October 2022

Controllable Neural Synthesis for Natural Images and Vector Art

Difan Liu
University of Massachusetts Amherst

CONTROLLABLE NEURAL SYNTHESIS FOR NATURAL IMAGES AND VECTOR ART

A Dissertation Presented

by

DIFAN LIU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2022

Manning College of Information and Computer Sciences

CONTROLLABLE NEURAL SYNTHESIS FOR NATURAL IMAGES AND VECTOR ART

A Dissertation Presented

by

DIFAN LIU

Approved as to style and content by:

Evangelos Kalogerakis, Chair

Subhransu Maji, Member

Mohit Iyyer, Member

Aaron Hertzmann, Member

James Allan, Chair of the Faculty
Manning College of Information and Computer Sciences

ACKNOWLEDGMENTS

First and foremost, I would like to thank my dear advisor, Prof. Evangelos Kalogerakis, for his guidance, patience, and encouragement during my Ph.D. study. His professional guidance and immense knowledge helped me in all the time of my academic research and daily life. He spent countless hours with me patiently discussing ideas, reviewing results, and refining presentations and papers to minute detail. In the first couple of years, I had little background knowledge in both computer graphics and computer vision. Vangelis patiently waited for me to grow up and supported me academically, financially, and mentally. His nurture and care have made me a better researcher, colleague, friend, and human being. Any amount of future success that I will have in my life, Vangelis's teachings will have played a major role behind that.

Secondly, I would like to thank Aaron Hertzmann for mentoring me during my first two projects. During the collaboration with him, I had great freedom of the research topics and had the opportunity to investigate my passion for non-photorealistic rendering. He has professional experience in these directions and gave me strong guidance and constructive feedback. I would also like to thank my other colleagues and project partners at Adobe Research. I thank Matthew Fisher and Oliver Wang who helped manage my internship experience and offered me the great opportunity of work position at Adobe. I thank Tobias Hinz who gave me great advice on generative models and paper writing. I would also like to thank Richard Zhang, Taesung Park, Michaël Gharbi for their valuable guidance and encouragement. Without their precious support, it would not be possible to conduct this thesis.

Thirdly, I would like to thank my collaborators Gopal Sharma, Li Yi, Mohamed Nabail and Sandesh Shetty. Gopal and I worked on exciting projects towards better

shape decomposition. Prof. Li Yi is an expert in the field of 3D perception and shape analysis. It is always fun and inspiring to chat and collaborate with him. I would also like to thank Mohamed and Sandesh for their kind help and great contribution to my research projects.

Besides my collaborators, I would like to thank the rest of my thesis committee: Prof. Subhransu Maji and Prof. Mohit Iyyer for their insightful comments and encouragement which helped me widen my research from various perspectives.

I am glad to be part of the Computer Graphics and Vision lab at UMass Amherst, where I met a lot of energetic and eager young minds. I would like to thank my fellow lab-mates: Zhan Xu, Yang Zhou, Gopal Sharma, Pratheba Selvaraju, Dmitrii Petrov, Zezhou Cheng, Chenyun Wu, Matheus Gadelha, Tsung-Yu Lin, Jong-Chyi Su, Hang Su, Huaizu Jiang, Aruni RoyChowdhury, Zitian Chen, SouYoung Jin and Ashish Singh. Also I thank my friends Pengshan Cai, Dongxu Zhang, Zhichao Yang, Xiang Li, Zhipeng Tang, Mengxue Zhang, Zhiqi Huang, Puxuan Yu and Shufan Wang. They made my life at Amherst much more joyful.

Finally, many thanks to my parents and family for their continued support, encouragement, and unconditional love. It would be impossible for me to finish my Ph.D. study without their encouragement and love from the other side of the ocean.

ABSTRACT

CONTROLLABLE NEURAL SYNTHESIS FOR NATURAL IMAGES AND VECTOR ART

SEPTEMBER 2022

DIFAN LIU

B.E., UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Evangelos Kalogerakis

Neural image synthesis approaches have become increasingly popular over the last years due to their ability to generate photorealistic images useful for several applications, such as digital entertainment, mixed reality, synthetic dataset creation, computer art, to name a few. Despite the progress over the last years, current approaches lack two important aspects: (a) they often fail to capture long-range interactions in the image, and as a result, they fail to generate scenes with complex dependencies between their different objects or parts. (b) they often ignore the underlying 3D geometry of the shape/scene in the image, and as a result, they frequently lose coherency and details.

My thesis proposes novel solutions to the above problems. First, I propose a neural transformer architecture that captures long-range interactions and context for image synthesis at high resolutions, leading to synthesizing interesting phenomena in

scenes, such as reflections of landscapes onto water or flora consistent with the rest of the landscape, that was not possible to generate reliably with previous ConvNet- and other transformer-based approaches. The key idea of the architecture is to sparsify the transformer's attention matrix at high resolutions, guided by dense attention extracted at lower image resolution. I present qualitative and quantitative results, along with user studies, demonstrating the effectiveness of the method, and its superiority compared to the state-of-the-art. Second, I propose a method that generates artistic images with the guidance of input 3D shapes. In contrast to previous methods, the use of a geometric representation of 3D shape enables the synthesis of more precise stylized drawings with fewer artifacts. My method outputs the synthesized images in a vector representation, enabling richer downstream analysis or editing in interactive applications. I also show that the method produces substantially better results than existing image-based methods, in terms of predicting artists' drawings and in user evaluation of results.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Visual content creation has a long history. It was manifested as cave art during prehistoric ages, while throughout the mankind history, it is incarnated in art paintings, drawings, sculptures, crafts, and architecture. More recently, it is demonstrated in photography, filmmaking, and computer-generated imagery. Creating compelling and expressive visual content has traditionally been the domain of experienced artists. For casual users, attempts at creating or editing visual content end up quickly "falling off" the manifold of aesthetically plausible images. With recent advances in deep learning, neural image synthesis [38, 61, 161] has provided a learning-based alternative able to assist non-experts to synthesize or manipulate plausible imagery while expressing their creativity. Neural networks have been particularly effective in removing unwanted objects and inpainting in photographs, changing the season of landscape images, and turning photographs into plausible artwork or vice-versa [147, 105, 9]. To achieve more controllable generation, a popular line of research focuses on generating and editing images conditioned on certain guidance. Visual clues such as sketches [64] and semantic maps [99] have been widely used as additional input, which enable users to guide the process of image generation through easy-to-use operations.

Despite recent advances, there are still open problems in controllable image generation. First, 2D neural image synthesis often lack an understanding of the 3D world and the image formation process. Explicit guidance in the form of a 3D representation [163] can provide more precise control over camera viewpoint or object pose, and can effectively bridge the 2D image space and the 3D physical world. Second, there

are often prohibitive computational costs of neural image synthesis, which prevents controllable image synthesis at high resolutions. More specifically, previous algorithms fail to generate scenes with complex dependencies at high resolutions, such as the phenomena of water reflection in landscape images. Third, the synthesis of non-photorealistic images is less explored than photorealistic images. More importantly, the underlying 3D geometry of the artistic images is often ignored. Without the understanding of the underlying 3D world, previous methods generally produce artistic images that frequently lose detail and do not capture artists' drawing styles.

The main focus of this thesis is to address the above challenges to produce high-quality photorealistic and non-photorealistic images. I investigate several data-driven approaches with different guidance signals including segmentation maps and 3D shapes. The presented methods not only help users easily synthesize more visually appealing images but also enable new visual effects not possible before this work.

1.1 Natural Image Synthesis Guided by Semantic Maps

I first propose a method, called ASSET, to address the problem of generating scenes with complex dependencies at high resolution. My method allows users to easily edit a given image by modifying a corresponding segmentation map. To obtain realistic and consistent results, an effective system needs to consider global context from across the full image. For example, to properly hallucinate a reflection in the water on the bottom of the image, the model should consider the content from the top. Traditional CNN based approaches [61, 14, 104, 99, 164] rely entirely on convolutional layers which have difficulty modeling such long-range dependencies [137].

Transformers are well equipped to handle these long-range dependencies through their attention mechanism allowing them to focus on distant image areas at each sampling step. However, the heavy computational cost for using attention, which usually increases quadratically with the input size, makes it infeasible to use standard

transformers for high-resolution image editing. One way to address this is to use a sliding-window approach [29], in which the transformer only attends to a small area around the currently sampled token, thereby reducing the computational cost to a fixed budget. While this approach enables the synthesis of high-resolution images, it forgoes the benefit of modeling long-range dependencies. This leads to inconsistencies when edits are dependent on image regions that are far away in pixel space.

I introduce a novel attention mechanism, called *Sparsified Guided Attention* (SGA), to facilitate long-range image consistency at high resolutions. While the sliding window approach is limited to local contexts, SGA can attend to far contexts that are relevant for the current sampling location. The core idea is to efficiently determine a small list of relevant locations that are worth attending to, and compute the attention map only over these locations. To achieve this, I use a guiding transformer that operates at the downsampled version of the input image and performs the same edit, but enjoys the full self-attention map thanks to the reduced input size. Based on the guiding transformer's attention map, I rank the importance of different areas of the image, and have the high resolution transformer attend only to the top-K most important image regions. In practice, the SGA leads to a large reduction in computational cost due to the obtained sparse attention matrix. Compared to other approaches, my method obtains more realistic and consistent edits while still achieving high diversity in the outputs.

The model takes as input a quantized representation of the image and its edited segmentation map, both obtained through a modified VQGAN encoder [29]. I then mask out all image tokens in the image representation corresponding to the edited area and replace those tokens with a specific `[MASK]` token. The transformer then samples new image tokens at the edited areas, conditioned on the original (masked) image and the edited segmentation map. Finally, a VQGAN decoder is used to decode the image tokens into the final RGB image. As the edited tokens are sampled autoregressively

based on a likelihood-based model, I can sample a diverse set of image outputs, all of which are consistent with the overall image characteristics.

I highlight the following contributions:

- I propose a transformer-based model that outputs realistic and diverse edits specified through modified segmentation maps.
- I introduce *Sparsified Guided Attention* (SGA), which allows the transformer to only attend to the most important image locations, leading to sparse attention matrices and reduced computational cost.
- The model achieves diverse, realistic, and consistent image edits even at 1024×1024 resolution.

1.2 Vector Art Synthesis Guided by 3D Shapes

The second part of this thesis is an algorithm to produce stylized line drawings with the guidance of 3D shapes. Understanding and creating stylized outline drawings is a key task for stylization [85, 6], sketch understanding [42], and human vision [21, 49]. Artists and amateurs alike draw pictures of 3D objects in many different styles, whether for art, animation, architectural design, 3D authoring, or simply the pleasure of drawing. However, most recent research in image stylization does not take 3D geometry into account, producing drawings that frequently lose detail and do not capture image outlines. Conversely, there is a long history of 3D drawing algorithms that create precise line drawings in hand-authored procedural styles, but they cannot be learned from data, making them difficult to control and inapplicable for analysis of existing sketches. While there is a long literature on analysis and shape reconstruction from sketches, these methods typically assume that artists draw with plain line styles.

I propose a two-stage approach for line drawing synthesis and stylization from 3D shapes. In the first stage, the model implements geometric line drawing based on suggestive contours, apparent ridges, ridges, and valleys. Existing geometric line drawing approaches employ hard-to-tune user-specified parameters that need to be determined separately for each object. In contrast, my method learns to automatically select these parameters through a differentiable module. In the second stage, the model converts the plain line drawing synthesized in the first stage into stylized strokes with the guidance of 3D shapes.

There are several challenges in making such a system work. First, classic geometric lines are not readily differentiable with respect to their parameters. I combine soft thresholding functions along with image-space rendered maps of differential surface properties. Second, to generate stylized strokes, the method must disentangle several stroke attributes, including spatially-varying thickness, geometric deformations and smoothness, and texture. These elements may often be quite noisy, with strokes being wiggly or messy; the final pixel values are an entangled combination of these factors. I propose a differentiable rendering formulation of stroke attributes. This allows the model to learn to accurately predict stroke thickness, deformation, and texture. Moreover, it's difficult to collect a large training set of skilled artistic drawings from artists. I describe a crowdsourcing approach to gather data using unskilled crowdworkers for ranking evaluations and a training procedure designed to work with a single training example alone.

I highlight the following contributions for this part of the work:

- I propose a ranking module trained to assess the plausibility of the line drawing, which can be used to drive the optimization of geometric lines.
- I introduce a convolutional network operating along parameterized stroke paths.

- I combine 3D geometry, 2D image, and 1D curve feature maps to learn stroke properties with a differentiable vector renderer.
- I propose a training procedure that enables learning from a single training example.
- My method significantly improves over the existing line drawing synthesis methods and the generated drawings are comparable to artists' drawings.

1.3 Summary of Publications

The list of works that are part of this thesis:

- The content of Chapter 3:
  - [86] Difan Liu, Sandesh Shetty, Tobias Hinz, Matthew Fisher, Richard Zhang, Taesung Park, Evangelos Kalogerakis. ASSET: Autoregressive Semantic Scene Editing with Transformers at High Resolutions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2022.
- The content of Chapter 4:
  - [85] Difan Liu, Mohamed Nabail, Aaron Hertzmann, Evangelos Kalogerakis. Neural Contours: Learning to Draw Lines from 3D Shapes. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.
  - [84] Difan Liu, Matthew Fisher, Aaron Hertzmann, Evangelos Kalogerakis. Neural Strokes: Stylized Line Drawing of 3D Shapes. *International Conference on Computer Vision*, 2021.

The list of publications I co-authored and are not part of this thesis:

- [121] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, Radomir Mech. ParSeNet: A Parametric Surface Fitting Network for 3D Point Clouds. *European Conference on Computer Vision*, 2020.

- [146] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, Leonidas Guibas. Deep Part Induction from Articulated Object Pairs. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 2018.

- [120] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, Subhransu Maji. Neural Shape Parsers for Constructive Solid Geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

- [119] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, Subhransu Maji. CSGNet: Neural Shape Parser for Constructive Solid Geometry. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we discuss the most relevant prior works. In Section 2.1 we review literature on semantic scene editing. In Section 2.2 we discuss prior methods of line drawing synthesis and stylization from 3D shapes.

2.1 Natural Image Synthesis Guided by Semantic Maps

Our work is related to prior work in CNN-based image editing, transformers for image synthesis, efficient transformers, as well as image synthesis with a guidance image, discussed below.

CNN-based image editing. CNN-based methods have achieved impressive results by enabling users to move bounding boxes containing objects [58, 54], modifying scene representations [26, 123], or following textual instructions [98, 15, 107]. Other approaches enable user guides with edges or color information [64, 89] or perform simple inpainting, typically without user guidance [87, 147, 148, 142, 124, 88]. Exemplar-based image translation methods [152, 160, 157] can synthesize images from semantic maps, but they cannot hallucinate new content that does not exist in the exemplar image. Other approaches fine-tune or train a generator for a specific image to perform editing on that single image [3, 117, 53, 132]. However, these models need to be adapted for each new image. More similarly to us, other methods allow for direct editing via segmentation maps [44, 77, 99, 83]. However, these approaches can only generate a single output for a given edit. In addition, previous CNN-based

approaches prioritize local interactions between image pixels for image synthesis due to their inductive bias. They also fail to effectively capture long-range interactions between image regions necessary for realistic image synthesis. Our approach is based on a transformer that is able to effectively capture such interactions and also allows the synthesis of diverse results for each edit.

Transformers for image synthesis. To apply transformers for image synthesis, they are trained on discrete sequences of image elements. Some models first learn a discrete image region representation [110, 29], whereas other approaches work directly on pixels [106, 16, 13, 63]. However, most of them model images in a row-major format, and thus cannot capture bidirectional context, leading to inconsistent editing results. PixelTransformer [128], iLAT [8], and ImageBART [28] add bidirectional context to their transformer models but do not support editing via segmentation maps. More importantly, due to their quadratic complexity in the number of image tokens, these methods are trained on small image resolutions of 256×256 pixels. Alternatively, some approaches model the image directly at a low resolution (e.g., 32×32) and then use a deterministic upsampling network [133, 150]. In this case, fine-grained edits are difficult to achieve due to the small resolution at which the images are modeled. For high-resolution image synthesis, [29, 28] proposed transformers with attention constrained on sliding windows. However, this hampers long-range interactions and, as a result, they often generate artifacts and inconsistent image edits. In contrast, our work incorporates a novel sparsified attention mechanism that can capture such interactions without compromising the synthesized image plausibility.

Diffusion models for image synthesis. Recent research on diffusion models [55] has made significant progress in generating high-fidelity and diverse images. SDEdit [97] "hijacks" the generative process of diffusion models for the task of image editing with colored strokes. ILVR [17] guides the generative process in DDPM [55] to sample

images from various sets directed by a reference image, which leads to applications such as paint-to-image and image editing with scribbles. These methods enable user guidance with edge and color information. However, it is not straightforward to adapt them for semantic image editing tasks. Very recently, text-to-image models such as DALL-E 2 [109] and Imagen [114] produce very high quality results by leveraging diffusion models and employing large-scale computation. These methods capture long-range dependencies at high-resolution by using dense attention at low-resolution and upsampling low-resolution results with attention-free diffusion models. Diffusion models have seen wide success in image generation, yet there are still many exciting directions for future work. For example, as opposed to raster images, diffusion models for vector image synthesis remain largely unexplored. An effective diffusion model operating in curve space can lead to various applications for vector graphics.

Efficient transformers. Much work has been invested in reducing the computational cost of the transformer's attention mechanism [126]. Broadly speaking, there are two ways to achieve this. One way is to reduce the computational cost of the attention mechanism directly, e.g., by approximating full attention through mechanisms where the computation cost grows linearly with the input length [73, 134]. Alternatively, several works explore reducing the cost by replacing full attention with a sparse variant [4, 151]. A few recent vision transformers reduce the computational complexity by spatially reducing attention [91, 136, 153, 143]. However, these methods use encoder-only transformers for feature extraction and do not support autoregressive image generation. Our method is inspired by BigBird's sparse attention mechanism for long-document NLP tasks [151]. BigBird achieves high efficiency by using a random sparse attention map over blocks of tokens. However, when applying the random attention mechanism of BigBird to our task it fails to capture correct context for a given edit. Instead of randomly choosing tokens, our approach picks the most relevant tokens for attention at each spatial location.

Image synthesis with a guidance image. Synthesizing high resolution outputs is challenging in terms of both quality and computational cost. The idea of utilizing a high-resolution guide to upsample a low-resolution output has been explored in computer graphics [76, 11]. In particular, constructing a high-resolution depth map from coarse sensor data, guided by an RGB image has been extensively investigated [145, 102, 31, 90, 144]. More recently, learning-based approaches were developed for similar tasks, by posing it as an image-to-image translation problem [95], fusing the guidance and low-res information at multiple scales [60], or transferring the mapping learned at low resolution to high resolution [122]. While these works primarily aim at leveraging high-resolution information in the input as a guide, our application must synthesize information from a flat input. In fact, our guide is a low-resolution version of the same image. Relatedly, [118] use a low-resolution network to predict parameters of a lightweight high-resolution network, for the purpose of fast image translation, using a convolutional network architecture.

2.2 Vector Art Synthesis Guided by 3D Shapes

How do artists create line drawings, and how do people perceive them? This question has been studied in art history [37], philosophy [39], neuroscience [116], and perceptual psychology [49, 71, 75]. Occluding contour algorithms are the foundation of non-photorealistic 3D computer graphics; see [6] for a survey.

