as under-sampled. In this case, we simply preserve the finer geometry inside the under-sampled region, which is equivalent to keeping the sampling rate unchanged (Figure 10 (a) compared to (b)).

In this way, the algorithm adaptively selects the appropriate grid size to reduce the overall sampling rate and number of line segments while maintaining the same vector path quality, See Figure 11) for an 8× dual contouring downsampling example.

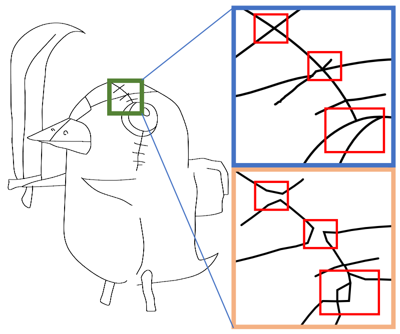

Fig. 11. 8× dual contouring downsampling with and without the under-sampling map. In the top close-up, down-sampling preserves edges inside the red boxes via the under-sampling map. In the bottom close-up, down-sampling loses the structure information due to the lower sampling rate. Image ©VFS Digital Design CC-BY-2.0.

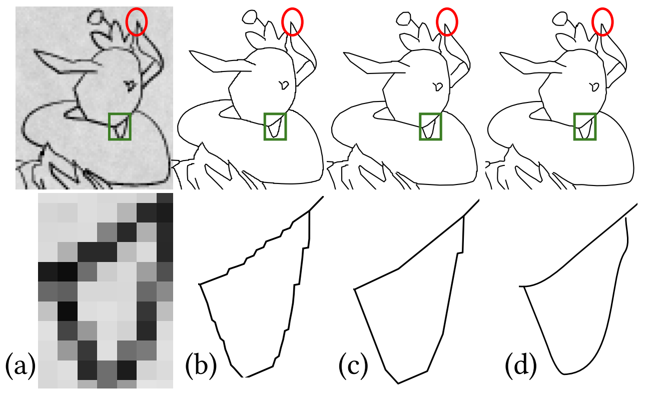

Fig. 12. Comparison of results using different line smoothing techniques. The bottom row contains a close-up view of the green boxes. The input raster sketch (a) produces jagged line artifacts when vectorized with our technique (b). Ramer-Douglas-Peucker (RDP) line simplification (c) and Schneider [1990]'s Bézier curve fitting (b) both effectively mitigate the issue. However, this simple Bézier curve fitting approach smooths sharp corners (red circle). Image©Krenz Cushart CC-BY-NC-4.0.

Line grouping & smoothing. Finally, we use a straightforward algorithm for grouping lines. We compute all-pairs' shortest paths on the entire graph of lines, select the longest shortest path, remove it from the graph, and repeat until no paths are left. After line grouping, jaggy artifacts may still exist due to incorrect edge flag or vertex predictions (Figure 12a). To resolve this, we experimented with Ramer-Douglas-Peucker (RDP) line simplification and Schneider [1990]'s Bezier curve fitting to simplify the output paths. They simplify and smooth the paths (Figure 12), respectively, with negligible performance overhead.

8 RESULTS

To evaluate the effectiveness of our method, we conducted a comparative analysis against 3 state-of-the-art line drawing vectorization methods [Bessmeltsev and Solomon 2019; Mo et al. 2021; Puhachov et al. 2021].

8.1 Benchmark

Dataset. We curated a **cleaned test dataset** consisting of 369 professionally cleaned vector sketches from the rough sketch benchmark dataset [Yan et al. 2020]. To simplify our metrics, we selected sketches from the benchmark dataset containing only curves (polylines, quadratic, or cubic). No instances from the training set were included in the testing set.

The cleaned vector sketches were rasterized at five resolutions based on the long-edge of the canvas: 256, 384, 512, 768, and 1024 pixels. Each of these five sets were generated using the same methodology as the training set (Section 4): random brush styles and widths and a random background texture.

Additionally, we curated a **rough test dataset** with 112 rough raster sketches. We could not run our evaluation metrics on these sketches, since we lack precisely corresponded vector ground truth data. All testing sketches can be seen in the supplemental materials.

Evaluation metrics. As in Yan et al. [2020]'s benchmark, we use the Chamfer distance between a vectorized drawing and the ground truth vector drawing to measure the overall quality. Unlike Yan et al. [2020]'s benchmark, we measure the Chamfer distance directly in the vector domain. (Yan et al. [2020] evaluated cleanup methods that operated entirely in the raster domain.) We also measure the absolute total stroke length difference $|A_{length} - B_{length}|$ between a vectorization A and ground truth B. This is necessary because foldovers or repeated strokes in a vectorization do not contribute to the Chamfer distance.

8.2 Comparison

We compared our vectorization results with three state-of-the-art techniques [Bessmeltsev and Solomon 2018; Mo et al. 2021; Puhachov et al. 2021]. We evaluated each technique on the cleaned test dataset and metrics described above. Example results can be seen in Figures 1 and 17.

Our code was implemented in Python with PyTorch. The post-processing algorithms are particularly unoptimized. We measured execution time on 3 different computer systems. Computer A is a 2019 laptop equipped with a 3.4GHz Intel(R) Core(TM) i7-7700HQ 4-core CPU, 32GB system RAM, and an Nvidia GTX555 GPU with 4GB VRAM. Computer B is a cluster node with a 2.8GHz AMD EPYC 7543 32-Core CPU (only 4-cores usable), 32GB system RAM, and an Nvidia A100 GPU with 40GB VRAM. Computer C is a 2021 MacBook Pro with an M1 Pro and 16GB RAM.

The complete set of input and output images are in the supplemental materials. We evaluated two variants of our model, a full-sized one that need larger VRAM and is compatible with Computer B and C, a reduced-size (basic) variant which shrinks the Distance Field Prediction network[2] to reduce GPU RAM making it feasible to execute on Computer A.

Both models produce similar quality results. All figures were created using the full size model. An overview of the comparative metrics at testing resolution 512 is presented in Tables 1 and 2. The distribution of stroke length difference, Chamfer Distance, and computational runtime are illustrated in Figures 13, 14, and 15,

[2]The basic model reduces 256 channels to 128 for its ResNet module, and 256 channels to 96 for its ResNext modules.

Table 1. Benchmark comparison with clean backgrounds and an unstylized stroke style at 512-pixel resolution. Lower (↓) is better for all metrics).

	Mo [2021]	Bessmeltsev [2019]	Puhachov [2021]	Ours (basic)	Ours (full)
Chamfer Distance	37,760.01	126,873.6	28,180.47	**15,820.81**	**15,413.22**
Stroke Length Diff	658.32	709.42	410.84	**256.75**	**231.30**
Success Rate	100%	99%	98%	100%	100%

Table 2. Benchmark comparison with paper texture and stylized strokes at 512-pixel resolution. Lower (↓) is better for all metrics).

	Mo [2021]	Bessmeltsev [2019]	Puhachov [2021]	Ours (basic)	Ours (full)
Chamfer Distance	243,620.07	186,946.52	66,310.09	**26,505.48**	**19,278.82**
Stroke Length Diff	1102.81	909.69	901.61	**406.23**	**267.57**
Success Rate	100%	99%	98%	100%	100%

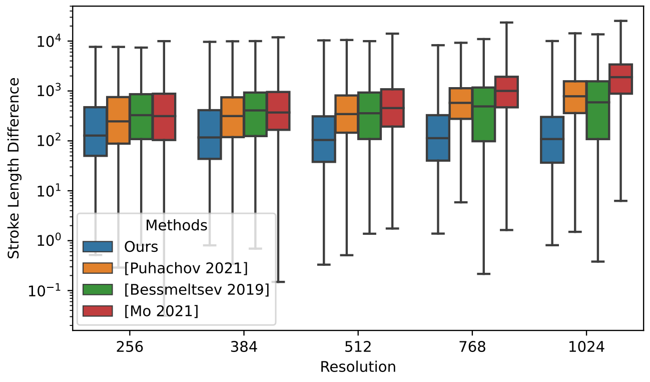

Fig. 13. The distribution of stroke length difference (lower is better) among different methods and image resolutions.

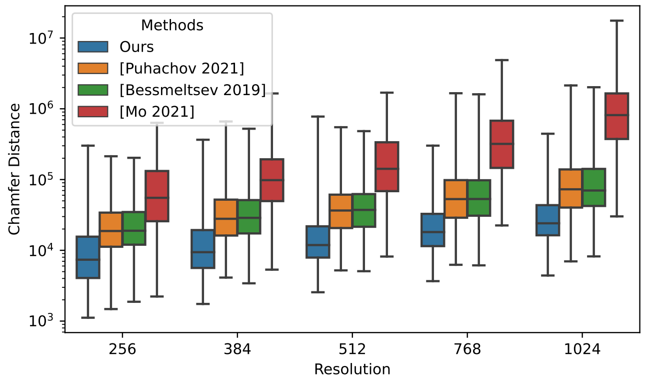

Fig. 14. The distribution of Chamfer distance (lower is better) among different methods and image resolutions.

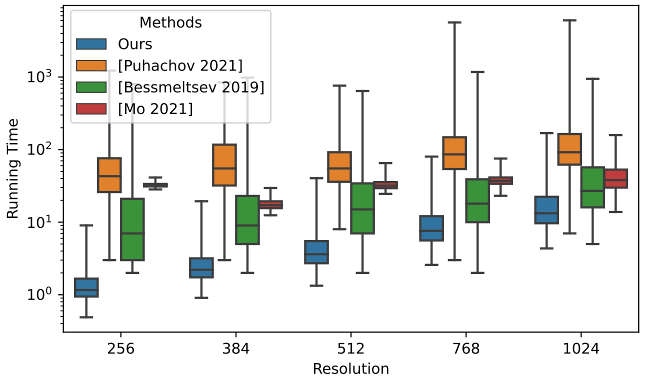

Fig. 15. The distribution of running time in seconds on Computer B. Lower is better.

respectively. Note that we use a logarithmically scaled Y-axis. No outliers were removed for the upper and lower extremes.

Our models reduce Chamfer error by a factor of ≈2–10 over competing approaches. Particularly for complex examples, our approach preserves far more details (Figure 17). Figure 1 shows an example of automatically processing our output with Yin et al. [2022]'s algorithm for finding closed regions in vector sketches.