Generalizations of occluding contours improved line drawing algorithms, beginning with the suggestive contours [24, 78], and continuing with Apparent Ridges [68] and several other methods; see [23] for a survey of contour generalizations. Cole *et al.* [20] performed a thorough study, enlisting human artists to create line drawings of known 3D objects. They found that existing methods could account for the majority of human-drawn lines, but many differences remained between hand-drawn and computer-generated lines. Gryaditskaya et al. [43] collect and analyze professional

illustrations of objects. While these analyses yield deep insights into the nature of hand-drawn lines, the synthesis algorithms fail to match the quality of hand-drawn lines, while also requiring several parameters to be set by a user on a case-by-case basis.

Meanwhile, learned image stylization algorithms in computer vision and computer graphics have shown the potential of learning to capture artistic styles such as line drawings and paintings. The first such methods, Image Analogies [50] and Neural Style Transfer [36], used only single-image style exemplars. Many variants of Neural Style Transfer use Gram-matrix-like losses for training or optimization from single examples, e.g., [65, 59, 82]. Other recent approaches learn stylization from larger collections of paired [61, 81] or unpaired examples [162, 103]. All of these methods take only images for input and output. However, these methods lose important geometric information, often resulting in inaccurate portrayal of shape, such as broken outlines. Moreover, these methods do not produce vector output, limiting their usefulness for certain applications.

Stylized rendering of 3D shapes has a long history in Non-Photorealistic Rendering (NPR) research [6], and these algorithms have been used in numerous applications, including movies [22], and video games [127]. Most methods entail hand-designed procedural stylization, e.g., [41, 140, 141, 70]. None of these methods can learn stylization from examples, making authoring and definition of styles challenging. Moreover, none of these methods are differentiable, making them unsuitable for integration with other vision tasks, such as sketch analysis and interpretation.

A few previous methods learn stylized 3D rendering. Bénard *et al.*[5] and StyLit [33] extend Image Analogies to stylize 3D models and animation. In contrast, our method produces vector rather than raster output, which is a more interpretable and useful representation.

Our work builds on ideas from learning vector strokes and stylization. Most existing methods for example-based stroke stylization [51, 94, 93, 69] are not differentiable and require vector training data. More recent methods define differentiable strokes for painting and vector graphics [34, 80], though these methods do not support texture synthesis. Our work is perhaps most similar to SketchPatch [32]. SketchPatch is an image-to-image model that translates a plain sketch to a textured sketch. However, SketchPatch does not take 3D shape and stroke geometry into consideration, and its output is a raster image, and as a result, the method often loses detail from the input geometry.

CHAPTER 3

NATURAL IMAGE SYNTHESIS GUIDED BY SEMANTIC MAPS

The first part of this thesis discusses ASSET, a neural architecture for automatically modifying an input high-resolution image according to a user's edits on its semantic segmentation map [86] [1]. Our architecture is based on a transformer with a novel attention mechanism. Our key idea is to sparsify the transformer's attention matrix at high resolutions, guided by dense attention extracted at lower image resolutions. While previous attention mechanisms are computationally too expensive for handling high-resolution images or are overly constrained within specific image regions hampering long-range interactions, our novel attention mechanism is both computationally efficient and effective. Our sparsified attention mechanism is able to capture long-range interactions and context, leading to synthesizing interesting phenomena in scenes, such as reflections of landscapes onto water or flora consistent with the rest of the landscape, that were not possible to generate reliably with previous convnets and transformer approaches. We present qualitative and quantitative results, along with user studies, demonstrating the effectiveness of our method.

3.1 Method

Overview. Our method synthesizes images guided by user input in the form of an edited label map ("semantic map") of an input image. More specifically, given an

[1] This work is published at the ACM Transactions on Graphics, Vol. 41, No. 4, 2022, and was also presented in the Proceedings of ACM SIGGRAPH 2022.

Figure 3.1: ASSET allows users to create diverse editing results by specifying a region and a new label on the input image. Our efficient transformer captures long-range dependencies in the image, such as the detailed reflection of the trees on the water, even at high resolutions (1024×1024 pixels in this example).

RGB image and its corresponding label map, the user paints some desired changes on the label map, e.g., replace mountain regions with water (Figure 3.2). Since there exist several possible output images reflecting the input edits, our method generates a diverse set of outputs allowing the user to select the most preferable one. Moreover, our method generates high-resolution images of up to 1024×1024 resolution.

Our architecture is shown in Figure 3.2. Inspired by recent approaches [29] we represent images and label maps as a spatial collection of quantized codebook entries (Section 3.1.1). These codebook entries are processed by a transformer model which aims to update the codebook entries of the edited areas in an autoregressive manner (Section 3.1.2). All codebook entries are subsequently decoded to the output set of images (Section 3.1.3). A crucial component of the transformer is its attention mechanism, which enables long-range interaction between different parts of the image such that the synthesized output is coherent as a whole. E.g., if a lake is generated by the semantic edits it must also capture any reflections of landscape (Figure 3.1). One complication is that the quadratic complexity of the traditional attention mechanism

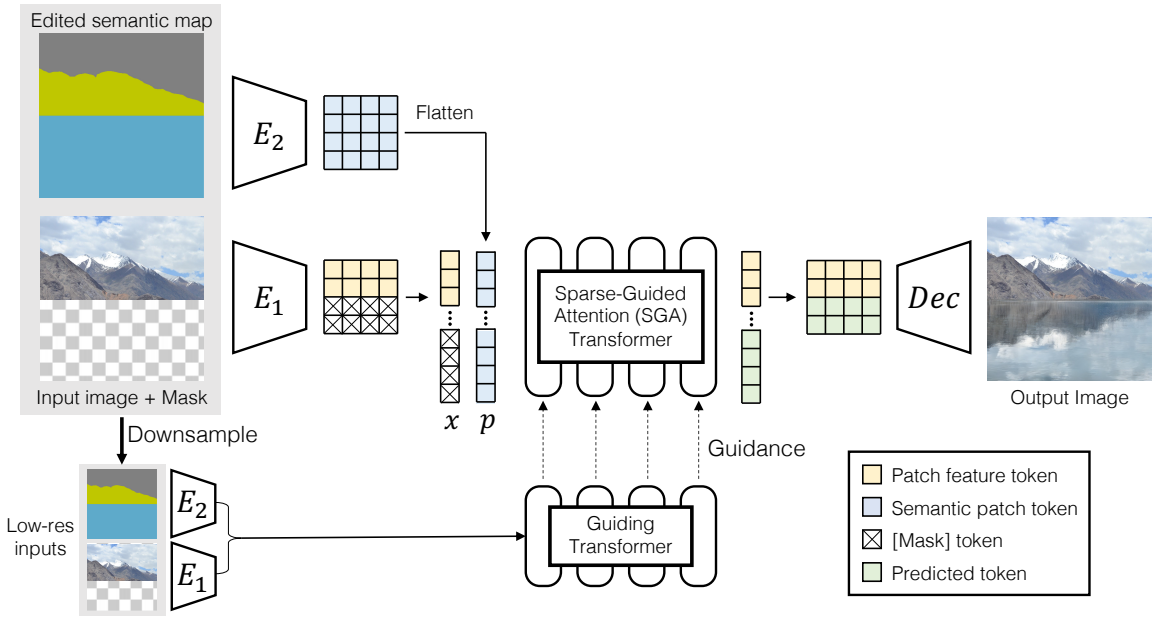

Figure 3.2: Overview of our semantic image editing model. Our transformer model operates in the codebook space using the encoder E_1. To incorporate user edits the tokens inside the edited region are masked and are augmented with a semantic encoder E_2. The mask tokens are filled in by our SGA-Transformer Encoder-Decoder network, whose sparse attention mechanism is guided by the Guiding Transformer that computes the full attention on downsampled inputs. Finally, the generated tokens are decoded into the output image via *Dec*.

leads to a large time and memory cost for high-resolution images. The key idea of our method is to compute the full attention at lower resolution first, and then use that as guidance for sparsifying the attention at full resolution (Section 3.1.2). This approach allows us to model long-range dependencies even at high resolutions, resulting in more coherent and plausible output images compared to existing approaches that constrain attention within sliding windows [29] or alternative attention models [151].

3.1.1 Image encoder

The input RGB image $\mathbf{X}$ of size $H_{\text{im}} \times W_{\text{im}} \times 3$ is processed by a convolutional encoder resulting in a feature map $\mathbf{F}$ of size $\frac{H_{\text{im}}}{16} \times \frac{W_{\text{im}}}{16} \times d$. We also create a $H_{\text{im}} \times W_{\text{im}}$ binary mask indicating image regions that must be replaced according to the semantic map edits. Masked image regions should not affect features produced for unmasked

regions, e.g., information about the edited area of Figure 3.2 should not "leak" into the feature map of the unmasked area. To avoid information leakage, we employ partial convolutions [87] and region normalization [149] in our encoder while processing the unmasked regions. The feature map $\mathbf{F}$ is subsequently quantized following VQGAN [29] with the help of a learned codebook $\mathcal{Z}$, i.e., each feature map entry $\mathbf{f}_{i,j}$ at position (i, j) in $\mathbf{F}$ is mapped to the closest codebook entry $\hat{\mathbf{f}}_{i,j} = \arg\min_{\mathbf{z}_\kappa \in \mathcal{Z}} ||\mathbf{f}_{i,j} - \mathbf{z}_\kappa||$, where $\{\mathbf{z}_\kappa\}_{\kappa=1}^{|\mathcal{Z}|}$ are codebook entries with dimensionality d. The codebook indices of the edited regions, as indicated by the binary mask, are replaced with a special `[MASK]` token (Figure 3.2). We use a second encoder with regular convolutions to obtain a feature representation of the edited semantic map $\mathbf{P}$ which is subsequently quantized in the same way as the RGB image, resulting in codebook entries $\hat{\mathbf{g}}_{i,j}$.

3.1.2 Autoregressive transformer

Our transformer follows a sequence-to-sequence architecture inspired by [131] which consists of a bidirectional encoder and an autoregressive decoder, both of which are equipped with our novel sparsified attention mechanism. The transformer encoder captures bi-directional context of the image, which is used by the transformer decoder to generate new codebook indices autoregressively.

Traditional Dense Attention. Traditional attention transforms each embedding linearly into a learned query, key, and value representation $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ of size $L \times d$, where $L = H_{\text{feat}} W_{\text{feat}} = \frac{H_{\text{im}}}{16} \frac{W_{\text{im}}}{16}$ is the length of the flattened codebook indices in our case [131]. The output embedding is then computed as $\text{softmax}(\mathbf{A}/\sqrt{d})\mathbf{V}$, where attention $\mathbf{A} = \mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{L \times L}$. The advantage of attention is that it allows for interactions across all positions in the sequence, i.e., in our case the whole encoded image, as illustrated in Figure 3.3 *(left)*. The disadvantage is computation of the attention matrix $\mathbf{A}$ has quadratic time and memory complexity in terms of sequence length: $\mathcal{O}(L^2) = \mathcal{O}(H_{\text{feat}}^2 W_{\text{feat}}^2)$. For an input image with resolution 1024×1024, the

sequence has length $L = 4096$, and practically the cost of performing the above matrix multiplication becomes prohibitively high, as discussed by several other works [126]. One simple way to reduce the computational cost is to use a sliding window approach [29], i.e., crop a fixed sized patch around the currently sampled token and feed it into the transformer. However, this introduces a bias towards preferring interactions only within local regions of the image, missing other useful longer-range interactions.

Sparsified Guided Attention (SGA). We propose an efficient sparsification strategy, without sliding windows. The key idea is to first compute coarse full attention with downsampled images, determine which locations are worth attending to, and then use it to avoid computing most entries of the attention matrix $\mathbf{A}$.

To do this, we first proceed by downsampling the original input image and semantic map to 256×256 resolution. We further encode them to obtain a feature map of size 16×16. At this resolution, computing full attention is fast, because the sequence length of codebook indices is only $16^2 = 256$. Then we employ a *guiding transformer*, which has the same architecture as the main transformer but is trained at the low resolution, to calculate the dense attention matrix $\mathbf{A}_{\text{low}} \in \mathbb{R}^{256 \times 256}$.

Then we leverage $\mathbf{A}_{\text{low}}$ to construct a block attention matrix $\mathbf{B} \in \mathbb{R}^{L \times L}$ at the original feature resolution that will guide the sparsification. To do this, we divide the original feature map into non-overlapping blocks, as illustrated in Figure 3.3 *(right)* for an 8×8 grid of blocks. For each block, we find the corresponding locations in $\mathbf{A}_{\text{low}}$ and average their affinity values. Note that the matrix $\mathbf{B}$ essentially consists of blocks, each of which is populated with a single affinity value.

Then we construct the sparse attention matrix $\mathbf{A}_{\text{sparse}}$ by considering only the attention weights that are likely important in approximating the true attention. To this end, we keep the attention if the corresponding affinity is high in the block attention matrix $\mathbf{B}$. In other words, we argue that the selection of sparse attention pairs can be reliably guided by the dense attention evaluated at lower resolution. In

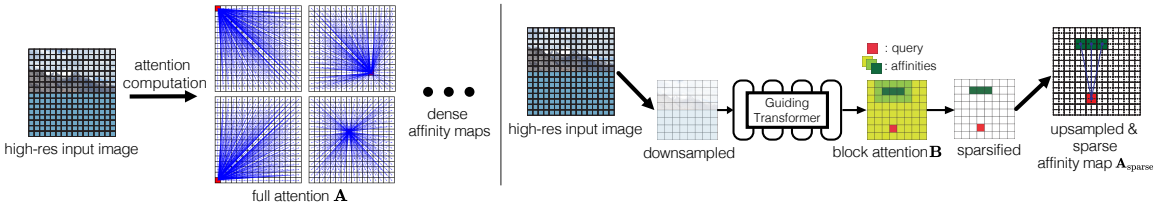

Figure 3.3: Details of our Sparsified Guided Attention (SGA, right) compared to full attention (left). We downsample the input and the user-edited semantic layout and use a Guiding Transformer to obtain the full attention map, which identifies the most important attention locations for each sampling step. By keeping only the locations with high attention weight, we construct the high-resolution, sparse attention map for the SGA transformer.

addition, following the proposition in the context of NLP models [151], we always compute attention within the current and adjacent blocks, no matter their affinity values.

$$\mathbf{A}_{\text{sparse}}(r,t) = \begin{cases} \mathbf{A}(r,t), & \text{if } t \in \mathcal{N}(r) \text{ or } t \in \mathcal{K}(r). \\ -\infty, & \text{otherwise,} \end{cases} \tag{3.1}$$

where $\mathcal{N}(r)$ contains the entries of the neighborhood blocks of r, and $\mathcal{K}(r)$ contains the entries of the blocks with the top-K highest affinities outside the neighborhood.

In our experiments, we set $K = 3$, resulting in the sparsity ratio <10%, and significantly reduce the computational cost of attention.

Transformer encoder. The input to our transformer encoder is a sequence of embeddings jointly representing the masked image codebook indices $\mathbf{x} = \{x_l\}_{l=1}^{L}$ and semantic codebook indices $\mathbf{p} = \{p_l\}_{l=1}^{L}$ produced by the image encoders (flattened using row-major format), and position information of each index in the corresponding sequence. Note that here, the transformers are operating both at full-resolution and low-resolution. Specifically, for each position in the sequence, three d-dimensional learned embeddings are produced: (i) an image embedding $E_{\text{im}}(x_l)$ representing the token x_l at position l in our sequence and in turn the corresponding RGB image

region, (ii) an embedding $E_{\text{map}}(p_l)$ of the semantic token p_l at the same position, and finally (iii) a positional embedding $E_{\text{pos}}(l)$ for that position l. The summation of token embeddings and positional embeddings follows other popular transformers [131, 27]:

$$\mathbf{e}_l = E_{\text{im}}(x_l) + E_{\text{map}}(p_l) + E_{\text{pos}}(l) \tag{3.2}$$

The sequence of embeddings $\{\mathbf{e}_l\}_{l=1}^{L}$ is fed to the first transformer encoder layer and is transformed to a continuous representation by the stack of transformer encoder layers. To preserve the position information of each index at subsequent layers, the position encoding generator (PEG) is placed before each encoder layer [19, 18].

Transformer decoder. The decoder predicts codebook indices for the edited region with the help of the global context obtained through the transformer encoder. Similar to BART [79], the autoregressive generation starts by pre-pending a special index (token) `[START]` to the decoder input. At each step, the decoder predicts a distribution over codebook indices from our dictionary $\mathcal{Z}$ learned in our image encoder (Section 3.1.1). Specifically, the decoder predicts $p(\mathcal{X}_l|\{\chi_{<l}\})$, where $\mathcal{X}_l$ is a categorical random variable representing a codebook index to be generated at position l in the sequence and $\{\chi_{<l}\}$ are all indices of the previous steps. We note that the tokens corresponding to unmasked image regions (i.e., image regions to be preserved) are set to the original image codebook indices. We predict the distributions only for positions corresponding to the edited image regions.

To predict the output distribution at each step, the decoder first takes as input a learned embedding $D_{\text{im}}(x_l)$ representing the input token x_l, and a learned positional embedding $D_{\text{pos}}(l)$ for that position l. It sums the two embeddings $\mathbf{d}_l = D_{\text{im}}(x_l) + D_{\text{pos}}(l)$, then passes $\mathbf{d}_l$ into a self-attention layer (attention between generated tokens) and a cross-attention layer (attention between generated tokens and encoder output features). For both self-attention and cross-attention we make use of the sparsified

guided attention mechanism. We also note that the self-attention in the decoder layer is modified to prevent tokens from attending to subsequent positions. Based on the predicted distribution of codebook indices, we use top-k sampling [57, 29] ($k = 100$ in our experiments) to create multiple candidate output sequences, each of which can be mapped to a new image by the image decoder. The generated images are ordered by the joint probability of the distributions predicted by the decoder.

3.1.3 Image decoder

The image decoder takes as input the quantized feature map and decodes an RGB image of size $H_{\text{im}} \times W_{\text{im}} \times 3$ following VQGAN [29]. Due to the quantization process, the reconstruction of the encoder-decoder pair is not perfect and leads to minor changes in the areas that are not edited. To avoid this, we follow the same strategy as SESAME [99] and retain only the generated pixels in the masked regions while the rest of the image is retrieved from the original image. To further decrease any small artifacts around the borders of the masked regions we apply Laplacian pyramid image blending as a final post-processing step.

3.1.4 Training

We randomly sample free-form masks following [99] and use the semantic information in the masked area as user edits. The image encoder, decoder, and transformer are trained in a supervised manner on training images which contain ground-truth for masked regions. We first train our image encoders and decoders following VQGAN [29]. We then train our transformer architecture on images with 256×256 resolution using the original attention mechanism (full attention), which will be used as the *guiding transformer*. Following that, we switch to train our SGA-transformer with the sparsified guided attention on high resolution, specifically, we initialize its weights from the previously trained guiding transformer and fine-tune it at 512×512 resolu-

Encoder	Decoder
$\mathbf{X} \in \mathbb{R}^{H_{\text{im}} \times W_{\text{im}} \times 3}$	$\hat{\mathbf{F}} \in \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 256}$
Conv2D $\rightarrow \mathbb{R}^{H_{\text{im}} \times W_{\text{im}} \times 128}$	Conv2D $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$
4× { Residual Block, Downsample Block} $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$	Residual Block $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$
Residual Block $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$	Non-Local Block $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$
Non-Local Block $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$	Residual Block $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$
Residual Block $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 512}$	4× { Residual Block, Upsample Block} $\rightarrow \mathbb{R}^{H_{\text{im}} \times W_{\text{im}} \times 128}$
GroupNorm, Swish, Conv2D $\rightarrow \mathbb{R}^{H_{\text{feat}} \times W_{\text{feat}} \times 256}$	GroupNorm, Swish, Conv2D $\rightarrow \mathbb{R}^{H_{\text{im}} \times W_{\text{im}} \times 3}$

Figure 3.4: Architecture of the image encoder and decoder. Note that $H_{\text{feat}} = \frac{H_{\text{im}}}{16}$, $W_{\text{feat}} = \frac{W_{\text{im}}}{16}$.

tion again at 1024×1024 resolution. In all cases, we use the same losses proposed in VQGAN [28].