Our method produced high-quality output for the rough test sketches, which often contain repetitive, messy strokes such as hatching and scaffold lines (Figure 20). Other methods either failed to capture the same level of detail (Figure 18) or failed to vectorize more than 50% of the examples (Table 4). See the supplemental materials for more rough sketch vectorization results. This suggests a promising direction for future research. If we can precisely vectorize every stroke in a sketch, it may be easier to clean messy strokes in the vector domain.

Our algorithm is typically more efficient than competing approaches (Figure 15). In particular, our algorithm contains no lengthy optimization. 61–67% of the total runtime was spent on neural network inference (basic vs. full). For 512-pixel images, our algorithm required, on average, 17, 5, and 10 seconds for processing on Computer A, Computer B, and Computer C, respectively.

Table 3. In our ablation study, we compared the performance on the benchmark dataset between our full model and our model without Multi-Resolution (MR), Sketch Skeleton (SS), Sub-Pixel Sampling (SPS), and Post Processing (PP). Lower (↓) is better for all metrics).

	w/o MR	w/o SS	w/o SPS	w/o PP	Full model
Chamfer Distance	26,388.66	23,770.21	24,115.89	**16,657.89**	19,278.82
Stroke Length Diff	396.54	348.47	404.77	281.7	**267.57**

Table 4. Speed and success rate on our rough test dataset. This dataset lacks ground truth for quality evaluation.

	Mo [2021]	Bessmeltsev [2019]	Puhachov [2021]	Ours (full)
Speed (seconds/image)	**57.6**	182.2	306.5	72.8
Success rate	100%	44.6%	17.9%	100%

8.3 Ablation study

To evaluate the degree of improvement due to different components of our approach, we re-trained our model under 4 different configurations: without the Line Reconstruction network's multi-resolution convolution branches, without its skeleton skeleton loss,

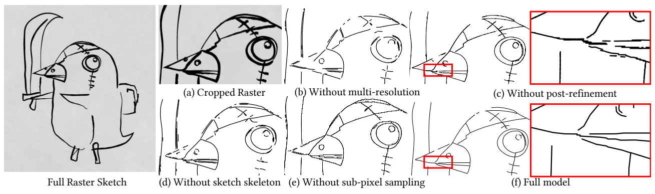
Full Raster Sketch
(a) Cropped Raster
(b) Without multi-resolution
(c) Without post-refinement
(d) Without sketch skeleton
(e) Without sub-pixel sampling
(f) Full model

Fig. 16. Ablation study results. Without multi-resolution and the sketch skeleton loss, our network struggles to correctly extract clean centerlines and output numerous messy parallel lines. Without sub-pixel sampling, our network performs poorly at dual contouring vertex prediction. In our full model, although most of the predicted lines are near the ground truth centerlines, there are still messy predictions (upper red boxes). Our refinement post-processing correctly removes redundant lines and connect broken lines based on the network output. Image ©VFS Digital Design CC-BY-2.0.

without super-resolution prediction, and without post-processing. We evaluated those models' performance on our benchmark dataset. Metrics can be seen in Table 3. Figure 16 shows a visual comparison. Configurations without post-processing often have better (lower) Chamfer Distance and worse (higher) stroke length difference. This is because post-processing eliminates superfluous lines and bridges gaps in the output, which contribute to a reduction in the Chamfer Distance.

8.4 Interactive topology editing

We created a GUI for users to interactively refine the output topology. Our algorithm provides natural handles in the form of the under-sampling map and keypoint positions. This data is output from our networks and used by our light-weight post-processing algorithms. Our interface allows users to update the under-sampling map with lasso and pencil tools and create, move, and delete keypoints. Our interface also displays suggested locations for under-sampled regions. When under the interactive mode, our under-sampling map refinement will process a USM region only if: a) the USM contains keypoints detected by our network, and b) the keypoint type match the truncation number of current USM (e.g, a USM contains 2 truncation and a sharp turn keypoint). Otherwise current refinement process will be skipped and this USM region will be shown as suggested locations waiting user's confirmation. See the supplemental materials for a video demonstration.

9 CONCLUSION

We presented an implicit approach for vectorizing raster line drawings. Our approach leverages and improves upon recent results in image-to-image translation (for distance field prediction) and neural dual contouring. Our network scales to any resolution due to its use of Fully Convolutional Neural Networks (FCN). We address under-sampling with additional predictions and simple yet effective post-processing that supports automatic and interactive topology refinement. Experimental results demonstrate that our method outperforms state-of-the-art techniques in terms of fidelity, allowing far more complex inputs to be faithfully vectorized. Our line reconstruction network's improvements may be beneficial for other neural dual contouring applications.

Limitations and Future Work. In the presence of very thick or noisy lines, or regions with complex topology, our method may fail to resolve individual lines and instead output multiple hatching lines (Figure 19). We believe this is caused by highly ambiguous sketches, as discussed in [Yan et al. 2020]. This means that there are several ideal vectorizations instead of only one correct solution. However, this contracts our paired training strategy. To address this limitation, our GUI supports running a line normalization preprocessing step [Simo-Serra et al. 2018] or a line extraction step [Xiang et al. 2022] to reduce ambiguity. Reducing the input resolution also helps to reduce such ambiguity. However, this may introduce slight distortions in regions with complex topology.