3.1.5 Implementation Details

Here we provide implementation details of our network architecture and training procedure. Our model is implemented in PyTorch.

Image encoder / decoder. Our image encoder (Section 3.1.1) and decoder (Section 3.1.3) use the architecture shown in Figure 3.4. The design of the networks follows the architecture proposed in VQGAN [29]. One difference is that we employ partial convolutions [87] and region normalization [149] in our image encoder while processing the unmasked image regions. The reason is that the features produced for unmasked regions should not be affected by the masked image regions. Partial convolutions and region normalization avoid any information leakage of the masked area.

The semantic map encoder follows the same architecture with regular convolutions. We note that during training, VQGAN measures the reconstruction error in terms of both the image and semantic map. Thus, along with the decoder for the image, we also use a decoder to reconstruct the semantic map. The semantic map decoder uses an architecture following the "decoder" column in Figure 3.4. A mi-

Dataset	n_E	n_D	# params $[M]$	n_h	$\lvert\mathcal{Z}\rvert$	dropout	n_e
Flickr-Landscape	7	15	343	16	1024	0.0	1024
COCO-Stuff	7	15	365	16	8192	0.0	1024
ADE20K	7	10	269	16	4096	0.1	1024

Table 3.1: Transformer hyperparameters. For every experiment, the number of total blocks N, the number of blocks in $\mathcal{N}(r)$, the number of blocks in $\mathcal{K}(r)$ are set to 64, 3, 3 respectively. n_E denotes the number of transformer layers in the bidirectional encoder, n_D is the number of transformer layers in the autoregressive decoder, # params is the number of transformer parameters, n_h is the number of attention heads in the transformer, $|\mathcal{Z}|$ is the number of codebook entries, dropout is the dropout rate used for training the transformer, and n_e is the token embedding dimensionality.

nor difference is that the number of input/output channels is changed from 3 to the number of categories C in the semantic map encoder and decoder.

Transformer. Our transformer (Section 3.1.2) follows the architecture presented in BART [79]. All the hyperparameters for the transformer are described in Table 3.1.

Following [19, 18], the position encoding generator (PEG) is placed before each transformer encoder layer. The position encoding generator is a 5×5 depth-wise convolution with the padding size of 2, which convolves with each block independently. Similar to the encoder layer, we add one PEG before the first decoder layer which takes the encoder output representation as input to produce positional embeddings for the transformer decoder.

Sparsified Guided Attention. For each transformer layer and attention head, a full attention matrix $\mathbf{A}_{low} \in \mathbb{R}^{256\times256}$ is computed from the downsampled input image. The computation of the block attention matrix $\mathbf{B}$ for high-resolution is guided by $\mathbf{A}_{low}$. Specifically, in our experiments, the number of blocks N is set to 64. Each block consists of $\frac{256}{64} = 4$ tokens at low-resolution. The affinity value of each block

corresponds to a 4×4 region in $\mathbf{A}_{low}$. In our implementation, we use a 2D average pooling layer with kernel size 4 and stride 4 to downsample $\mathbf{A}_{low}$ into $\mathbf{B}$.

Training details. For the image encoder/decoder and semantic encoder/decoder, we used the Adam optimizer [72] with learning rate $7.2 \cdot 10^{-6}$ and batch size 16. During the training of the guiding transformer, we used the AdamW optimizer [92] with learning rate $3.2 \cdot 10^{-5}$ and batch size 224. During the finetuning of the SGA-transformer, we used the AdamW optimizer [92] with learning rate $1.2 \cdot 10^{-5}$ and batch size 8. All training is done on 8 A100 GPUs.

3.2 Results

In this section, we present qualitative and quantitative evaluation for ASSET.

Dataset. We evaluate our ability to perform high-resolution semantic editing on the Flickr-Landscape dataset consisting of images crawled from the Landscape group on Flickr. It contains $440K$ high-quality landscape images. We reserve 2000 images for our testing set, while the rest are used for training. Following [99], we use 17 semantic classes including mountain, clouds, water, and so on. To avoid expensive manual annotation, we use a pre-trained DeepLabV2 [12] to compute segmentation maps for all images. We also use the ADE20K [158] and COCO-Stuff [7] datasets for additional evaluation.

Evaluation metrics. To automatically generate test cases, we simulate user edits by masking out the pixels belonging to a random semantic class for each test image. As explained in Section 3.1.2, we can sample several output images and also rank them according to their probability. For our experiments, we sample 50 images, and keep the top 10 of them, as also done in other works [88, 156]. This results in 20K generated images for our test set. To evaluate the perceptual quality of our edits

we compute the FID [52], LPIPS [154], and SSIM metrics [138]. For each ground-truth image of the test split, we evaluate these metrics against the synthesized image that achieves the best balance of them, as done in [88, 156]. To evaluate how well the models adhere to the specified label map at the edited areas we also compare the mean Intersection over Union (mIoU) and the pixel-accuracy between the ground truth semantic map and the inferred one using the pretrained DeepLabV2 model [12]. Finally, to evaluate diversity, we utilize the LPIPS metric following [156, 88]. The diversity score is calculated as the average LPIPS distance between 5K pairs of images randomly sampled from all generated images, as also done in [88, 156]. We also perform a user study to evaluate the perceptual quality of several models.

Baselines. We compare our method with several semantic image editing baselines: SESAME [99], INADE [125], Taming Transformers (TT) [29], and ImageBART [28]. SESAME and INADE are based on convolutional networks and only support image resolutions of up to 256×256 pixels. TT and ImageBART are based on Transformers, but use a sliding window approach at high resolution, in which the attention is only calculated on a local neighborhood for each sampling location. While SESAME can only produce a single output, the other three methods can generate diverse outputs for a given edit. For a fair comparison, when generating image samples using TT or ImageBART, we also selected top-10 images out of 50 sampled ones based on their probability. We selected top-10 images for INADE based on discriminator scores as done in [88, 156]. For all baselines, we use the authors' implementation and train them on the same datasets as our method.

Quantitative evaluation. For multimodal editing tasks we aim to obtain outputs that are both diverse and consistent. There is an inherent trade-off between diversity and consistency, as higher diversity can be achieved by sacrificing image consistency. As such, we aim to achieve maximal diversity without generating inconsistent results.

Res	Method	LPIPS ↓	FID ↓	SSIM ↑	mIoU ↑	accu ↑	div ↑
256	*INADE*	0.233	11.2	0.826	48.6	59.1	0.145
	SESAME	0.213	10.2	0.830	50.3	61.7	0.000
	TT	0.201	10.4	0.839	46.1	58.3	**0.187**
	ImageBART	0.196	10.0	0.841	47.3	58.5	0.163
	ASSET	**0.187**	**9.2**	**0.846**	**51.5**	**63.0**	0.151
512	*TT*	0.203	10.6	0.850	52.2	63.7	**0.186**
	ImageBART	0.199	10.4	0.851	52.4	63.3	0.168
	ASSET	**0.186**	**8.4**	**0.856**	**53.5**	**64.7**	0.145
1024	*TT*	0.210	10.9	0.881	50.4	61.7	**0.160**
	ImageBART	0.201	10.4	0.880	50.8	62.1	0.139
	ASSET	**0.160**	**7.7**	**0.887**	**54.1**	**65.2**	0.124

Table 3.2: Quantitative evaluation on the Flickr-Landscape dataset at various resolutions.

For all models that can generate more than one solution for a given edit, we choose the sample with the best balance of quantitative measures (out of the top 10 samples), as done in [88] and [156]. Table 3.2 shows the comparison with competing models on the Flickr-Landscape dataset at different resolutions.

Except for the diversity metric our model outperforms all competing methods on all resolutions. While TT and ImageBART achieve a higher diversity than our model, we observe that this higher diversity comes at the cost of inconsistent images (see Figure 3.6 and Figure 3.7). In contrast, our approach also achieves high diversity but shows much more consistent image outputs, both at lower and higher resolutions. At low resolution (256×256), our method differs from TT by using a bidirectional transformer encoder to capture global context and partial convolutions to prevent information leakage from masked regions. As we increase the resolution (512×512 and higher), our approach continues to obtain consistent results, thanks to the Sparsified Guided Attention (SGA) that captures long-range dependencies. In contrast, the

Dataset	Method	LPIPS ↓	FID ↓	SSIM ↑	mIoU ↑	accu ↑	div ↑
COCO-Stuff	*TT*	0.237	20.2	0.820	36.9	51.7	**0.192**
	ASSET	**0.194**	**14.7**	**0.845**	**43.4**	**58.7**	0.156
ADE20K	*TT*	0.197	15.7	0.860	51.8	68.1	**0.155**
	ASSET	**0.191**	**14.0**	**0.862**	**52.6**	**68.8**	0.140

Table 3.3: Quantitative evaluation on the COCO-Stuff and ADE20K datasets at 512×512 resolution.

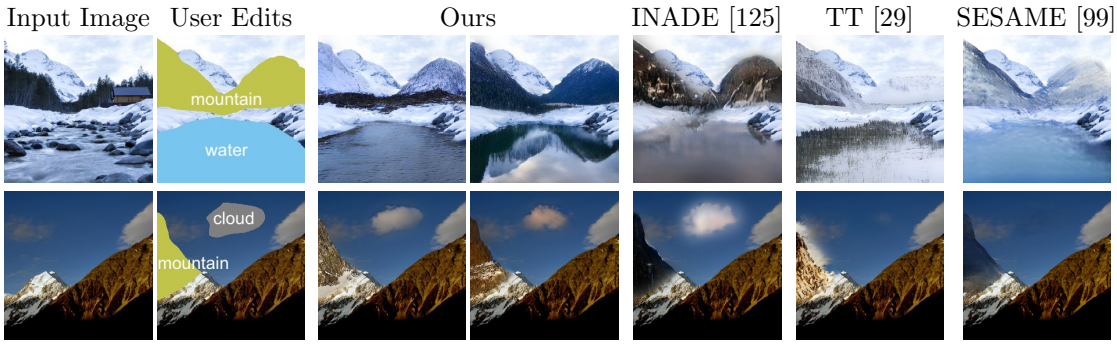

Figure 3.5: Comparison of our approach and all baselines on images of 256×256 pixels resolution.

perceptual performance of TT and ImageBART decreases with increasing resolution, as the sliding window approach is unable to enforce consistency over long distances.

We also conduct experiments on the COCO-Stuff and ADE20K datasets. Table 3.3 shows the comparison with TT at 512×512 resolution. Our model outperforms TT on both datasets.

Qualitative evaluation. For qualitative evaluation, a brush of a semantic class is used, painting over the image. Figure 3.5 shows comparison with all competing methods at 256×256 resolution. Since we do not need SGA at small resolutions, we only use our guiding transformer for these examples. Compared to other approaches, our method produces more coherent content with fewer artifacts. In Figure 3.6 and Figure 3.7, we show the comparison on 1024×1024 images against Taming Trans-

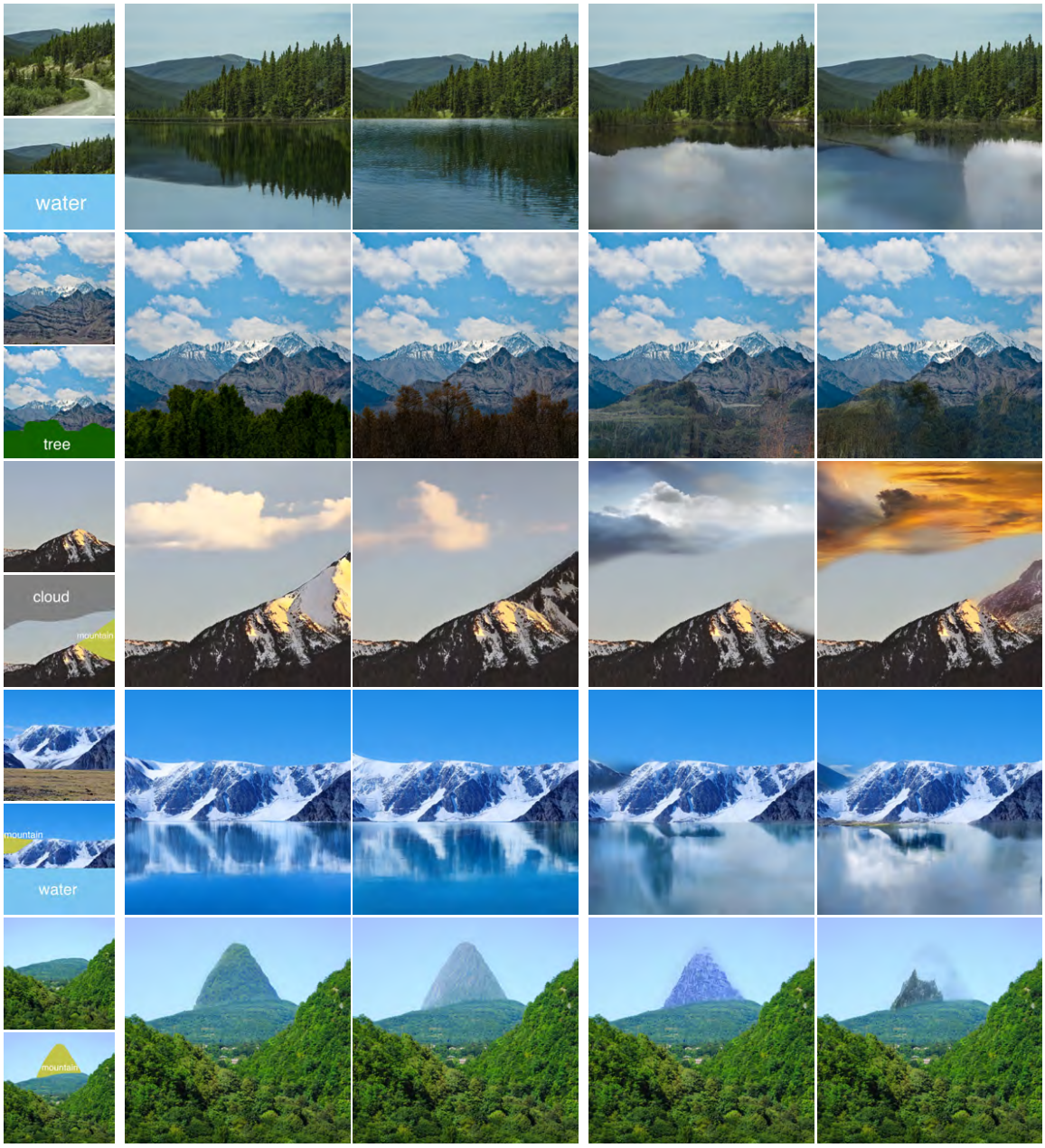

Figure 3.6: Comparison of ASSET with Taming Transformers [29] at 1024×1024 resolution.

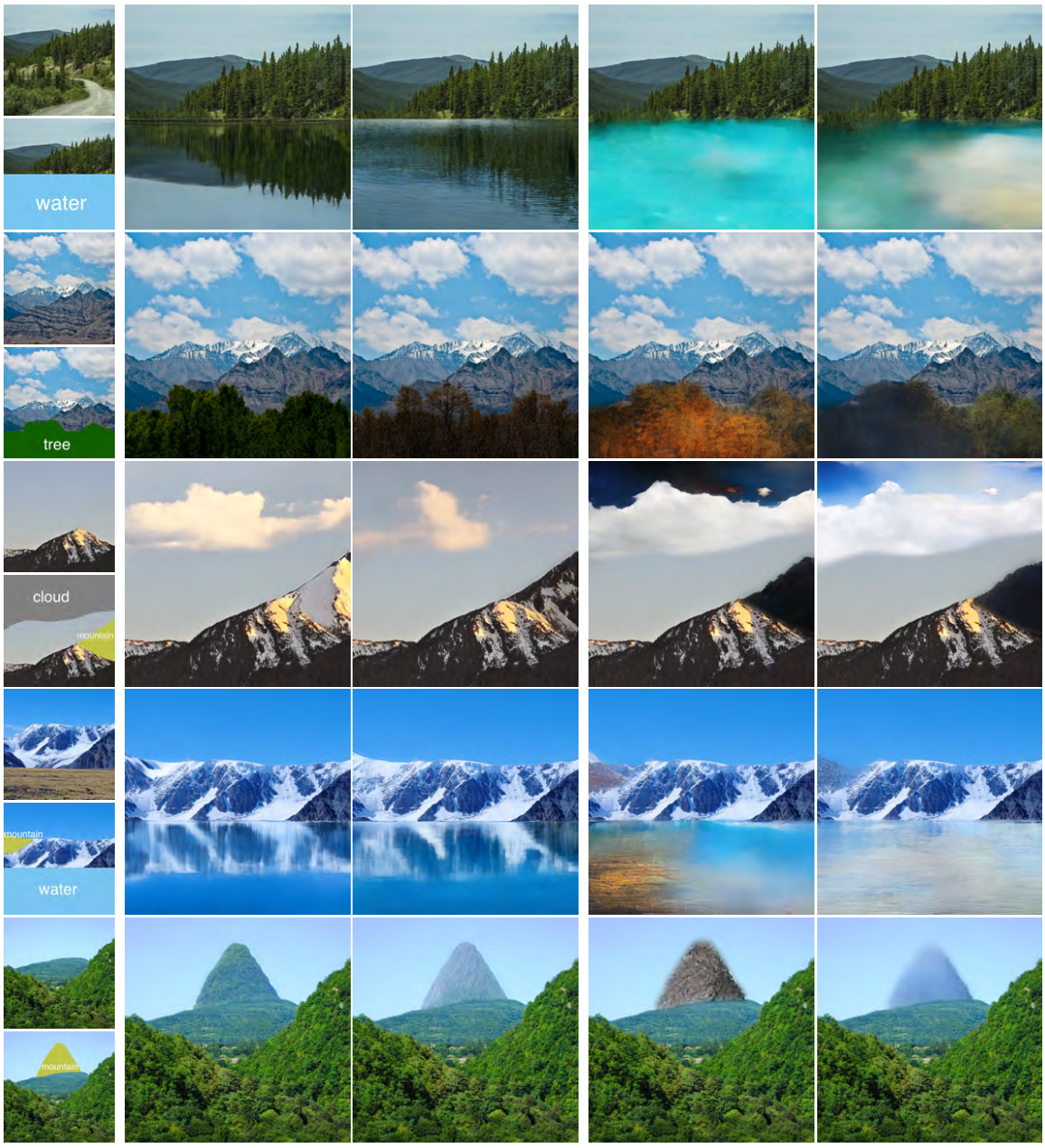

Figure 3.7: Comparison of ASSET with ImageBART [28] at 1024×1024 resolution.

Figure 3.8: Qualitative results and comparisons with Taming Transformers [29] on COCO-Stuff at 512×512 resolution.

Figure 3.9: Qualitative results and comparisons with Taming Transformers [29] on ADE20K at 512×512 resolution.

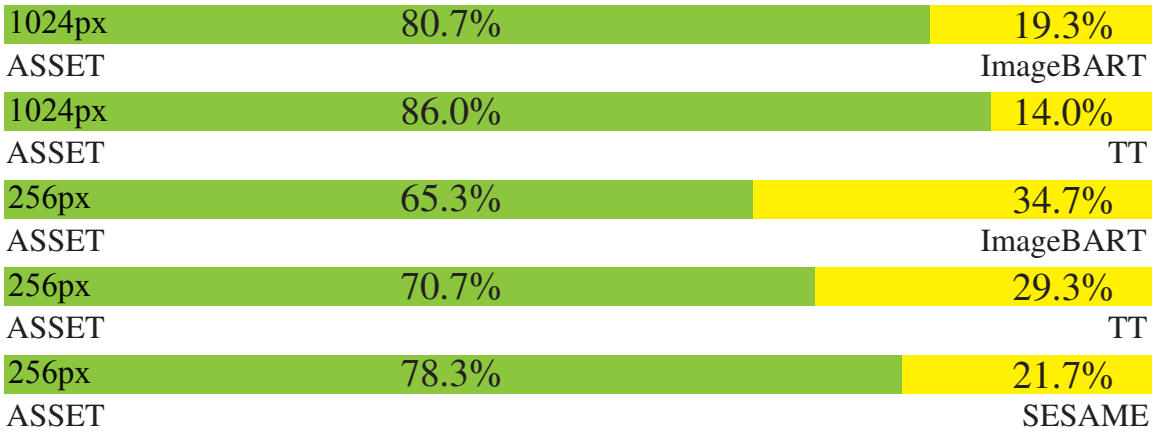

Figure 3.10: User study results. At both low and high resolution, our method ASSET is dominantly preferred over the baselines.

formers and ImageBART respectively. Figure 3.8 and Figure 3.9 show qualitative results on COCO-Stuff and ADE20K at 512×512 resolution. Even at high resolution, our method can synthesize coherent content across the whole image, while Taming Transformers and ImageBART fail to capture long-range dependency and sometimes ignore the user edits.

User study. To further evaluate perceptual image quality we also conduct an Amazon MTurk study. We showed participants a masked input image, along with a randomly ordered pair of images synthesized by ASSET and one of our baseline algorithms. The participants were then asked which edited image looks more photo-realistic and coherent with the rest of the image. Figure 3.10 summarizes 1500 user responses for 256×256 and 1024×1024 resolutions. The study shows that our method receives the most votes for better synthesis compared to other methods in both resolutions, with the largest margin at the highest 1024×1024 resolution.

Ablation study. To evaluate the effect of our SGA, we perform several ablations with different attention approaches. The following variants are evaluated at 512×512 resolution: **(1)** ***Sliding***: we use our guiding transformer with the sliding window ap-

Method	LPIPS ↓	FID ↓	SSIM ↑	mIoU ↑	accu ↑	div ↑
Sliding	0.214	10.7	0.847	52.6	63.1	**0.180**
Local	0.209	10.1	0.853	51.1	62.4	0.152
Random	0.202	9.6	0.851	50.0	61.6	0.157
ASSET	**0.186**	**8.4**	**0.856**	**53.5**	**64.7**	0.145

Table 3.4: Effects of different forms of attention on the Flickr-Landscapes dataset at 512×512 resolutions.

Figure 3.11: Comparison showing the effects of using different kinds of attention at 1024×1024 resolution.

proach as in [29]. **(2)** ***Local***: we remove our top-K attention and only use neighboring window attention $\mathcal{N}(r)$. **(3)** ***Random***: we use random instead of top-K attention similar to [151]. Table 3.4 shows the performance of all variants compared to our full model. Our model outperforms all variants in all metrics except for diversity. As before, we observe that higher diversity can be achieved at the cost of poorer image consistency. In Figure 3.11 we show qualitative comparisons with the proposed variants trained at 1024×1024 resolution. As we can see, without the SGA component, the image consistency and perceptual quality decreases as the model either only attends to local areas (*sliding* and *local*) or fails to attend to important image regions at each sampling step (*random*).

Method	LPIPS ↓	FID ↓	SSIM ↑	mIoU ↑	accu ↑
R+G	0.207	9.5	0.849	50.3	61.7
Random	0.202	9.6	0.851	50.0	61.6
ASSET	**0.186**	**8.4**	**0.856**	**53.5**	**64.7**

Table 3.5: Ablation for use of global blocks at 512×512 resolution.

Dataset	TT	ASSET
Landscape	**307M**	343M
COCO-Stuff	651M	**365M**
ADE20K	405M	**269M**

Table 3.6: Number of transformer parameters for TT and ASSET.

Ablation of global blocks. In our ablation study, we also experimented with the global block presented in BigBird [151]. Specifically, we make the first and last blocks "global", which attend over the entire sequence. Similar to [151], we use the global attention together with the local attention and random attention – this variant is referred to as *R+G*. The results did not improve compared to the *Random* variant in terms of our evaluation metrics (see Table 3.5).

Model capacity comparison with TT. We compare the number of transformer parameters with TT in Table 3.6. Our transformer's number of parameters is $\sim$12% larger than the one in TT for the Landscape dataset, and much smaller for the COCO-Stuff and ADE20K datasets. We note that ASSET's and TT's CNNs have the same number of parameters.

Attention visualization. We use Attention Rollout [1] to visualize the attention map of our guiding transformer encoder. Specifically, we average attention weights of the guiding transformer encoder across all heads and then recursively multiply the resulting averaged attention matrices of all layers. The attention maps for two

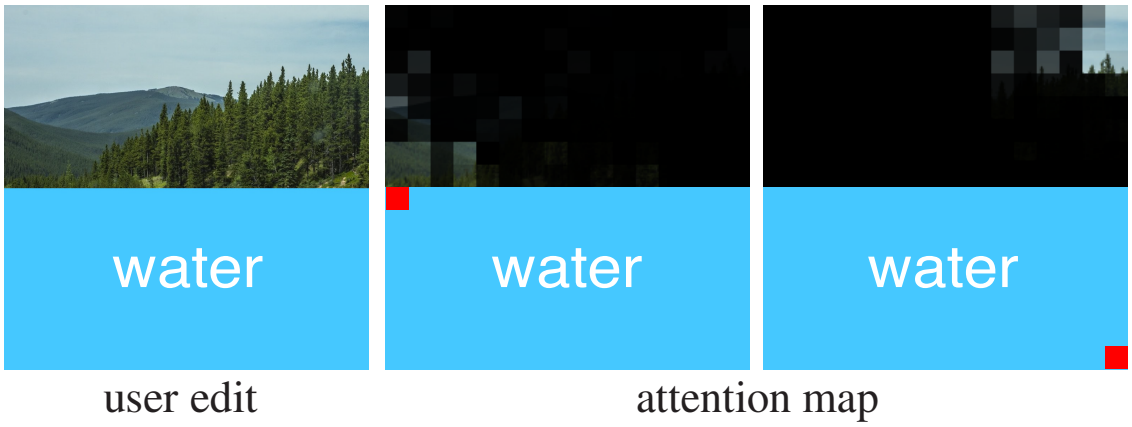

Figure 3.12: Encoder self-attention visualized for two different query points (shown as red). The image regions acquiring higher attention are the ones more relevant to generate water reflections at each of these points.

different query points are presented in Figure 3.12. The guiding transformer can attend to informative regions for different query points. For the query points in Figure 3.12, the regions with high attention correspond to image areas useful to synthesize reflection of scenery at each of these points.

VQGAN leakage visualization. We employ partial convolutions [87] and region normalization [149] in our image encoder while processing the unmasked image regions. The reason is that the features produced for unmasked regions should not be affected by the masked image regions. Partial convolutions and region normalization avoid any information leakage of the masked area. In Figure 3.13, we visualize leakage for the original VQGAN and our improved image encoder. With our modification, the latent features produced for unmasked regions are independent of the masked area.

Inference speed comparisons. Following [8, 28], we record the average inference time on Flickr Landscape and ADE20K as shown in Table 3.7. The inference speed is influenced by the size of the masked region relevant to the size of the input image (i.e., ratio of the masked region). Following [8], we report the average masked ratio

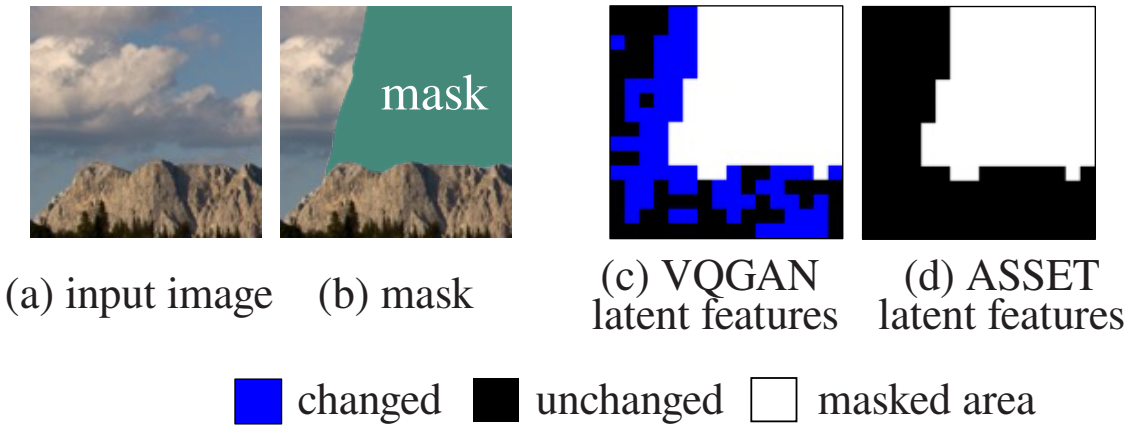

Figure 3.13: The masked area of the input image is replaced with random noise. The difference of 16×16 latent features between the original image (a) and the masked image (b) are visualized on the right. Changed and unchanged latent features are visualized in blue and black respectively. Image (c) shows how unmasked image latent tokens are affected (blue tokens) by the masked area in the original VQGAN. Image (d) shows that our image encoder successfully prevents leakage from the masked area to the unmasked area.

Resolution	Dataset	Masked Ratio	TT	ASSET
1024	Landscape	0.287	**52.3**	55.8
512	ADE20K	0.296	16.9	**10.6**

Table 3.7: Average inference time in seconds per image.

in this table. Our method achieves similar inference speed with TT, while producing much higher-quality results than TT.

Inference time of guiding transformer. Measured on the Landscape dataset, the average inference time of our guiding transformer (256×256 resolution) represents only a small fraction (3.4%) of the total inference time of the full ASSET pipeline. The majority of the inference time (96.6%) is taken by our architecture operating at high resolution, which is the crucial part significantly accelerated by our SGA mechanism.

Figure 3.14: Example of a less successful result.

Comparison with full attention. Based on an NVIDIA A100 (40GB VRAM) at 1024×1024 resolution with a batch size of 1, the transformer architecture requirements with full attention exceeds the available memory during training. Using our Sparsified Guided Attention mechanism, the transformer architecture utilizes 37GB at train time. In terms of inference time during testing, the cost of the guiding transformer is significantly lower: ASSET is about 20 times faster at test time compared to using full attention at 1024 × 1024 resolution.

Failure case. Structured textures such as the mountain in Figure 3.14 is challenging for reflection synthesis. In this case, our result may not reproduce the texture well.

3.3 Conclusion

We introduce a novel transformer-based approach for semantic image editing at high resolutions. Previous approaches have difficulty in modeling long-range dependencies between image areas that are far apart, resulting in unrealistic and inconsistent image edits. To this end, we introduce a novel attention mechanism called *Sparsified Guided Attention* (SGA), which uses the full attention map at the coarse resolution to produce a sparse attention map at full resolution. Our experiments show that SGA outperforms other variants of localized or sparse attention, and allows us

to obtain realistic and diverse image edits even at high resolutions of 1024×1024 pixels.

While our approach can perform consistent and diverse edits at high resolutions of up to 1024×1024 pixels, there are still avenues for further improvements. A common issue in transformers including ours is that directly applying a trained model to generate content at a higher resolution degrades performance, since the learned positional embedding cannot adapt to the new resolution. In addition, the repeated autoregressive sampling takes several minutes to perform a full edit at 1024×1024 resolution. To alleviate this issue, we can sample a diverse set of outputs for a given edit in parallel on multiple GPUs. Finally, the synthesized content may not be perfectly aligned with the provided mask since the masking takes place at a low resolution in the latent space.

CHAPTER 4

VECTOR ART SYNTHESIS GUIDED BY 3D SHAPES

The second part of my thesis focuses on a two-stage approach for line drawing synthesis and stylization from 3D shapes. In Section 4.1, I will present a method that learns to draw lines for 3D models based on a combination of a differentiable geometric module and an image translation network. In Section 4.2, I will discuss a model for stylizing line drawings with the guidance of 3D shapes.

4.1 Neural Contours: Line Drawing Synthesis

The first section of this chapter discusses Neural Contours, a method for learning to generate line drawings from 3D models [85] [1]. Our architecture incorporates a differentiable module operating on geometric features of the 3D model, and an image-based module operating on view-based shape representations. At test time, geometric and view-based reasoning are combined with the help of a neural module to create a line drawing. The model is trained on a large number of crowdsourced comparisons of line drawings. Experiments demonstrate that our method achieves significant improvements in line drawing over the state-of-the-art when evaluated on standard benchmarks, resulting in drawings that are comparable to those produced by experienced human artists.

[1] This work is published in the Proceedings of CVPR 2020.

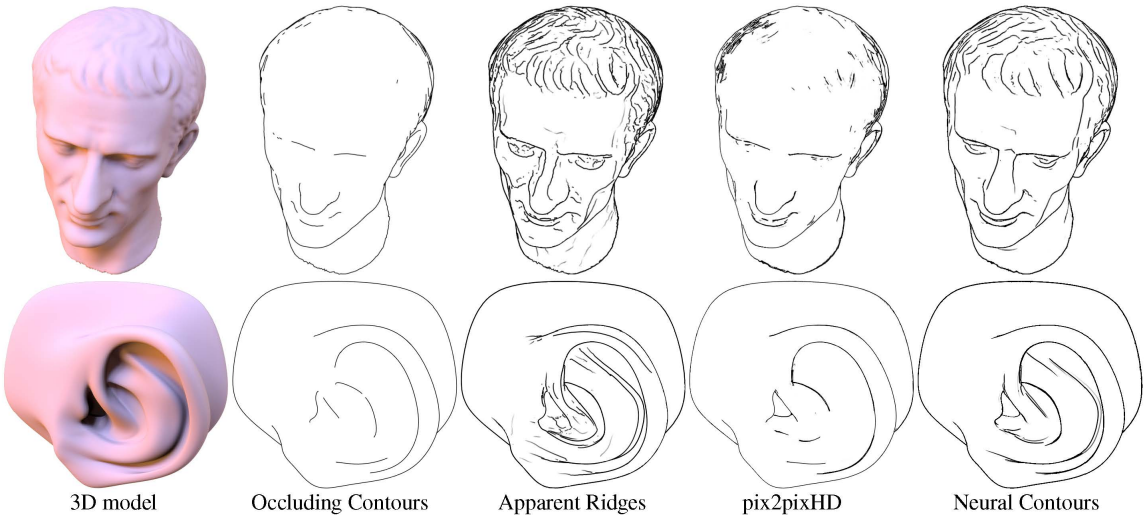

Figure 4.1: Given a 3D model (left), our network creates a line drawing that conveys its structure more accurately compared to using other methods individually, such as Occluding Contours [6], Apparent Ridges [68], and pix2pix variants [135].

4.1.1 Line Drawing Model

We first describe our architecture for computing a line drawing from a 3D shape. The model takes a 3D shape and camera as input, and outputs a line drawing. The 3D shape is represented as a triangle mesh that approximates a smooth surface. The output line drawing is specified as a 2D grayscale image. Our architecture has two branches (Figure 4.3): a "geometry branch" that performs line drawing based on geometric features of the 3D model, and an "image translation branch" that learns lines through image-to-image translation. Parameters for the geometric lines are set at run-time by a "ranking module". Training for the model is described in Section 4.1.3.

4.1.1.1 Geometry branch

The first branch of our model is based on classic geometry-based line drawing definitions, namely suggestive contours, ridges, valleys, and apparent ridges. Given a camera viewpoint and 3D shape, each of these formulations contributes to a grayscale pixel intensity map, which are combined to produce the final image $\mathbf{I}$. Their contribu-

tions depend on a set of thresholding parameters. Instead of setting these by hand, we introduce differentiable formulations to allow learning the thresholding parameters.

The first set of curves produced are **Occluding Contours** [6]. The model generates a binary mask $\mathbf{I}_C$ with "on" pixels at projections of occluding contours (Figure 4.2b), computed using the interpolated contour algorithm [50]. Occluding contours are parameter-free, and do not require any learning; they are used in all of our renderings. Another parameter-free set of lines are mesh boundaries, also rendered as a binary mask $\mathbf{I}_B$.

Suggestive Contours (SCs) [24] represent surface points that are occluding contours in nearby views. See DeCarlo [24] for a detailed explanation and definitions. Let κ be the *radial curvature*, and $D\kappa$ be the directional derivative of the radial curvature at a surface point, as defined in [24]. SCs are points where $\kappa = 0$ and $D\kappa > 0$. For meshes, these curves are computed by interpolation to find the zero set of κ. As seen in Figure 4.2c, rendering all SCs is undesirable. Instead, "weak" SCs are filtered by only rendering SCs with $D\kappa > t_S$ for some threshold t_S, and tapered off below t_S (Figure 4.2d) [24]. In previous work, this t_S parameter is manually adjusted for each 3D model. In order to determine this threshold automatically, we introduce a formulation that is differentiable with respect to t_S. For a given threshold, the method outputs a per-pixel intensity map $\mathbf{I}_S$. We build two image-space maps. First, $S(\mathbf{x})$ is a binary map that is 1 at the projections of suggestive contours, and 0 otherwise, where $\mathbf{x}$ indexes pixel locations in the image. Second, $D\kappa(\mathbf{x})$ associates each pixel $\mathbf{x}$ to the directional derivative of the radial curvature at the surface point visible from that pixel. Figure 4.3 shows these two image-space maps for an input 3D shape. Then, the SC image is computed for each pixel $\mathbf{x}$ as:

$$\mathbf{I}_S(\mathbf{x}, t_S) = S(\mathbf{x}) \ \max\left(1 - \frac{t_S}{D\kappa(\mathbf{x})},\ 0\right) \tag{4.1}$$

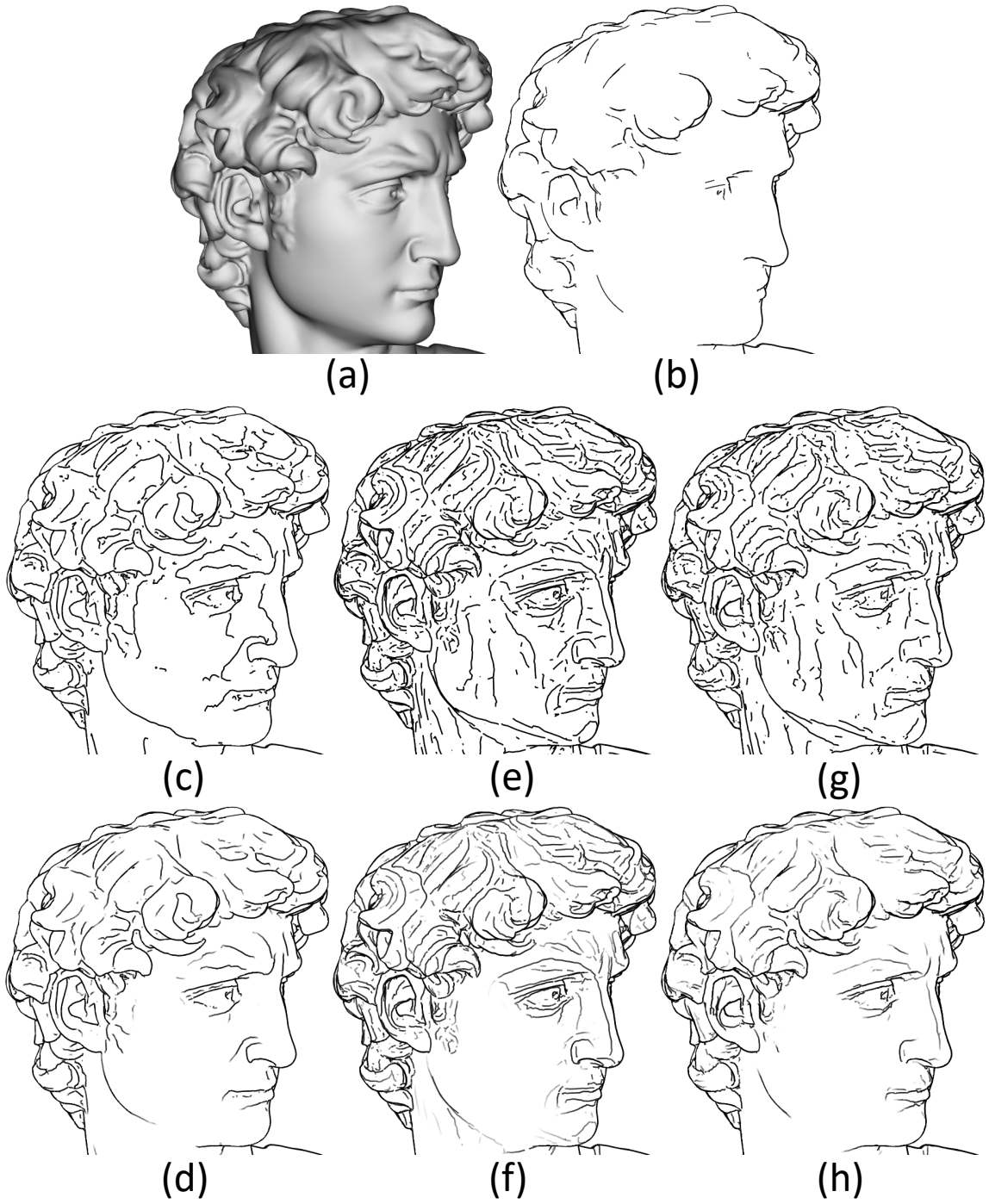

Figure 4.2: Given a 3D shape (a), we show (b) occluding contours, (c,d) unfiltered and filtered suggestive contours, (e,f) unfiltered and filtered ridges and valleys, (g,h) unfiltered and filtered apparent ridges.