In the future, we believe our approach has great potential for improvement. One straightforward direction could be introducing generative networks into the 0-level curve extraction step to train the vectorization under a one-to-many mapping task, perhaps causing the network to generalize better on highly ambiguous sketches. We can also apply other networks (e.g., [Müller et al. 2022]) which naturally support different sampling rates when representing the unsigned distance field. This could further increase the vectorization accuracy. Additionally, we would like to "vectorize" additional properties such as line thickness and stroke color. We can also consider the stroke tangents and curvature on the boundary of the under-sampled regions to create appropriate curve continuity at junctions. We would also like to explore a hybrid approach that can vectorize solid or shaded regions.

ACKNOWLEDGMENTS

This research was partly conducted during an Adobe internship. This project was supported by resources provided by the Office of Research Computing at George Mason University (URL: https://orc.gmu.edu), which is funded in part by grants from the United States National Science Foundation (Award Number 2018631). Finally, we thank the anonymous reviewers for taking the time to read our paper and provide feedback. Their comments and suggestions ultimately led to a better paper for everyone.

REFERENCES

Bin Bao and Hongbo Fu. 2023. Line Drawing Vectorization via Coarse-to-Fine Curve Network Optimization. *Computer Graphics Forum* n/a, n/a (March 2023). https://doi.org/10.1111/cgf.14787

Mikhail Bessmeltsev and Justin Solomon. 2018. Vectorization of Line Drawings via PolyVector Fields. *arXiv:1801.01922 [cs]* (Sept. 2018). arXiv:1801.01922 [cs]

Mikhail Bessmeltsev and Justin Solomon. 2019. Vectorization of Line Drawings via Polyvector Fields. *ACM Transactions on Graphics* 38, 1 (Jan. 2019), 1–12. https://doi.org/10.1145/3202661

Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 16351–16361.

Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. 2022. Neural Dual Contouring. *ACM Transactions on Graphics* 41, 4 (July 2022), 104:1–104:13. https://doi.org/10.1145/3528223.3530108

Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. 2021. Cloud2Curve: Generation and Vectorization of Parametric Sketches. arXiv:2103.15536 [cs]

Edoardo Alberto Dominici, Nico Schertler, Jonathan Griffin, Shayan Hoshyari, Leonid Sigal, and Alla Sheffer. 2020. PolyFit: Perception-Aligned Vectorization of Raster Clip-Art via Intermediate Polygonal Fitting. *ACM Transactions on Graphics* 39, 4 (Aug. 2020), 77:77:1–77:77:16. https://doi.org/10.1145/3386569.3392401

Luca Donati, Simone Cesano, and Andrea Prati. 2018. A Complete Hand-Drawn Sketch Vectorization Framework. https://doi.org/10.48550/arXiv.1802.05902 arXiv:1802.05902 [cs]

Zheng-Jun Du, Liang-Fu Kang, Jianchao Tan, Yotam Gingold, and Kun Xu. 2023. Image vectorization and editing via linear gradient layer decomposition. *ACM Transactions on Graphics (TOG)* 42, 4 (Aug. 2023).

Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. 2020. Deep Vectorization of Technical Drawings. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Vol. 12358. Springer International Publishing, Cham, 582–598. https://doi.org/10.1007/978-3-030-58601-0_35

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. Simplicity: A Global Approach to Line Drawing Vectorization. *ACM Transactions on Graphics* 35, 4 (July 2016), 120:1–120:10. https://doi.org/10.1145/2897824.2925946

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2017. Photo2clipart: image abstraction and vectorization using layered linear gradients. *ACM Transactions on Graphics* 36, 6 (Dec. 2017), 1–11. https://doi.org/10.1145/3130800.3130888

Songwei Ge, Vedanuj Goswami, C. Lawrence Zitnick, and Devi Parikh. 2021. Creative Sketch Generation. *arXiv:2011.10039 [cs]* (March 2021). arXiv:2011.10039 [cs]

Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Transactions on Graphics* 39, 6 (Nov. 2020), 167:1–167:16. https://doi.org/10.1145/3414685.3417851

Yi Guo, Zhuming Zhang, Chu Han, Wenbo Hu, Chengze Li, and Tien-Tsin Wong. 2019. Deep Line Drawing Vectorization via Line Subdivision and Topology Reconstruction. *Computer Graphics Forum* 38, 7 (Oct. 2019), 81–90. https://doi.org/10.1111/cgf.13818

Olga Gutan, Shreya Hegde, Erick Jimenez Berumen, Mikhail Bessmeltsev, and Edward Chien. 2023. Singularity-Free Frame Fields for Line Drawing Vectorization. *Computer Graphics Forum (CGF)* (2023).