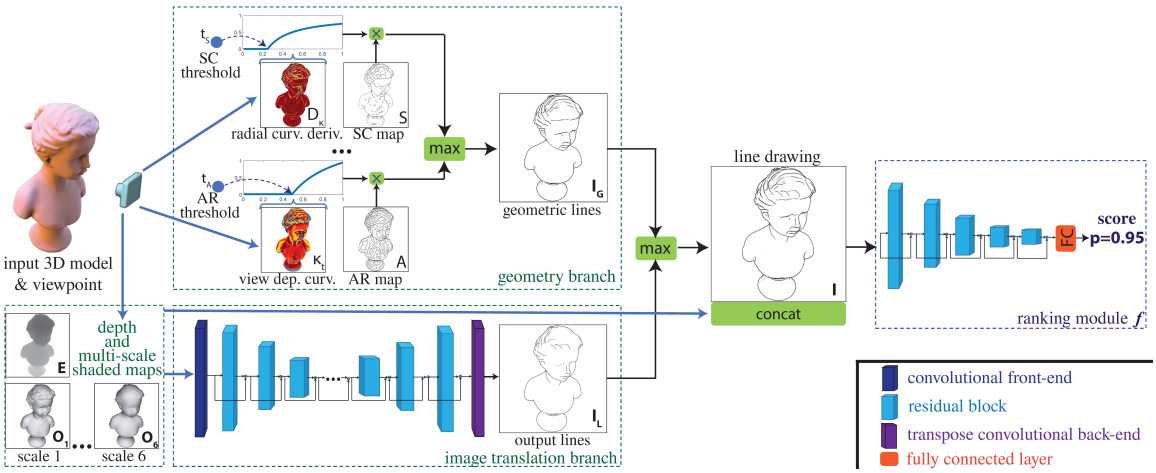

Figure 4.3: Our network architecture: the input 3D model is processed by a geometry branch operating on curvature features, and an image-based branch operating on view-based representations. Their outputs are combined to create a line drawing, which is in turn evaluated by a ranking module that helps determining optimal line drawing parameters.

The second term filters out lines with small $D\kappa$. For $t_S = 0$, all suggestive contours are displayed, while, as t_S increases, they are eliminated. The inverse function is used rather than a linear dependence, e.g., $\max(D\kappa(\mathbf{x}) - t_S, 0)$, to produce a sharper tapering, following the implementation in `rtsc` [113]. DeCarlo *et al.*[24] also proposed filtering according to the radial direction magnitude, but we did not find that it was much different.

Ridges and Valleys (RVs) are viewpoint-independent surface extrema; see [100] for a detailed explanation. As with SCs, we introduce a formulation that is differentiable with respect to the filtering function. We introduce a threshold for ridges (t_R) and one for valleys (t_V). The per-pixel intensity maps showing locations of ridges and valleys are generated as $R(\mathbf{x})$ and $V(\mathbf{x})$, along with maps $\kappa_1(\mathbf{x})$, $\kappa_2(\mathbf{x})$ containing the two principal curvatures of the surface point visible from each pixel, respectively. Ridges and valleys are then filtered as:

$$\mathbf{I}_R(\mathbf{x}, t_R) = R(\mathbf{x}) \ \max\left(1.0 - \frac{t_R}{\kappa_1(\mathbf{x})},\ 0.0\right) \tag{4.2}$$

$$\mathbf{I}_V(\mathbf{x}, t_V) = V(\mathbf{x}) \ \max\left(1.0 - \frac{t_V}{\kappa_2(\mathbf{x})},\ 0.0\right) \tag{4.3}$$

The interpretation of the formula is similar to SCs and yields RVs consistent with `rtsc` [113]. Figures 4.2e and 4.2f show an example of RVs before and after filtering.

Apparent Ridges (ARs) [68] are object ridges from a given camera position, e.g., object points that "stick out" to the viewer; see [68] for a detailed description. We define $A(\mathbf{x})$ as the map containing ARs, and filter by the view-dependent curvature $\kappa_t(\mathbf{x})$:

$$\mathbf{I}_A(\mathbf{x}, t_A) = A(\mathbf{x}) \ \max\left(1.0 - \frac{t_A}{\kappa_t(\mathbf{x})}, 0.0\right) \tag{4.4}$$

Figures 4.2g and 4.2h show ARs before and after filtering.

Line drawing function. Given each of these functions, we define a combined *geometric line drawing function* $\mathbf{I}_G$ conditioned on the set of parameters $\mathbf{t} = \{t_S, t_R, t_V, t_A\}$ (we drop the pixel id $\mathbf{x}$ for clarity):

$$\mathbf{I}_G(\mathbf{t}) = \max\big(\mathbf{I}_S(t_S), \mathbf{I}_R(t_R), \mathbf{I}_V(t_V), \mathbf{I}_A(t_A), \mathbf{I}_C, \mathbf{I}_B\big) \tag{4.5}$$

where the max function operates per pixel independently.

Preprocessing. In a pre-processing step, we compute the curvatures required for the above lines from the input mesh. using [112]. We normalize object size so that the longest dimension is equal to 1 and the curvature quantities are divided by the their $90th$ percentile value.

4.1.1.2 Image translation branch

An alternative approach to create line drawings from shapes is to use a neural network that directly translates shape representations to 2D images. To simplify the

mapping, one can feed view-based shape representations as input to such network (i.e., depth images, shaded renderings), which also allows us to re-purpose existing image-to-image translation networks. Our method also incorporates this generative approach. Specifically, following pix2pixHD [135], we used an image translation network module, shown in Figure 4.3. As input to the network, we use a combination of view-based representations. First, we use a depth image $\mathbf{E}$ of the shape from the given camera. Then we also compute shaded rendering images representing Lambertian reflectance (diffuse shading) [108] created from the dot product of surface normals with light direction (light is at the camera). We use these shaded renderings because shading features are important predictors of where people draw lines [20]. To increase robustness to rendering artifacts, we also smooth the mesh normals using diffusion [66] with different smoothing parameters. As a result, we create a stack of six smoothed versions of shaded images $\mathbf{O} = \{\mathbf{O}_1, \mathbf{O}_2, ..., \mathbf{O}_6\}$, which are concatenated channel-wise with the depth image (Figure 4.3, bottom left). The resolution of all images is set to 768×768. We found that using these combined multiple inputs produces better results.

We also experimented with feeding rendered curvature maps, however, as we discuss in our experiments section, the results did not improve. Finally, since the network outputs per-pixel line probabilities, we use the ridge detection procedure of Cole *et al.*[20] so that the output image $\mathbf{I}_L$ contains cleaner curves.

4.1.1.3 Image translation branch implementation

We provide here more implementation details for our image translation branch.

Multi-scale shaded maps. We smooth mesh normals by diffusing the vertex normal field in one-ring neighborhoods of the mesh through a Gaussian distribution. Each vertex normal is expressed as a weighted average of neighboring vertex normals. The weights are set according to Gaussian functions on vertex distances. The

Layer	Activation size
Input	$768 \times 768 \times 7$
Conv(7x7, 7→64, stride=1)	$768 \times 768 \times 64$
Conv(3x3, 64→128, stride=2)	$384 \times 384 \times 128$
Conv(3x3, 128→256, stride=2)	$192 \times 192 \times 256$
Conv(3x3, 256→512, stride=2)	$96 \times 96 \times 512$
Conv(3x3, 512→1024, stride=2)	$48 \times 48 \times 1024$
9 Residual blocks	$48 \times 48 \times 1024$
Conv(3x3, 1024→512, stride=1/2)	$96 \times 96 \times 512$
Conv(3x3, 512→256, stride=1/2)	$192 \times 192 \times 256$
Conv(3x3, 256→128, stride=1/2)	$384 \times 384 \times 128$
Conv(3x3, 128→64, stride=1/2)	$768 \times 768 \times 64$
Conv(7x7, 64→1, stride=1)	$768 \times 768 \times 1$

Table 4.1: Architecture of the Image Translation Branch.

standard deviation σ of the Gaussians control the degree of influence of neighboring vertex normals: when σ is large, the effect of smoothing is larger. The map $\mathbf{O}_1$ is generated based on the initial normal field, while $\mathbf{O}_2, \mathbf{O}_3, \mathbf{O}_4, \mathbf{O}_5, \mathbf{O}_6$ are created using smoothing based on $\sigma = \{1.0, 2.0, 3.0, 4.0, 5.0\}$ respectively.

Architecture details. Our image translation branch uses the architecture shown in Table 4.1. All convolutional layers are followed by batch normalization and a ReLU nonlinearity except the last convolutional layer. The last convolutional layer is followed by a sigmoid activation function. The branch contains 9 identical residual blocks, where each residual block contains two 3×3 convolutional layers with the same number of filters for both layers.

4.1.1.4 Neural Ranking Module

As discussed earlier, the thresholding parameters $\mathbf{t}$ play an important role in determining the existence and tapering of the geometric lines. The threshold values determine how much intensity each geometric line will contribute to the final image, if at all. Our approach determines the threshold parameters $\mathbf{t}$ at test time, since

different 3D models may be best rendered by different combinations of geometric lines. We employ a Neural Ranking Module (NRM) that scores the quality of a given line drawing. Then, at test time, the thresholds $\mathbf{t}$ are set by optimization of the NRM score.

Specifically, the module is a function of the merged line drawing $\mathbf{I}(\mathbf{t}) = \max(\mathbf{I}_G(\mathbf{t}), \mathbf{I}_L)$, the depth image of the shape from the given viewpoint $\mathbf{E}$, and also the multi-scale shaded images $\mathbf{O}$ (Figure 4.3). The module is a neural network $f(\mathbf{I}(\mathbf{t}), \mathbf{E}, \mathbf{O}, \boldsymbol{\phi}) = p$, where p is the output score, and $\boldsymbol{\phi}$ are the learned network parameters. At test time, we aim to maximize this function (i.e., the quality of the drawing) by modifying the parameters $\mathbf{t}$:

$$\arg\max_{\mathbf{t}} f(\mathbf{I}(\mathbf{t}), \mathbf{E}, \mathbf{O}) \tag{4.6}$$

The maximization is done with L-BFGS using analytic gradients $(\partial f/\partial \mathbf{I}) \cdot (\partial \mathbf{I}/\partial \mathbf{t})$, computed from backpropagation since our modules are differentiable. We also impose a non-negativity constraint on the parameters $\mathbf{t}$, as specified by the geometric definitions of the lines. To avoid local minima, we try multiple initializations of the parameter set $\mathbf{t}$ through a grid search.

The function 4.6 is also used to choose whether to render mesh boundaries $\mathbf{I}_B$. This is simply a binary check that passes if the function value is higher when boundaries are included in $\mathbf{I}(t)$. Once the ranking network has determined the optimal parameters $\mathbf{t}_{opt}$, the final drawing is output as $\mathbf{I}(\mathbf{t}) = \max(\mathbf{I}_G(\mathbf{t}_{opt}), \mathbf{I}_L)$. We note that the NRM does not adjust the contribution of the image translation module at test time. We tried using a soft thresholding function on its output $\mathbf{I}_L$, but it did not help. Instead, during training, the image translation module is fine-tuned with supervisory signal from the NRM. We also experimented with directly predicting the parameters $\mathbf{t}$ with a feed-forward network, but we found that this strategy resulted in much worse performance compared to test-time optimization of $\mathbf{t}$.

Layer	Activation size
Input	$768 \times 768 \times 3$
Conv(7x7, 8→64, stride=2)	$384 \times 384 \times 64$
Max-pool(3x3, stride=2)	$192 \times 192 \times 64$
ResBlock(64→64, stride=1, blocks=3)	$192 \times 192 \times 64$
ResBlock(64→128, stride=2, blocks=4)	$96 \times 96 \times 128$
ResBlock(128→256, stride=2, blocks=6)	$48 \times 48 \times 256$
ResBlock(256→512, stride=2, blocks=3)	$24 \times 24 \times 512$
ResBlock(512→1024, stride=2, blocks=3)	$12 \times 12 \times 1024$
Average-pool(12x12)	1024
FC(1024→1)	1

Table 4.2: Architecture of the Neural Ranking Module.

NRM architecture. The neural ranking module follows the ResNet-34 architecture. The input is the line drawing **I**, depth **E**, and shaded maps **O** at 768×768 resolution that are concatenated channel-wise. To handle this input, we added one more residual block after the original four residual blocks of ResNet-34 to downsample the feature map by a factor of 2. The newly added residual block produces a $12 \times 12 \times 1024$ map. After mean pooling, we get a 1024-dimension feature vector. We remove the softmax layer of ResNet-34 and use a fully connected layer to output the "plausibility" value.

4.1.1.5 Neural Ranking Module Implementation

We provide here implementation details for our Neural Ranking Module.

Architecture details. Our Neural Ranking Module uses the architecture shown in Table 4.2. It follows the ResNet-34 architecture. We add one more residual block with 1024 filters after the original four residual blocks. After average pooling, we get a 1024-dimensional feature vector. We remove the softmax layer of ResNet-34 and use a fully connected layer to output the "plausibility" value.

4.1.2 Dataset

To train the the neural ranking and image translation modules, we need a dataset of line drawings. Although there are a few large-scale human line drawing datasets available online [67, 115], the drawings are not associated to reference 3D shapes and include considerable distortions. An alternative scenario is to ask artists to provide us with line drawings depicting training 3D shapes. However, gathering a large number of human line drawings for training is labor-intensive and time-consuming. Cole *et al.*'s dataset [20] was gathered this way, and is too small on its own to train a deep model. In contrast, for each training shape, we generated multiple synthetic line drawings using `rtsc` [113] through several combinations of different lines and thresholds, and asked human subjects to select the best drawing in a relative comparison setting. Since selecting the best drawing can be subjective, we gathered votes from multiple human subjects, and used only training drawings for which there was consensus. Below we describe our dataset, then we describe the losses to train our modules.

Shape dataset. The first step to create our dataset was to select training 3D shapes from which reference line drawings will be generated. We used three collections: ShapeNet [10], Models Resource [111], and Thingi10K [159]. These shape collections contain a large variety of human-made and organic objects. In the case of ShapeNet, we sampled a random subset of up to 200 shapes from each category, to avoid category imbalance. All models have oriented ground planes specified. We removed duplicate shapes and avoided sampling low-resolution shapes with fewer than 2K faces. We also processed the meshes by correctly orienting polygons (front-facing with respect to external viewpoints), and repairing connectivity (connect geometrically adjacent but topologically disconnected polygons, weld coincident vertices). The number of shapes in all collections is 23,477.

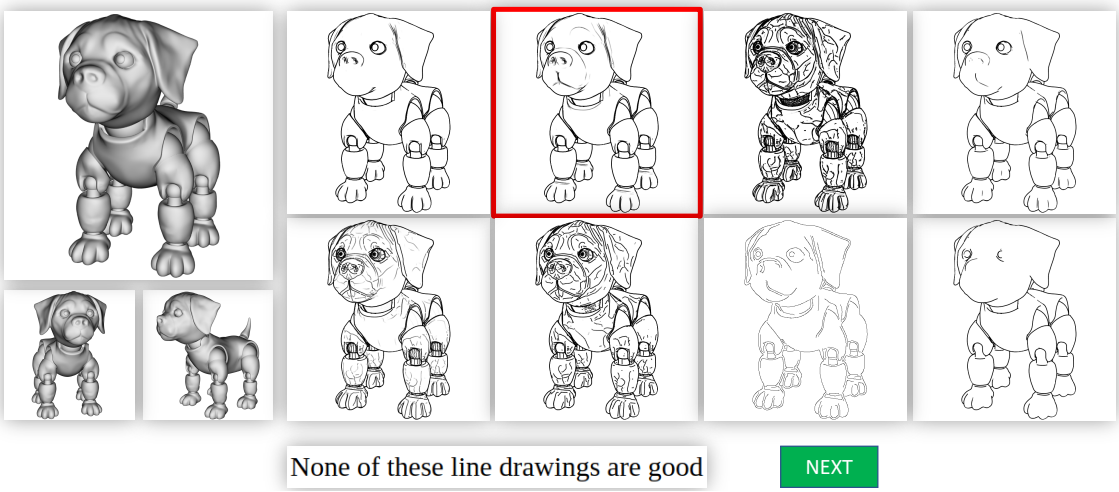

Figure 4.4: A snapshot from our MTurk questionnaires used for gathering training line drawing comparisons. The most voted answer is highlighted as red.

Generating candidate line drawings. We select two random camera positions for each 3D model under the constraint that they are elevated 30 degrees from the ground plane, are aligned with the upright axis, and they point towards the centroid of the mesh (i.e., only the azimuth of each camera position is randomized). Then for each of the two camera positions, we generated 256 synthetic line drawings using all possible combinations of suggestive contours, apparent ridges, ridges with valleys under 4 thresholds (e.g. suggestive contours have 4 different thresholds [0.001, 0.01, 0.1, off]), including combinations with and without mesh creases and borders ($4 \times 4 \times 4 \times 2 \times 2 = 256$ combinations). We also generated additional synthetic line drawings using Canny edges and edge-preserving filtering [35] on rendered shaded images of shapes, each with 4 different edge detection thresholds resulting in 8 more drawings. In total, this process resulted in 264 line drawings for each shape and viewpoint. These line drawings can be similar to each other, so we selected the 8 most distinct ones by applying k-mediods ($k = 8$) and using Chamfer distance between drawn lines as metric to recover the clusters.

Questionnaires. We then created Amazon MTurk questionnaires, where, on each page, we showed 8 candidate line drawings of a 3D shape, along with rendered images of it from different viewpoints [139] (Figure 4.4). Each page asked human participants to select the line drawing that best conveyed the rendered shape, and was most likely to be selected by other people as well. We also provided the option not to select any drawing (i.e., if none depicted the shape reliably). We employed sentinels (drawings of irrelevant shapes) to filter unreliable participants. We had total $3,739$ reliable MTurk participants in our study. For each shape and viewpoint, we gathered votes from 3 different participants. We accepted a drawing for training if it was chosen by at least two users. As a result, we gathered $21,609$ training line drawings voted as "best" per shape and viewpoint. A random subset (10% of the original dataset) was kept for hold-out validation.

Dataset Collection Details. We created Amazon MTurk questionnaires to collect our training dataset. Each questionnaire had 35 questions. 5 of the questions were randomly chosen from a pool of 15 sentinels. Each sentinel question showed eight line drawings along with renderings from a reference 3D model. One line drawing was created by an artist for the reference shape, and seven line drawings were created for different 3D models. The line drawings were presented to the participants in a random order. Choosing one of the seven line drawings (or the option "none of these line drawings are good") resulted in failing the sentinel. If a worker failed in one of these 5 sentinels, then he/she was labeled as "unreliable" and the rest of his/her responses were ignored. A total of 4396 participants took part in this user study to collect the training data. Among 4396 participants, 657 users (15%) were labeled as "unreliable". Each participant was allowed to perform the questionnaire only once.

4.1.3 Training

The goal of our training procedure is to learn the parameters $\boldsymbol{\phi}$ of the neural ranking module, and the parameters $\boldsymbol{\theta}$ of the image translation branch from our training dataset, so that high-quality drawings can be generated for 3D models.

NRM training. To train the Neural Ranking Module f, we use a ranking loss based on above crowdsourced comparisons. Given a drawing $\mathbf{I}_{best}^{s,c}$ selected as "best" for shape s and viewpoint c, we generate 7 pairwise comparisons consisting of that drawing and every other drawing $\mathbf{I}_{other,j}^{s,c}$ that participated in the questionnaire. We use the hinge ranking loss to train the module [48, 25]:

$$L_R = \sum_{s,c,j} \max(m - f(\mathbf{I}_{best}^{s,c}, \mathbf{E}, \mathbf{O}, \boldsymbol{\phi}) + f(\mathbf{I}_{other,j}^{s,c}, \mathbf{E}, \mathbf{O}, \boldsymbol{\phi}), 0) \tag{4.7}$$

where m is the margin set to 1.0.

Image translation module training. Based on the line drawings selected as "best" per reference shape and viewpoint, we use cross-entropy to train the image translation module. Specifically, we treat the intensity values $\mathbf{I}_{best}$ as target probabilities for drawing, and measure the cross-entropy of the predicted output $\mathbf{I}_L$ and $\mathbf{I}_{best}$:

$$L_{ce} = -\sum_x (\mathbf{I}_{best}(x) \log \mathbf{I}_L(x) + (1 - \mathbf{I}_{best}(x)) \log(1 - \mathbf{I}_L(x)))$$

We then further end-to-end fine-tune the image translation module along with the NRM module based on the ranking loss. We also experimented with adding the GAN-based losses of pix2pix [62] and pix2pixHD [135], yet, their main effect was only a slight sharpening of existing lines, without adding or removing any new ones.

Implementation. For the ranking module, we used the Adam optimizer [72] with learning rate $2 \cdot 10^{-5}$ and batch size 32. For the image translation module, we used Adam with learning rate set to $2 \cdot 10^{-4}$ and batch size 2.

4.1.4 Results

We evaluate our method and alternatives quantitatively and qualitatively. To perform our evaluation, we compare synthesized line drawings with ones drawn by humans for reference shapes. Below, we describe our test datasets, evaluation measures, and comparisons with alternatives.

Test Datasets. Cole *et al.* [20] conducted a study in which artists made line drawings intended to convey given 3D shapes. The dataset contains 170 precise human line drawings of 12 3D models under different viewpoints and lighting conditions for each model. Their resolution is 1024×768 pixels. Since the number of 3D test models is small, we followed the same setup as Cole *et al.* to gather 88 more human line drawings from 3 artists for 44 3D models under two viewpoints (same resolution), including printing renderings, scanning their line drawings, and aligning them. Our new test dataset includes 13 3D animal models, 10 human body parts, 11 furniture, 10 vehicles and mechanical parts. All 3D models (including the ones from Cole *et al.*'s dataset) are disjoint from the training and validation sets.

Evaluation measures. We use precision and recall measures for comparing synthetic drawings to human-made drawings, computed in the manner proposed by Cole *et al.*[20]. Each drawing is first binarized through thinning and thresholding. Precision is defined as the fraction of drawn pixels in a synthetic drawing that are near any drawn pixel of the human drawing of the same shape under the same viewpoint. Recall is defined as the fraction of pixels in the human drawing that are near any line of the synthetic drawing. Two pixels are "near" if they are within 1mm in the coordinates of the physical page the drawing was made on; this distance was based on measurements of agreement between human drawings in Cole *et al.*'s dataset. We aggregate precision and recall into F1-score.

We also report Intersection over Union (IoU) to measure overlap between synthetic and human drawings, based on the same definition of nearness. Lastly, we report the symmetric Chamfer distance, which measures the average distance between lines of synthetic and human drawings. Since both human and synthetic drawings include aligned silhouettes, all measures will appear artificially improved because of them. To avoid this biasing, we remove silhouettes from all human and synthetic drawings and measure performance based on the rest of the lines only.

Comparisons. We compare our method, Neural Contours (NCs), with several alternatives. (1) *Occluding Contours*. (2-4) *SC-rtsc, RV-rtsc, AR-rtsc* using `rtsc` [113] with the default thresholding parameters; occluding contours are also included in all renderings. (5) *all-rtsc* renders SCs, RVs, and ARs all together with `rtsc` [113], using the default parameters. We note that we also tried to tune these parameters using an exhaustive grid search to minimize average Chamfer distance in the training set, but this resulted in worse performance. (6) *decision tree:* The method of Cole *et al.*[20], namely, a decision tree (M5P from Weka [46]) operating on rendered curvature and gradient maps, trained on our dataset. (7) *Canny* edges extracted from the shaded shape rendering, as suggested by [20]. The edge detection parameters are selected using grid search to minimize average Chamfer distance in our training dataset. (8) *pix2pixHD* image translation [135], trained to output line drawings from an input depth image and shaded renderings of a shape (same as $\mathbf{E}, \mathbf{O}$ in our method). Training was done on the same dataset ("best" drawings) as ours using the GAN and feature matching loss [135]. The original architecture outputs a 4096×2048 image through three local enhancer networks. In our case, since the input and output have the same resolution (1024×768), we use only the global generator of pix2pixHD.

Ablation study. We also compare with training the following reduced variants of our method. *NC-geometry* uses our geometry-based branch and our neural ranker

Method	IoU	CD	F1	P	R
contours	31.5	31.70	34.8	83.0	22.0
AR-rtsc	53.4	12.56	54.1	52.5	55.7
RV-rtsc	49.8	12.96	52.3	44.5	63.5
SC-rtsc	40.5	13.96	44.0	43.9	44.1
all-rtsc	48.2	12.63	52.5	40.4	75.1
Decision Tree	46.9	12.17	49.9	38.6	70.4
Canny Edges	51.9	12.59	52.9	50.4	55.8
pix2pixHD	45.0	15.73	48.7	69.6	37.5
NCs	**57.9**	**10.72**	**60.6**	60.8	60.5

Table 4.3: Comparisons with competing methods using all drawings from Cole *et al.*'s dataset. IoU, F1, P, R are reported in percentages, CD is pixel distance.

Method	IoU	CD	F1	P	R
contours	43.5	24.63	49.6	90.2	34.3
AR-rtsc	59.9	10.64	63.3	62.6	64.0
Decision Tree	49.7	11.12	53.0	41.1	74.6
Canny Edges	58.0	11.16	61.3	56.7	66.7
pix2pixHD	50.5	13.35	54.2	75.1	42.4
NCs	**65.2**	**8.71**	**67.6**	66.3	69.0

Table 4.4: Comparisons with other methods using the most "consistent" human drawings from Cole *et al.*'s dataset.

module. *NC-image* uses the image translation module alone trained with the same losses as ours, and multi-scale shaded images as input. *NC-image-noms* uses the image translation module alone trained with the same losses as ours, and using a single shaded and depth image as input (no multi-scale shaded images). *NC-curv* is an alternative image translation module that uses curvature maps rendered in image-space concatenated with the multi-scale shaded images and depth.

Results. Tables 4.3 reports the evaluation measures for Cole *et al.*'s dataset for competing methods. Specifically, the synthetic drawings are compared with each human line drawing per shape and viewpoint, and the measures are averaged. Since

Method	IoU	CD	F1	P	R
contours	49.0	19.11	54.9	92.2	39.1
AR-rtsc	66.8	9.19	69.9	69.2	70.7
RV-rtsc	64.8	9.36	66.2	62.8	70.1
SC-rtsc	65.0	9.88	63.3	61.5	65.2
all-rtsc	64.4	9.70	68.6	58.6	82.7
Decision Tree	62.1	8.93	61.1	50.9	76.6
Canny Edges	65.6	8.57	64.6	59.8	70.2
pix2pixHD	66.0	9.62	68.2	76.9	61.2
NCs	**72.4**	**7.25**	**74.6**	74.5	74.8

Table 4.5: Comparisons in our new test dataset.

Method	IoU	CD	F1	P	R
NC-geometry	60.3	10.34	64.5	76.9	55.6
NC-image	60.0	9.97	62.9	65.0	61.0
NC-image-noms	58.4	10.85	60.7	59.1	62.3
NC-image-curv	56.1	10.72	60.0	61.0	59.0
NCs	**62.8**	**9.54**	**65.4**	65.5	65.4

Table 4.6: Ablation study.

there are artists that draw more consistently than others, we also include Table 4.4 as an alternative comparison. This table reports the evaluation measures in Cole *et al.*'s dataset when synthetic drawings are compared only with the most "consistent" human line drawing per shape and viewpoint, defined as the drawing that has the least Chamfer distance to the rest of the human drawings for that shape and viewpoint. We believe this comparison is more reliable than using all drawings, because in this manner, we disregard any "outlier" drawings, and also get a measure of how well methods match the most consistent, or agreeable, human line drawing. Table 4.5 also reports the evaluation measures for our new dataset. Based on the results, Neural Contours outperforms all competing methods in terms of IoU, Chamfer Distance, and F1, especially when we compare with the most consistent human drawings.

Table 4.6 reports comparisons with reduced variants of our method for the purpose of our ablation study. Our two-branch architecture offers the best performance compared to using the individual branches alone.

Figure 4.5 shows characteristic comparisons with competing methods, and Figure 4.6 shows comparisons with reduced variants of our method. We also include representative human drawings. Both figures indicate that our full method produces lines that convey shape features more similarly to what an artist would do. Figure 4.1 also shows comparisons with other alternatives. Our method tends to produce more accurate lines that are more aligned with underlying shape features, and with less artifacts.

Are the two branches learning the same lines? To check this hypothesis, we measure the IoU between the line drawings created from the geometry branch alone and the ones created from the image translation branch. The average IoU is 69.4%. This indicates that the two branches outputs have a partial overlap, but still they have substantial differences. As shown in Figure 4.6, the geometry branch makes explicit use of surface information in 3D, such as surface curvature, to identify important

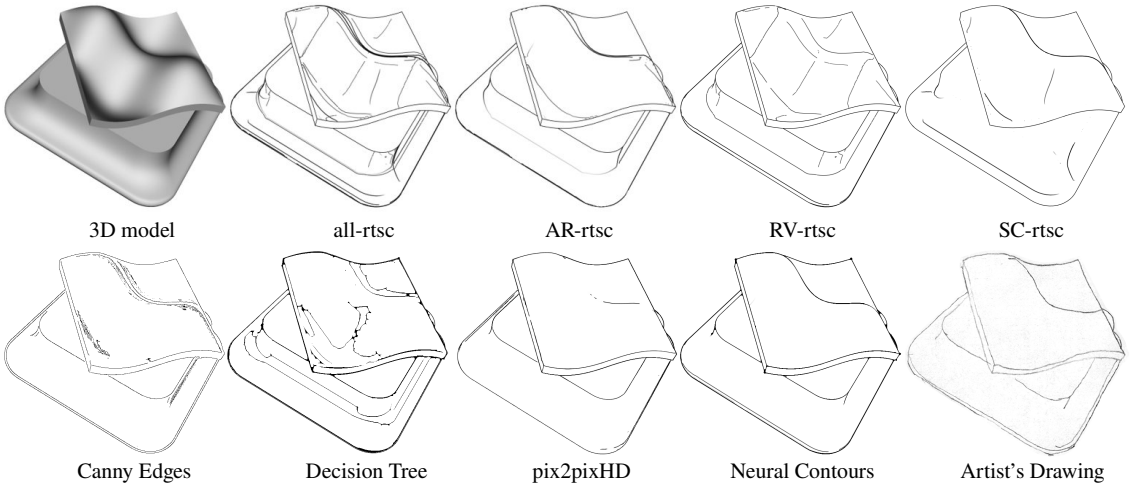

Figure 4.5: Comparisons with other methods. Neural Contours are more consistent with underlying shape features.

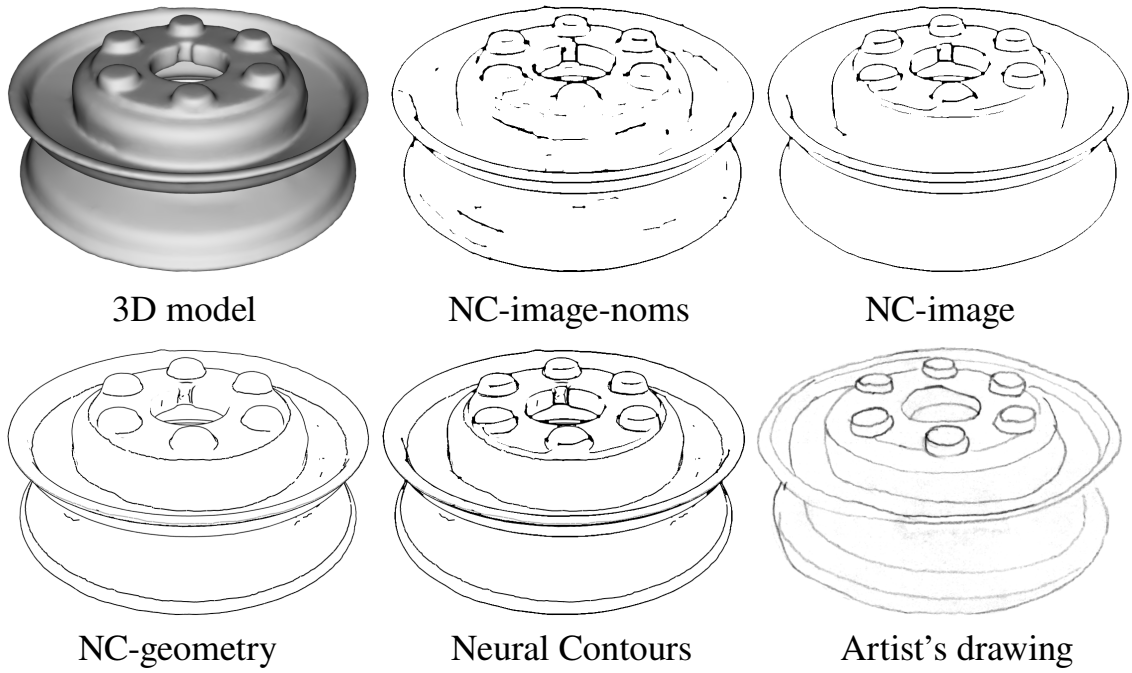

Figure 4.6: Comparisons with reduced NCs variants.

curves, which appear subtle or vanish in 2D rendered projections. In contrast, the image branch identifies curves that depend on view-based shading information that is not readily available in the 3D geometry.

Which geometric lines are used more in our method? The average percentage of SCs, RVs, ARs selected by our geometry-based stylization branch are 32.2%, 16.5%, 51.3% respectively. It seems that ARs are used more dominantly, while RVs are the least frequent lines.

User study. We also conducted an Amazon MTurk study as additional perceptual evaluation. Each questionnaire page showed participants shaded renderings of a shape, along with a randomly ordered pair of synthetic drawings: one synthetic drawing from our method, and another from a different one. We asked participants which drawing best conveyed the shown 3D model. Participants could pick either drawing, specify "none", or "both" drawings conveyed the shape equally well. We asked questions twice in a random order to verify participants' reliability. We had 187 reliable participants. Figure 4.7 summarizes the number of votes for the above options. Our method received twice the number of votes compared to the best alternative (ARs) found in this study.

In the Amazon Mechanical Turk perceptual evaluation where we showed participants (a) a rendered shape from a viewpoint of interest along with two more views based on shifted camera azimuth by 30 degrees, (b) a pair of line drawings placed in a randomized left/right position: one line drawing was picked from our method, while the other came from *pix2pixHD*, *NC-geometry*, *NC-image*, or *AR-rtsc*. We asked participants to select the drawing that best conveyed the shown 3D shape. Participants could pick one of four options: left drawing, right drawing, "none of the drawings conveyed the shape well", or "both" drawings conveyed the shape equally well". The study included the 12 shapes (2 viewpoints each) from both Cole *et al.*'s and our new

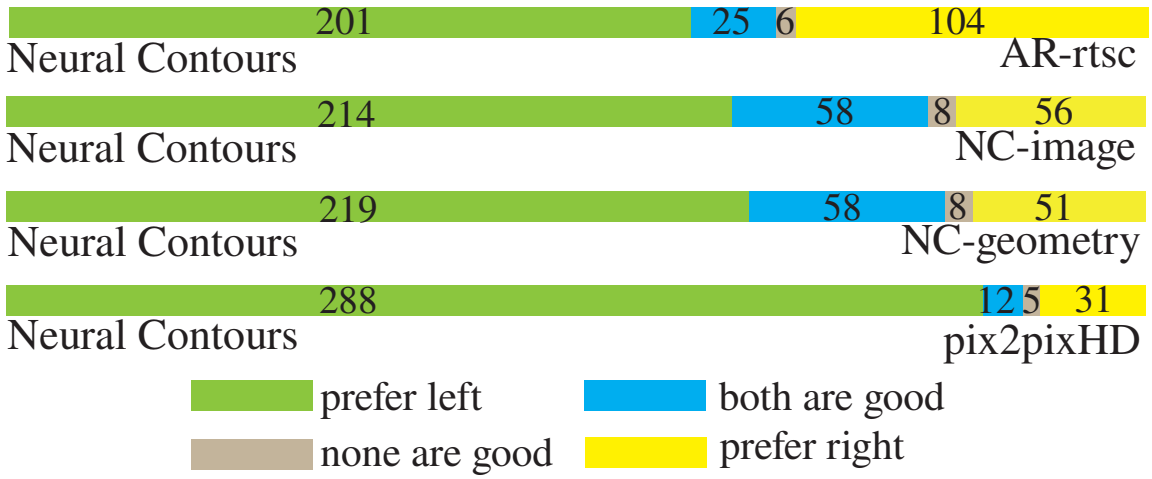

Figure 4.7: User study voting results.

collected test dataset (44 shapes, two viewpoints each). Thus, there were total 112 test cases, each involving the above-mentioned 4 comparisons of techniques (448 total comparisons).