David Ha and Douglas Eck. 2017. A Neural Representation of Sketch Drawings. https://doi.org/10.48550/arXiv.1704.03477 arXiv:1704.03477 [cs, stat]

X. Hilaire and K. Tombre. 2006. Robust and Accurate Vectorization of Line Drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 6 (June 2006), 890–904. https://doi.org/10.1109/TPAMI.2006.127

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual Contouring of Hermite Data. *ACM Transactions on Graphics* 21, 3 (July 2002), 339–346. https://doi.org/10.1145/566654.566586

Matthew Kaplan and Elaine Cohen. 2006. Producing Models From Drawings of Curved Surfaces.. In *SBM*. 51–58.

Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. 2009. Automatic and Topology-Preserving Gradient Mesh Generation for Image Vectorization. *ACM Transactions on Graphics* 28, 3 (July 2009), 85:1–85:8. https://doi.org/10.1145/1531326.1531391

Gregory Lecot and Bruno Levy. 2006. *Ardeco: Automatic Region DEtection and COnversion*. The Eurographics Association. https://doi.org/10.2312/EGWR/EGSR06/349-360

Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: Consolidating Raw Sketches into Artist-Intended Curve Drawings. *ACM Transactions on Graphics* 37, 4 (Aug. 2018), 1–15. https://doi.org/10.1145/3197517.3201314

Hanyuan Liu, Chengze Li, Xueting Liu, and Tien-Tsin Wong. 2022. End-to-End Line Drawing Vectorization. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 4 (June 2022), 4559–4566. https://doi.org/10.1609/aaai.v36i4.20379 Number: 4.

Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A Learned Representation for Scalable Vector Graphics. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Seoul, Korea (South), 7929–7938. https://doi.org/10.1109/ICCV.2019.00802

Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. 2021. General Virtual Sketching Framework for Vector Line Art. *ACM Transactions on Graphics* 40, 4 (July 2021), 51:1–51:14. https://doi.org/10.1145/3450626.3459833

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. 2013. Topology-Driven Vectorization of Clean Line Drawings. *ACM Transactions on Graphics* 32, 1 (Jan. 2013), 1–11. https://doi.org/10.1145/2421636.2421640

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics* 27, 3 (Aug. 2008), 1–8. https://doi.org/10.1145/1360612.1360691

Amal Dev Parakkat, Uday Bondi Pundarikaksha, and Ramanathan Muthuganapathy. 2018. A Delaunay Triangulation Based Approach for Cleaning Rough Sketches. *Computers & Graphics* 74 (Aug. 2018), 171–181. https://doi.org/10.1016/j.cag.2018.05.011

Ivan Puhachov, William Neveu, Edward Chien, and Mikhail Bessmeltsev. 2021. Keypoint-Driven Line Drawing Vectorization via PolyVector Flow. *ACM Transactions on Graphics* 40, 6 (Dec. 2021), 1–17. https://doi.org/10.1145/3478513.3480529

Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J. Mitra. 2021. Im2Vec: Synthesizing Vector Graphics without Vector Supervision. https://doi.org/10.48550/arXiv.2102.02798 arXiv:2102.02798 [cs]

Philip J. Schneider. 1990. Graphics Gems. Chapter An Algorithm for Automatically Fitting Digitized Curves, 612–626.

Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. CrossShade: Shading concept sketches using cross-section curves. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–11. Publisher: ACM New York, NY, USA.

Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. 2018. Real-Time Data-Driven Interactive Rough Sketch Inking. *ACM Transactions on Graphics* 37, 4 (Aug. 2018), 1–14. https://doi.org/10.1145/3197517.3201370

Tibor Stanko, Mikhail Bessmeltsev, David Bommes, and Adrien Bousseau. 2020. Integer-Grid Sketch Simplification and Vectorization. *Computer Graphics Forum* 39, 5 (Aug. 2020), 149–161. https://doi.org/10.1111/cgf.14075

Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W Sumner, Markus Gross, and Jarek Rossignac. 2010. BetweenIt: An interactive tool for tight inbetweening. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 605–614. Issue: 2.

Xiaoyu Xiang, Ding Liu, Xiao Yang, Yiheng Zhu, Xiaohui Shen, and Jan P. Allebach. 2022. Adversarial Open Domain Adaptation for Sketch-to-Photo Synthesis. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Waikoloa, HI, USA, 944–954. https://doi.org/10.1109/WACV51458.2022.00102

Chuan Yan, David Vanderhaeghe, and Yotam Gingold. 2020. A Benchmark for Rough Sketch Cleanup. *ACM Transactions on Graphics* 39, 6 (Nov. 2020), 163:1–163:14. https://doi.org/10.1145/3414685.3417784

Wenwu Yang, Hock-Soon Seah, Quan Chen, Hong-Ze Liew, and Daniel Sýkora. 2018. FTP-SC: Fuzzy Topology Preserving Stroke Correspondence. *Computer Graphics Forum* 37, 8 (2018), 125–135. https://doi.org/10.1111/cgf.13518

Jerry Yin, Chenxi Liu, Rebecca Lin, Nicholas Vining, Helge Rhodin, and Alla Sheffer. 2022. Detecting Viewer-Perceived Intended Vector Sketch Connectivity. *ACM Transactions on Graphics* 41, 4 (July 2022), 87:1–87:11. https://doi.org/10.1145/3528223.3530097

T. Y. Zhang and C. Y. Suen. 1984. A Fast Parallel Algorithm for Thinning Digital Patterns. *Commun. ACM* 27, 3 (March 1984), 236–239. https://doi.org/10.1145/357994.358023

Zibo Zhang, Xueting Liu, Chengze Li, Huisi Wu, and Zhenkun Wen. 2022. Vectorizing Line Drawings of Arbitrary Thickness via Boundary-based Topology Reconstruction. *Computer Graphics Forum* 41, 2 (2022), 433–445. https://doi.org/10.1111/cgf.14485

Shuang Zhao, Fredo Durand, and Changxi Zheng. 2018. Inverse Diffusion Curves Using Shape Optimization. *IEEE Transactions on Visualization and Computer Graphics* 24, 7 (July 2018), 2153–2166. https://doi.org/10.1109/TVCG.2017.2721400

Haikuan Zhu, Juan Cao, Yanyang Xiao, Zhonggui Chen, Zichun Zhong, and Yongjie Jessica Zhang. 2022. TCB-spline-based Image Vectorization. *ACM Transactions on Graphics* 41, 3 (June 2022), 34:1–34:17. https://doi.org/10.1145/3513132

A LINE RECONSTRUCTION NETWORK DETAILS

A diagram of our line reconstruction network architecture can be seen in Figure 22. The Neural Dual Contouring method [Chen et al. 2022] works in 3D. Its training data is generated from 3D meshes which may be skewed towards watertight surfaces. Complex (non-manifold) structures common in 2D line drawing graphics, such as junctions, open curves, etc., may be under-represented. We re-created the 3D mesh training set from our 2D sketch data set for a fair comparison, shown in Figure 21.

B DUAL DOMAIN COORDINATE SHIFT

We have identified a problem related to the conversion of data between the raster and UDF domains. When working with a raster image of size $H \times W$, the corresponding size of the underlying UDF should be $(H+1) \times (W+1)$. Typically, we align the vector paths with the "pixel grid" in the raster domain, as illustrated in Figure 23(a). However, in the UDF domain, there is no sample point that precisely matches the position of each pixel due to a domain shift, as depicted in Figure 23(b). This shift causes a misalignment of coordinates when converting raster pixels to sampled points with distance value in UDF domain, ultimately resulting in sub-optimal vectorization performance.

To address this issue, a special convolutional layer is inserted into the Center Line Prediction network and the following Line Reconstruction network. This layer expands each pixel (raster domain)

Table 5. Comparing our method's precision/recall rate and valence loss with Neural Dual Contouring on clean UDFs (top) and noisy UDFs (bottom). The numbers in parentheses for "Valence" are the (min - max) loss values (lower is better).

Clean

	Ours	[Chen et al. 2022]
Precision	**97.7%**	96.8%
Recall	**95.5%**	93.7%
Valence error	**950 (257 - 3808)**	989 (263 - 3885)

Noisy

	Ours	[Chen et al. 2022]
Precision	**95.1%**	81.5%
Recall	**90.3%**	63.1%
Valence error	**1069** (279 - 4246)	1140 (**251** - **3781**)

into a grid with four corners, where each corner represents one sampled distance value (UDF domain), or merges every four distance values in one grid into one edge flag. This convolutional layer is designed with a kernel size of 2 and performs the expanding or merging operation as described above by setting the padding size to 1 or using zero padding, respectively. These modifications calibrate the domains, as shown in Figure 23, ensuring accurate preservation of the coordinate information in the final vector paths.

C DUAL CONTOURING COMPARISON

We trained our line reconstruction (2D NDC) network and the original 3D NDC network using the same training set, we created the 2D and 3D training data based on the same vector sketch, respectively. Subsequently, we evaluated the trained models on the rough sketch benchmark test set, which was distinct from the training set.

The evaluation metrics employed were average edge flag precision/recall and valence loss, as depicted in the presented table. Notably, when provided with precise UDFs, both the 2D and 3D NDC networks exhibited exceptional accuracy in reconstructing line segments. Furthermore, the visual similarity between the reconstructed segments was remarkable. In order to capture the quality of topology reconstruction more comprehensively, we also computed the valence loss, which could be also used to evaluate the reconstructed topology quality. The results (Table 5) demonstrated that our method consistently outperforms the 3D NDC approach in terms of both precision/recall rates and valence loss.

While the performance of the 3D NDC network closely matches that of our 2D NDC network on clean UDF input, our method is significantly better at dealing with noise. We introduced Gaussian noise with a mean of zero and a standard deviation of 1% of the current UDF's standard deviation. Table 5 clearly demonstrates the superiority of our approach over the 3D NDC network in handling noisy input. This robustness to noise is crucial in our task, given that the imperfect nature of the UDFs predicted by our distance field extraction network.