Each questionnaire was released via the MTurk platform. It contained 15 unique questions, each asking for one comparison. Then these 15 questions were repeated in the questionnaire in a random order. In these repeated questions, the order of compared line drawings was flipped. If a worker gave more than 7 inconsistent answers for the repeated questions, then he/she was marked as "unreliable". Each participant was allowed to perform the questionnaire only once. A total of 225 participants took part in the study. Among 225 participants, 38 workers were marked as "unreliable". For each of the 448 comparisons, we gathered consistent answers from 3 different users.

4.1.5 Additional Results

Parameter set t regression. We experimented with directly predicting the parameter set $\mathbf{t}$ with a network, but this did not produce good results. The network includes a mesh encoder which is a graph neural network based on NeuroSkinning and

Method	IoU	CD	F1	P	R
AR-grid	56.6	11.21	59.1	54.2	64.9
RV-grid	56.0	11.73	58.3	53.6	63.9
SC-grid	51.0	12.57	53.2	57.5	49.5
all-grid	54.6	11.61	57.4	47.9	71.7
Geometry-Regressor	52.9	11.05	54.2	48.2	62.0
NCs	**62.8**	**9.54**	**65.4**	65.5	65.4

Table 4.7: Comparisons with competing methods using drawings from Cole et al.'s dataset and our newly collected dataset. IoU, F1, P, R are reported in percentages, CD is pixel distance.

an image encoder based on ResNet-34. The mesh encoder takes a triangular mesh as input and outputs a 1024-dimensional feature vector. The image encoder takes $(\mathbf{E}, \mathbf{O})$ as input and outputs a 1024-dimensional feature vector. These two feature vectors are concatenated and processed by a $3-$layer MLP which outputs the parameter set $\mathbf{t}$. We used cross-entropy loss between $\mathbf{I}_G(\mathbf{t})$ and $\mathbf{I}_{best}$ to train the network. We note that combining the mesh and image encoder worked the best. We name this variant *Geometry-Regressor*. Table 4.7 reports the resulting performance compared to our method. The results of this approach are significantly worse.

Parameter set t exhaustive search. We also tried to tune parameters of ARs, RVs, SCs using an exhaustive grid search to minimize average Chamfer distance in the training set. The grid was based on regular sampling 100 values of the parameters in the interval $[0, 1]$. This exhaustive search did not produce good results. Table 4.7 reports the performance of these variants *AR-grid*, *RV-grid*, *SC-grid*, *all-grid*.

Image translation vs geometric branch output example. Figure 4.9 shows an additional example of comparison between the geometry branch and the image translation branch outputs; compare the areas around the antlers, and the shoulder to see the contributions of each branch. The geometry model makes explicit use of surface information in 3D, such as surface curvature, to identify important curves,

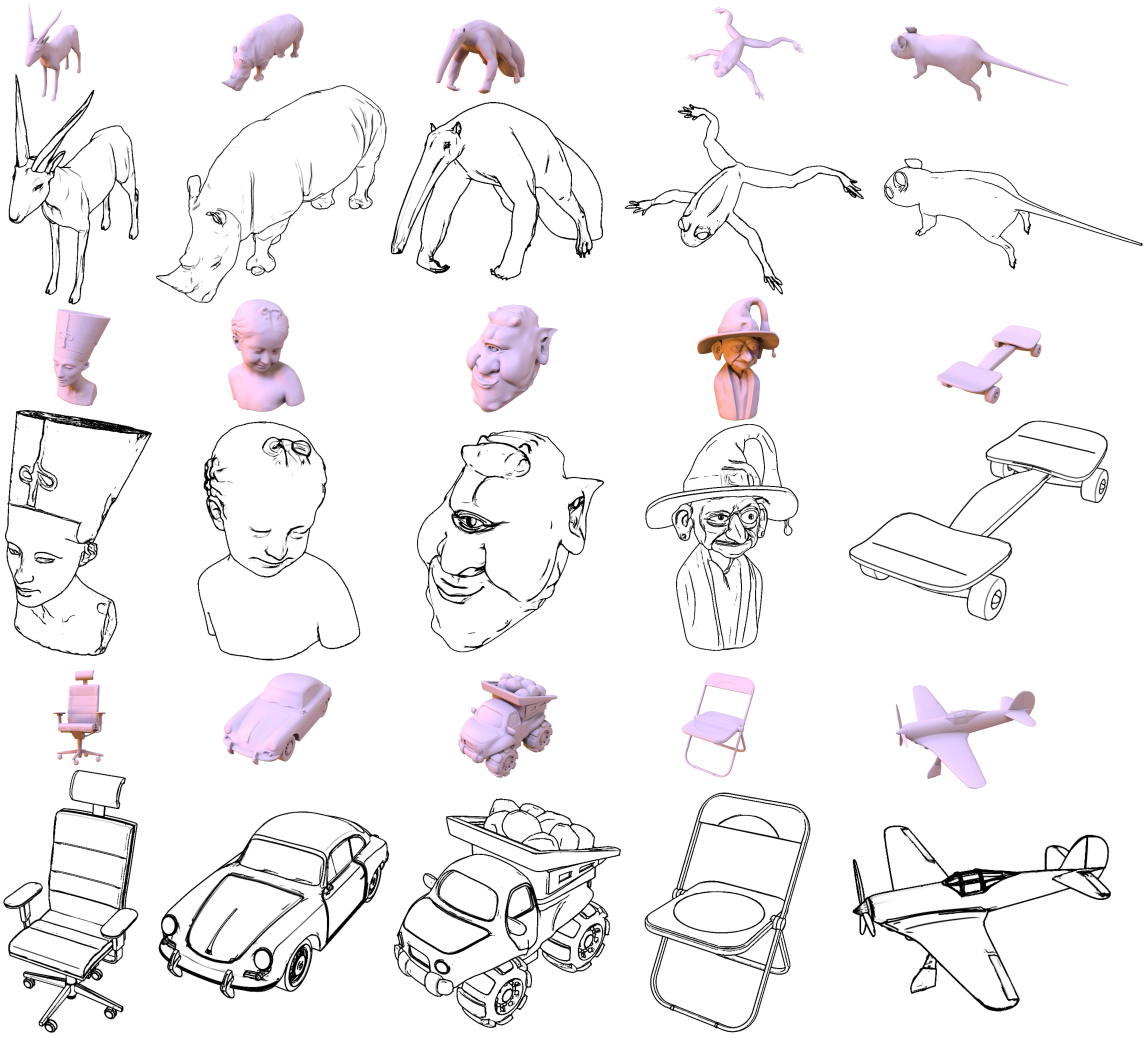

Figure 4.8: Results of our "Neural Contours" method on various test 3D models.

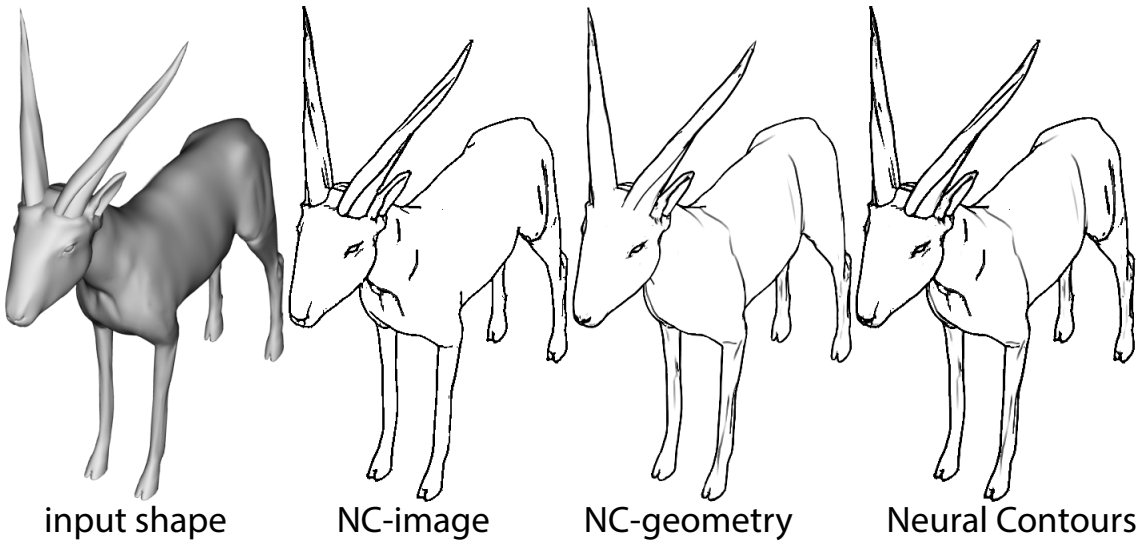

Figure 4.9: Additional comparison of our two branch outputs (image translation branch output "NC-Image" vs geometry branch output "NC-Geometry" vs Neural Contours).

which appear subtle or vanish in 2D rendered projections. In contrast, the image model identifies curves that depend on view-based shading information that is not readily available in the 3D geometry.

Gallery. Figure 4.8 shows a gallery of our results for various 3D models from our test set (please zoom-in to see more details).

4.1.6 Summary

In this section, we presented a method that learns to draw lines for 3D models based on a combination of a differentiable geometric module and an image translation network. Surprisingly, since the study by Cole *et al.*[20], there has been little progress on improving line drawings for 3D models. Our experiments demonstrate that our method significantly improves over existing geometric and neural image translation methods. There are still avenues for further improvements. Mesh artifacts (e.g., highly irregular tesselation) affect curvature estimation and shading, and in turn the outputs of both branches. Learning to repair such artifacts to increase robustness

would be fruitful. Predicting drawing parameters in real-time is an open problem. Rendering the lines with learned pressure, texture, or thickness could make them match human drawings even more. Finally, our method does not handle point clouds, which would either require a mesh reconstruction step or learning to extract lines directly from unstructured point sets.

4.2 Neural Strokes: Line Drawing Stylization

The second section of this chapter discusses Neural Strokes, a model for producing stylized line drawings from 3D shapes [84] [2]. The model takes a 3D shape and a viewpoint as input, and outputs a drawing with textured strokes, with variations in stroke thickness, deformation, and color learned from an artist's style. The model is fully differentiable. We train its parameters from a single training drawing of another 3D shape. We show that, in contrast to previous image-based methods, the use of a geometric representation of 3D shape and 2D strokes allows the model to transfer important aspects of shape and texture style while preserving contours. Our method outputs the resulting drawing in a vector representation, enabling richer downstream analysis or editing in interactive applications.

4.2.1 Model

This subsection describes our stylized rendering architecture. Subsection 4.2.2 describes how to train the model from a single artist-drawn example. Our trained model (Figure 4.11) takes as input a 3D shape and a camera position and produces a stylized vector rendering $\mathbf{I}$, represented as a set of strokes, that is, curves with varying thickness and texture. This allows us to simulate the appearance of drawing media (pen, pencil, paint), and the ways artists vary pressure/thickness along strokes [45, 56].

[2] This work is published in the Proceedings of ICCV 2021.

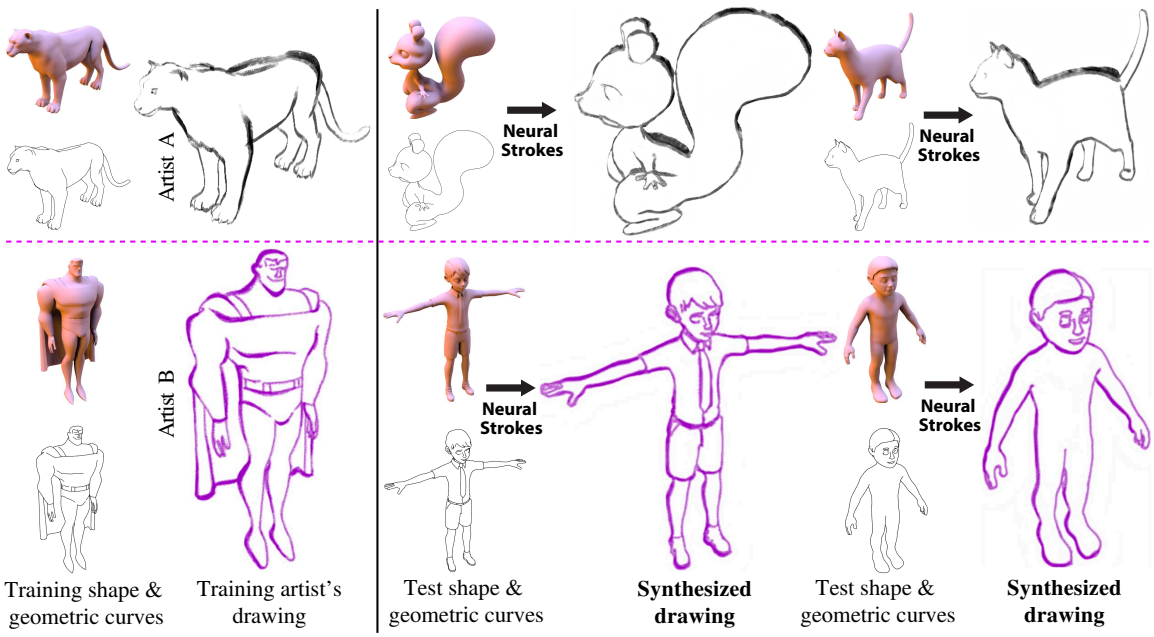

Figure 4.10: Our model learns to generate stylized line drawings from a single example of a training shape and corresponding drawing. Given a test 3D shape and 2D geometric curves representing the shape, our model synthesizes a line drawing in the style of the training example. Here we show synthesized drawings by transferring the artist's style A (top) or B (below).

4.2.1.1 Curve Extraction

The first stage of our model extracts geometric curves from the 3D shape, producing a set of curves $\mathbf{C}$. The goal of the rest of our model is to convert these plain curves into stylized strokes, by assigning thickness, displacement, and texture along these curves.

The curves are extracted using an existing algorithm for creating line drawings from 3D shapes. Many such methods have been developed [23], and our method can be used with any method that outputs vector curves. In our method, we extract curves using the pretrained Geometry Branch of Neural Contours (Section 4.1), which combines curves from many prior algorithms, including Occluding Contours [6], Suggestive Contours [24], Apparent Ridges [68], Ridges & Valleys [101].

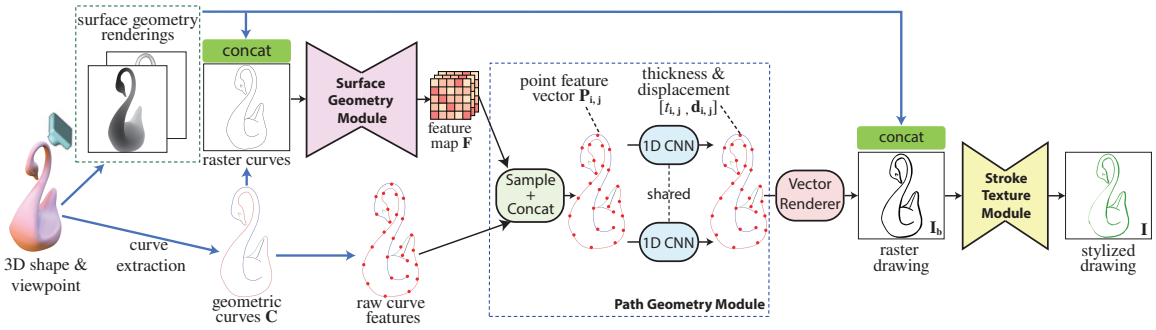

Figure 4.11: Our network architecture: the input 3D shape and a set of geometric curves are processed by a surface geometry module and a path geometry module to produce stroke thickness and displacement. With the predicted thickness and displacement, a stroke texture module creates a stylized line drawing with texture.

The geometric curves are represented as polylines: N vector paths $\mathbf{C} = \{\mathbf{c}_i\}_{i=1}^{N}$, where $\mathbf{c}_i$ is a sequence of densely sampled points $\mathbf{c}_i = \{\mathbf{c}_{i,j}\}_{j=1}^{M_i}$ with uniform spacing, M_i is the number of points on the path, and $\mathbf{c}_{i,j}$ represents the 2D position of point j on path i.

4.2.1.2 Stroke geometry prediction

The central portion of our model, described in this section, produces one stylized output stroke for each of the curves in $\mathbf{C}$. Texture synthesis is described in Section 4.2.1.3.

The geometric curves $\mathbf{C}$ are unstylized, and the goal of our model is to convert them to *strokes* with new shape, along with thickness and texture. The stroke control points are represented as displacements from the input curves. Displacement models the ways that artists deform curves, for example, smoothing curves, adding "wiggles" (e.g. Figure 4.21), and so on [30]. More precisely, for each input polyline $\mathbf{c}_i$, the module described in this section produces a 1D thickness $t_{i,j}$ for each control point $\mathbf{c}_{i,j}$, together with a displacement vector $\mathbf{d}_{i,j}$. Hence, the control points of the output stroke will be $\{\mathbf{c}_{i,j} + \mathbf{d}_{i,j}\}$ (Figure 4.12).

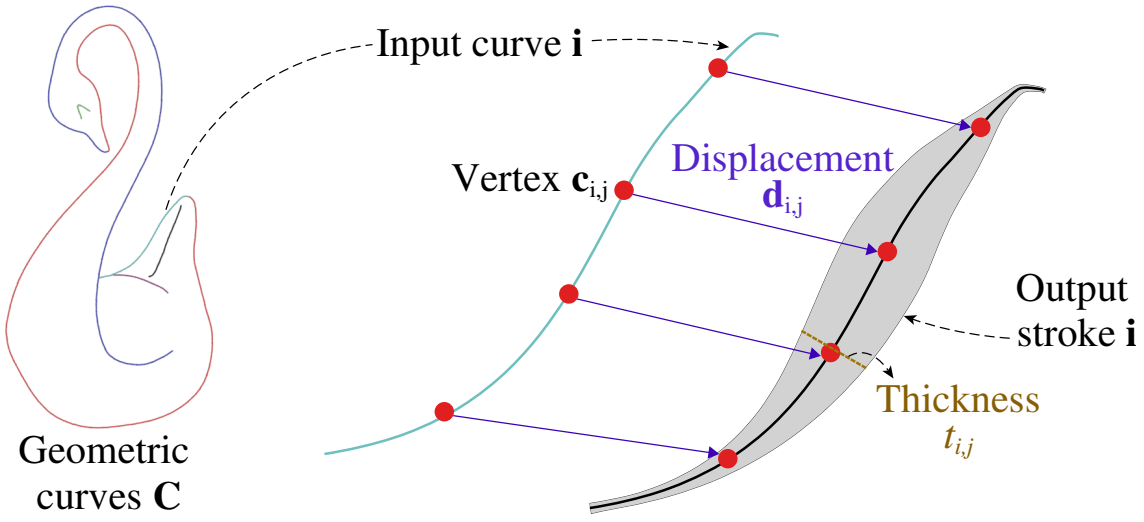

Figure 4.12: *Left:* an input set of geometric curves (each curve is highlighted with a different color). *Right:* For each input curve, our model outputs a stroke by predicting a thickness scalar and a 2D displacement vector for each control point.