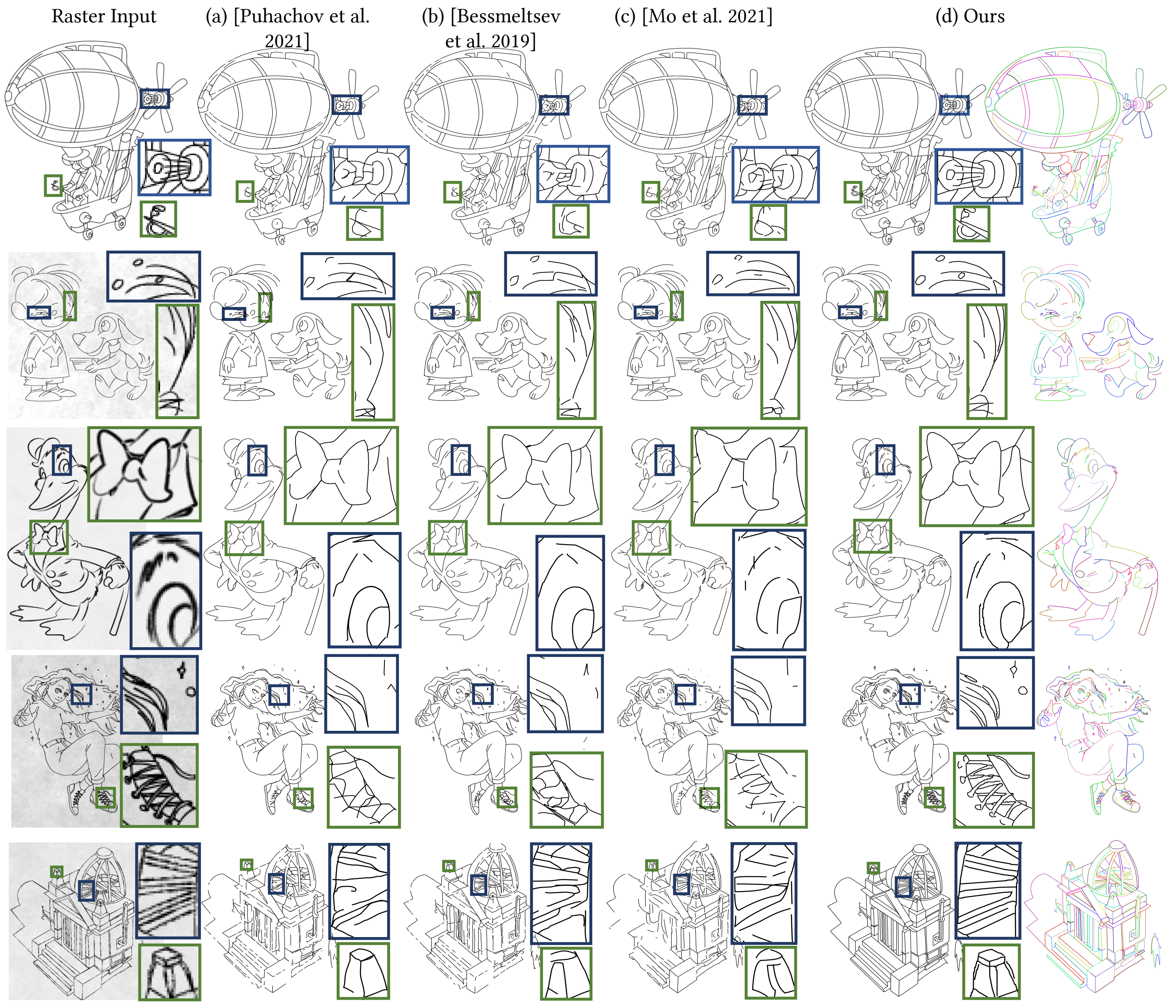

Fig. 17. A comparison of vectorization results on complex examples from Yan et al. [2020]. Please zoom in to inspect the visual differences between methods. Outputs for the entire benchmark can be seen in the supplemental materials. From top row to bottom: air ship ©David Revoy, CC-BY-4.0; boy&dog and duck images ©Preston Blair, explicit permission; girl ©Maria Fiddler, CC-BY-NC-SA-4.0; building ©Tinyhouse University, CC-BY-SA.

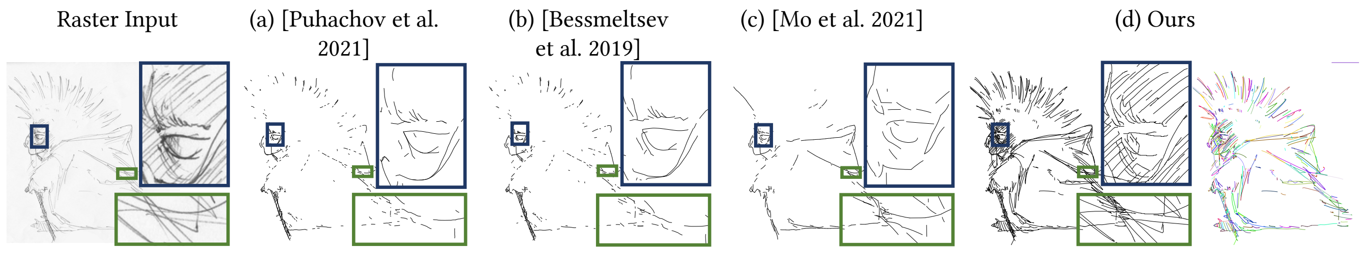

Fig. 18. A comparison of vectorization results on rough sketches from Yan et al. [2020]. Please zoom in to inspect the visual differences between methods. Outputs for the entire benchmark can be seen in the supplemental materials. Image ©AP, CC-BY-SA-3.0.

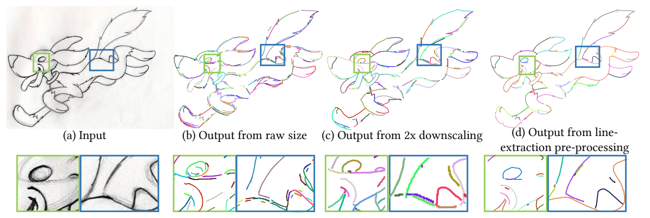
(a) Input
(b) Output from raw size
(c) Output from 2x downscaling
(d) Output from line-extraction pre-processing

Fig. 19. A failure case. Our network will fail on thick strokes or densely repeated strokes, shown in the green and blue boxes. Those regions usually have high ambiguity and it is difficult to find the only correct centerline position. Although this issue could be alleviated to a certain extent by downscale the input sketch, or applying line extraction pre-processing as [Xiang et al. 2022], we believe the pairwise training strategy (one sketch mapping to only one version of correct vector lines) is the fundamental limitation of our network. Dog image ©Ivan Huska.

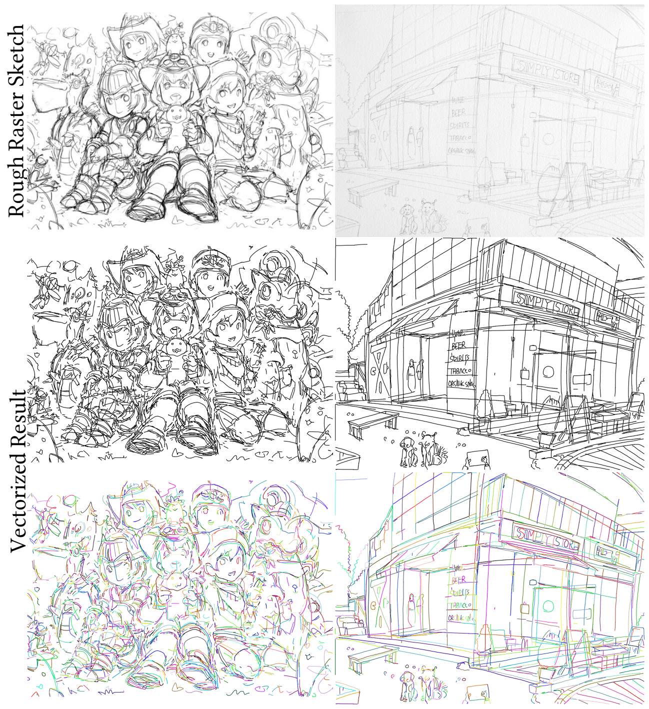

Fig. 20. More rough sketch vectorization examples. Left image ©Krenz Cushart, CC-BY-NC-4.0. Right image ©Jinho Jung, CC-BY-SA-2.0.

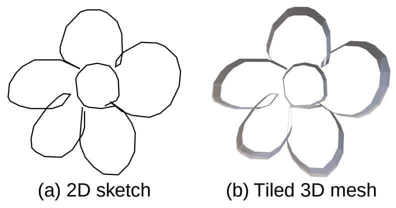

Fig. 21. Example of the training data used for [Chen et al. 2022] training. We tiled the 2D vector sketch (a) along its Z-axis and generate a 3D mesh (b) then generate UDFs from them as input.

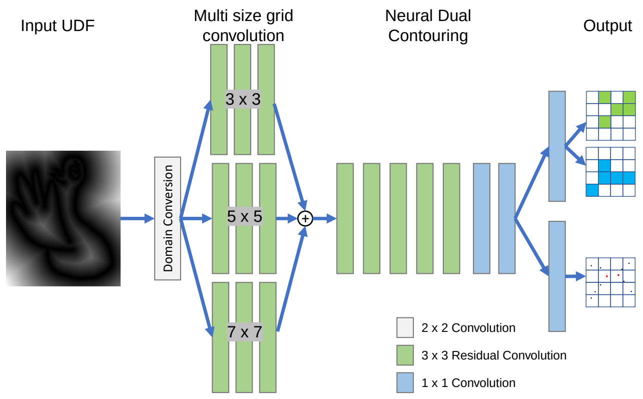

Fig. 22. Line Reconstruction Network Structure

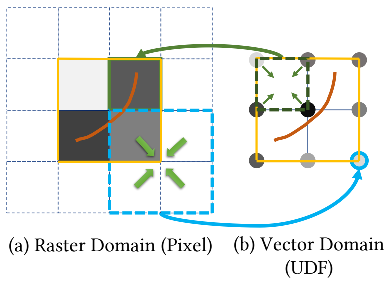

Fig. 23. The coordinate shift between the raster and vector domain. We designed our UDF network to output data in the dual domain, so that the second network, dual contouring, would output data back in the primary domain. The input raster data (a) has values in the center of each square visualized square, which is our primary domain. The UDF network (lower blue arrow) outputs distance values in the dual domain (b), at the corners of each square. The NDC network (upper green arrow) outputs points in the primary domain (squares). It takes the UDF network's dual domain data as input and outputs in the dual of the dual domain, which is the primary domain.