As observed in previous work, artistic stroke thickness and displacement depend on image-space geometric shape features (e.g., object depth, view-dependent curvature, and surface shading [40]). Stroke thickness and displacement also depend on the shape of the stroke itself, including phenomena like tapering, stroke smoothing, and "wiggliness," (e.g., [41]), which can be captured as deformations of the 1D curve. Hence, to predict stroke geometry, the model includes modules to incorporate information from both the shape's surface geometry and along the 1D stroke paths.

Surface geometry module. First, the surface geometry module processes surface geometry via a 2D convolutional neural network, outputting image-space feature maps $\mathbf{F}$.

Surface geometry is represented in the form of image-space renderings. Each pixel contains the geometric properties of the surface point that projects to that pixel. There are nine input channels per pixel: depth from camera, radial curvature, derivative of radial curvature [24], maximum and minimum principal surface curvatures, view-dependent surface curvature [68], dot product of surface normal with view vec-

tor, and a raster image containing the line segments of the vector paths $\mathbf{C}$. We found these geometric and shading features to be useful for predicting accurate stroke geometry. In this manner, the module jointly processes shape features and vector paths in the concatenated $768 \times 768 \times 9$ map $\mathbf{V}$. The map passes through a neural network function to output a $768 \times 768 \times 40$ deep feature map $\mathbf{F} = f(\mathbf{V}; \mathbf{w}_1)$, where f is a ResNet-based fully convolutional network [65] with four residual blocks, and $\mathbf{w}_1$ are learned during training.

Path geometry module. The path geometry module is a neural network applied separately to each input curve using 1D convolutions. Each point $\{i, j\}$ on a curve has a set of curve features and features from the shape geometry.

The curve features are 2D curve normals, 2D tangent directions, and the normalized arc length. The normalized arc length allows the model to learn to taper stroke thickness, whereas the other two features can capture image-space curve orientations. Since the orientation of the curve is ambiguous, there is a sign ambiguity in the tangent direction $\mathbf{e}_{i,j}$ and normal $\mathbf{n}_{i,j}$ per curve point. To handle the ambiguity, we extract two alternative curve features sets: one using $(\mathbf{e}_{i,j}, \mathbf{n}_{i,j})$ and another set using $(-\mathbf{e}_{i,j}, -\mathbf{n}_{i,j})$.

In addition to the curve features, the deep surface geometry features $\mathbf{F}$ generated by the surface geometry module are also included as input to the path geometry module. Specifically, for each point on a curve, we use nearest interpolation on the deep feature map $\mathbf{F}$ to produce 40-dim features. These features are concatenated with each of the two sets of the above 5 raw curve features of the vector path, resulting in two $M_i \times 45$ feature maps $(\mathbf{P}_i, \mathbf{P}'_i)$ for the path i, where M_i is the number of control points in the path. In this manner, the module jointly processes view-based surface features together with geometric properties specific to the path. We also experimented with processing these features independently and found the above combination yielded the best performance. The above features pass through a neural network function to

predict the thickness and 2D displacement along each vector path:

$$[\mathbf{t}_i, \mathbf{d}_i] = \text{avg}\big(h(\mathbf{P}_i; \mathbf{w}_2), h(\mathbf{P}'_i; \mathbf{w}_2)\big) \tag{4.8}$$

where $\mathbf{d}_i = \{\mathbf{d}_{i,j}\}_{j=1}^{M_i}$ are the predicted per-point displacements, and $\mathbf{t}_i = \{t_{i,j}\}_{j=1}^{M_i}$ are per-point thicknesses (Figure 4.12), and $\mathbf{w}_2$ are parameters learned from the training reference drawing. The avg function performs average pooling over predictions of the two alternative feature sets to ensure invariance to the sign of curve orientation. The function h is a 1D fully convolutional network made of 3 layers, each using filters of kernel size 3, stride 1, zero padding. The first two layers are followed by ReLU activation. The last layer has 3 output channels: two for 2D displacement, and one for thickness. For thickness, we use a ReLU activation to guarantee non-negative outputs, while for the 2D real-valued displacement output, we do not use any non-linearity.

Differentiable vector renderer. Given the predicted displacement $\mathbf{d}_i$ for each vector path $\mathbf{c}_i$, new vector paths are formed as $\mathbf{c}'_i = \mathbf{c}_i + \mathbf{d}_i$. Using the differentiable vector graphics renderer DiffVG [80], these new vector paths are rasterized into grayscale polylines based on predicted thickness $\mathbf{t}_i$. Specifically, for each pixel in the output image, its distance to the closest point on the vector paths is computed. If it is smaller than half the stroke thickness of the closest point, the pixel is inside the stroke's area and assigned black color; otherwise it is marked as white. The strokes are rendered in a 768×768 raster image $\mathbf{I}_b$ with anti-aliasing provided by the differentiable renderer. The resulting image is grayscale, lacking texture (see Figure 4.11).

4.2.1.3 Stroke Texture

The final module predicts texture for all strokes. Texture may vary according to depth and underlying shape features, e.g., an artist may use darker strokes for

strong shape protrusions, and lighter strokes for lower-curvature regions. As a result, we condition the texture prediction not only on the raster drawing $\mathbf{I}_b$ representing the generated grayscale strokes, but also the shape representations used as input to the surface geometry module of Section 4.2.1.2. Specifically, we formulate texture prediction as a 2D image translation problem. The input to our image translation module are the first eight channels of the view-based features $\mathbf{V}$ (Section 4.2.1.2) concatenated with the raster drawing $\mathbf{I}_b$ channel-wise, resulting in a $768 \times 768 \times 9$ map $\mathbf{U}$. This map is translated into a RGB image $\mathbf{I} = g(\mathbf{U}; \mathbf{w}_3)$ where g is a ResNet-based fully convolutional network [65] with four residual blocks, and $\mathbf{w}_3$ are parameters learned during training.

As an optional post-processing step, to incorporate the predicted texture into our editable vector graphics representation, we convert the predicted RGB colors into a per stroke texture map. Specifically, each stroke is parameterized by a 2D u-v map, whose coordinates are used as a look-up table to access the texture map for each stroke. The color of each pixel in the stroke's texture map is determined by the RGB color of the corresponding pixel in the translated image $\mathbf{I}$.

4.2.1.4 Architecture Details

We provide here details of our network architecture.

Surface geometry module. Our surface geometry module uses the architecture shown in Table 4.8. All convolutional layers are followed by instance normalization [130] and a ReLU nonlinearity. The module contains 4 residual blocks [47], where each residual block contains two 3×3 convolutional layers with the same number of filters for both layers.

Path geometry module. Our path geometry module uses the architecture shown in Table 4.9. The first two convolutional layers are followed by a ReLU nonlinearity.

Layer	Activation size
Input	$768 \times 768 \times 9$
Conv2D(7x7, 9→10, stride=1)	$768 \times 768 \times 10$
Conv2D(3x3, 10→20, stride=2)	$384 \times 384 \times 20$
Conv2D(3x3, 20→40, stride=2)	$192 \times 192 \times 40$
4 Residual blocks	$192 \times 192 \times 40$
Conv2D(3x3, 40→40, stride=1/2)	$384 \times 384 \times 40$
Conv2D(3x3, 40→40, stride=1/2)	$768 \times 768 \times 40$
Conv2D(1x1, 40→40, stride=1)	$768 \times 768 \times 40$

Table 4.8: Architecture of the surface geometry module.

Layer	Activation size
Input	$M_i \times 45$
Conv1D(3x3, 45→40, stride=1)	$M_i \times 40$
Conv1D(3x3, 40→40, stride=1)	$M_i \times 40$
Conv1D(3x3, 40→3, stride=1)	$M_i \times 3$

Table 4.9: Architecture of the path geometry module.

The last layer has 3 output channels: two for 2D displacement, and one for thickness. For thickness, we use a ReLU activation to guarantee non-negative outputs, while for the 2D real-valued displacement output, we do not use any nonlinearity.

Stroke texture module. Our stroke texture module uses the architecture shown in Table 4.10. All convolutional layers are followed by instance normalization [130] and a ReLU nonlinearity except for the last convolutional layer. The last convolutional layer is followed by a sigmoid activation function. The module contains 6 residual blocks [47], where each residual block contains two 3×3 convolutional layers with the same number of filters for both layers.

4.2.2 Training

In order to train a model, we gather drawings made by artists based on rendered line drawings. Due to the difficulties in creating multiple drawings in a consistent

Layer	Activation size
Input	$768 \times 768 \times 9$
Conv2D(7x7, 9→64, stride=1)	$768 \times 768 \times 64$
Conv2D(3x3, 64→128, stride=2)	$384 \times 384 \times 128$
Conv2D(3x3, 128→256, stride=2)	$192 \times 192 \times 256$
6 Residual blocks	$192 \times 192 \times 256$
Conv2D(3x3, 256→128, stride=1/2)	$384 \times 384 \times 128$
Conv2D(3x3, 128→64, stride=1/2)	$768 \times 768 \times 64$
Conv2D(7x7, 64→3, stride=1)	$768 \times 768 \times 3$

Table 4.10: Architecture of the stroke texture module.

style, our training procedure is designed to work with a single training example alone. The goal of our training procedure is to learn the parameters $\mathbf{w} = \{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ of our surface geometry module, path geometry module, and stroke texture module described in Section 4.2.1.

Obtaining an artist's drawing. We provide an artist the feature curves $\mathbf{C}$ for shape, produced from a 3D shape using the procedure in Section 4.1. The artist is asked to produce a drawing on a digital tablet, using the provided curves as a reference. They are specifically instructed not to trace the feature curves, so that we can capture the artist's natural tendency to deform curve shape and thickness. Given the input training drawing $\hat{\mathbf{I}}$, a binary mask $\hat{\mathbf{I}}_b$ is extracted by assigning black for pixels containing the artist's strokes, and white for background. To smooth out discontinuities, anti-aliasing is applied to the mask, in the same manner as in the vector renderer, making it "soft" i.e., a grayscale image.

Note that, although we have paired drawings, i.e., input 3D geometry and a drawing, our method is not fully supervised, because the drawing is provided in raster format; we do not know the stroke thickness and displacement in the drawings. This makes our data collection more flexible, allowing different data sources and allowing artists to use their favorite drawing tools.

Losses. Training a network from a single drawing is prone to overfitting. To avoid this problem, the core idea of our training procedure is to crop several random patches from the artist's drawing capturing strokes at different locations and scales. Each of the sampled patches is treated as a separate training instance. We use only patches that contain strokes. During training, we sample patches on the fly. For each patch, we randomly choose a crop size c from a set of scales $\{64 \times 64, 128 \times 128, 192 \times 192, 256 \times 256\}$, crop all images and input feature maps accordingly.

We use four terms in our loss function. First, we evaluate the cropped grayscale image $\mathbf{I}_b^c$ produced by the vector graphics renderer, as compared to the cropped reference soft mask $\hat{\mathbf{I}}_b^c$, using L_1 loss:

$$\mathcal{L}_b = ||\mathbf{I}_b^c - \hat{\mathbf{I}}_b^c||_1 \tag{4.9}$$

Using the above loss alone, we found that the network sometimes end up generating implausible self-intersecting and noisy strokes. To handle this problem, we add a shape regularization term on the predicted displacements:

$$\mathcal{L}_s = \frac{1}{N^c} \sum_{i=1}^{N^c} \frac{1}{(M_i^c - 1)} \sum_{j=1}^{M_i^c - 1} ||\mathbf{d}_{i,j} - \mathbf{d}_{i,j+1}||^2 \tag{4.10}$$

where N^c is the number of vector paths in the cropped patch and M_i^c is the number of points on the path i.

We use L_1 loss in RGB space for texture, comparing a crop $\mathbf{I}^c$ from our predicted drawing and the corresponding crop $\hat{\mathbf{I}}^c$ from the artist's drawing $\hat{\mathbf{I}}$:

$$\mathcal{L}_t = ||\mathbf{I}^c - \hat{\mathbf{I}}^c||_1 \tag{4.11}$$

Finally, we use an adversarial loss to encourage the output patches to be visually similar to random patches from the artist's drawing. To this end, we add a discriminator $\mathcal{D}$ during training that is trained in parallel with the stroke texture module.

Architecturally, the discriminator $\mathcal{D}$ is identical to a 70×70 PatchGAN [61] with instance normalization, and it employs a standard LSGAN [96] discriminator loss. The output patches of our model are taken as *fake*, and random patches from the artist's drawing are taken as *real*. The patches are always selected to contain stroke pixels. We add the adversarial loss below to our stroke texture module by encouraging output patches to be classified as *real* by the discriminator $\mathcal{D}$:

$$\mathcal{L}_a = (\mathcal{D}(\mathbf{I}^c) - 1)^2 \tag{4.12}$$

Implementation details. We train the surface geometry and path geometry modules using $\lambda_b \mathcal{L}_b + \lambda_s \mathcal{L}_s$, and train the stroke texture module with $\lambda_t \mathcal{L}_t + \lambda_a \mathcal{L}_a$. The hyperparameters are set to the default values: $\lambda_b = 1, \lambda_s = 0.02, \lambda_t = 1, \lambda_a = 1$. For all three modules, we used the Adam optimizer [72] with learning rate set to 0.0002 and batch size 16.

4.2.3 Experiments

We evaluated our method both qualitatively and quantitatively. Below we discuss our dataset, evaluation metrics, comparisons with baseline methods, and our ablation study.

Dataset. To create our dataset, we collected 48 3D shapes from an online repository (TurboSquid [129]) and the ABC dataset [74], spanning several categories, including animals, humanoids, human body parts, clothes, and mechanical parts. All the 3D shapes are oriented and normalized so that the longest bounding box dimension is equal to 1. A camera position is selected for each 3D shape under the constraint that it is aligned with the upright axis and points towards the centroid of the mesh. For each 3D shape and selected camera, a set of 2D geometric curves is extracted automatically using the geometry branch of Neural Contours [85].

We hired 12 professional artists via UpWork to stylize the 2D plain line drawings of the 3D shapes. Each artist drew with 2 to 4 different styles, resulting in 31 total styles. For each style, the artist stylized 4 plain drawings representing 4 different 3D shapes, resulting in a total of 124 drawings. Specifically, through a web questionnaire, an artist is shown each of the four 3D shapes rendered in grayscale color using a frontal view and two side views. We also provide the artist with the plain line drawing for each of the 4 shapes. The artist is explicitly instructed to use a textured brush, change the shape and vary the thickness of their strokes as they see fit to achieve their preferred style, and be stylistically consistent for all 4 drawings. Since our model is trained in a single image setting, for each style, we randomly select one drawing as training and keep the other three for testing and evaluation.

Qualitative Results. Results of our method are shown in Figure 4.10 and Figure 4.13. As shown in the leftmost image in Figure 4.13, our method accurately transfers variations in stroke thickness from the turtle to the dinosaur, giving thicker strokes to low-curvature regions; strokes are also thicker on right-facing parts of the surface. The method also transfers the charcoal-like stroke texture. In the second example, our method accurately transfers the thin strokes, with stroke thickness often thicker around convex bulges.

More generalization cases. Figure 4.14 demonstrates challenging generalization cases: given a training drawing of a shape belonging to one category (e.g., humanoid), we synthesize a drawing for a shape from an entirely different category (e.g., mechanical object) in the same style. Our method still generalizes sufficiently in these challenging cases.

Evaluation metrics. To perform our evaluation, we compare synthesized test drawings with ones drawn by artists. We use the following metrics for evaluation: (1) **LPIPS** Learned Perceptual Image Patch Similarity [155] defined as a weighted

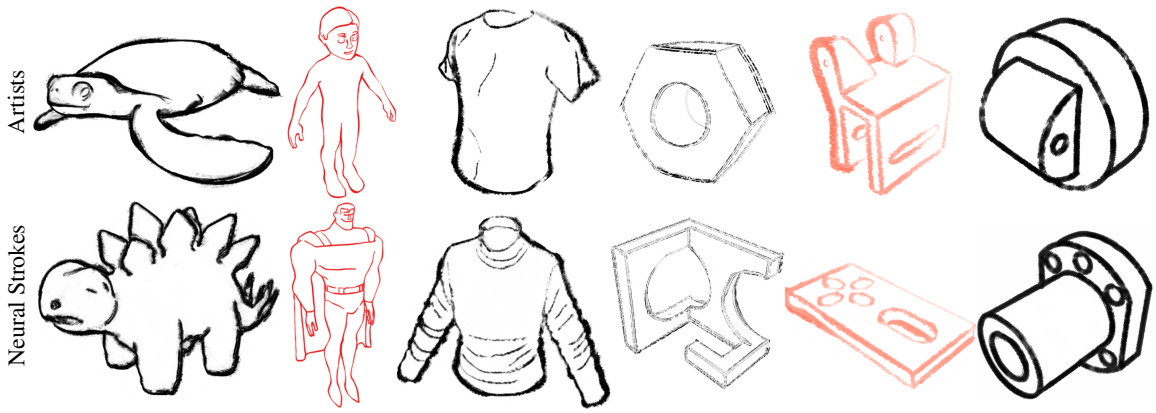

Figure 4.13: A gallery of our results. *Top:* artist-drawn training drawings. *Bottom:* drawings from Neural Strokes.

L_2 distance between learned deep features of images. The measure has been demonstrated to correlate well with human perceptual similarity [155]. We report LPIPS averaged over all test cases in our set across all 31 styles. (2) **FID** Frechet Inception Distance [52]. FID measures the distance between two set of images in terms of statistics on deep image features. In our evaluation, the set of images includes all the synthesized line drawings of our testing set, and we compare it with the set of artists' drawings.

Comparison methods. We compare our method, Neural Strokes, with several raster image stylization approaches that attempt to transfer the style from a single example image. **(1) *SketchPatch*** [32] is a paired image-to-image translation model that, like ours, operates on a patch level. During training, SketchPatch takes as input the plain line drawing patches and generates stylized line drawing patches. For a fair comparison, we also condition the SketchPatch model on the input shape representations of the surface geometry module (Section 4.2.1.2) by using them as additional input channels. We also experimented with using one SketchPatch model for stroke geometry prediction and another SketchPatch model for stroke texture prediction; yet, results did not improve. Thus, we show here the results from training

Figure 4.14: *Left to right:* training artist's drawing, test geometric curves, Neural Strokes.

Method	LPIPS ↓	FID ↓
SketchPatch	0.1104	83.60
SinCUT	0.1195	95.74
Bénard *et al.*[5]	0.1618	181.36
NST	0.2782	155.79
Neural Strokes	**0.0956**	**62.40**

Table 4.11: Numerical comparisons with other methods.

a single SketchPatch model for both. **(2) *SinCUT*** [103] is an unpaired image-to-image translation model designed to be trained from a single image. During training of SinCUT, random crops from the training drawing are used as training instances, as in our method. We also condition the SinCUT model on the input shape representations of the surface geometry module (Section 4.2.1.2) for a fair comparison. **(3) *NST*** [36] performs artistic style transfer by jointly minimizing a content loss and a style loss. Given a training drawing and a testing shape, we use the test plain line drawing as content image and the artist's training drawing as style image. **(4) Bénard *et al.*** [5] performs non-parametric line drawing stylization by copying pixel values from the artist's stylized drawing to corresponding pixels in the synthesized line drawing. The correspondence is optimized by PatchMatch [2], maximizing patch-level similarity between the reference and synthesized drawings. As pointed out by Bénard *et al.*[5], their "parameters are style specific;" i.e., one has to tune the parameters for each style. For purposes of comparison, we manually tuned their parameters to obtain the best results.

Results. Table 4.11 reports the evaluation measures for Neural Strokes and other competing methods. Based on the results, Neural Strokes outperforms all competing methods in terms of both LPIPS and FID. Figure 4.15 shows characteristic comparisons with competing methods. We also include the artists' drawings for the test shapes. Since other methods do not take explicit advantage of the input geometry,

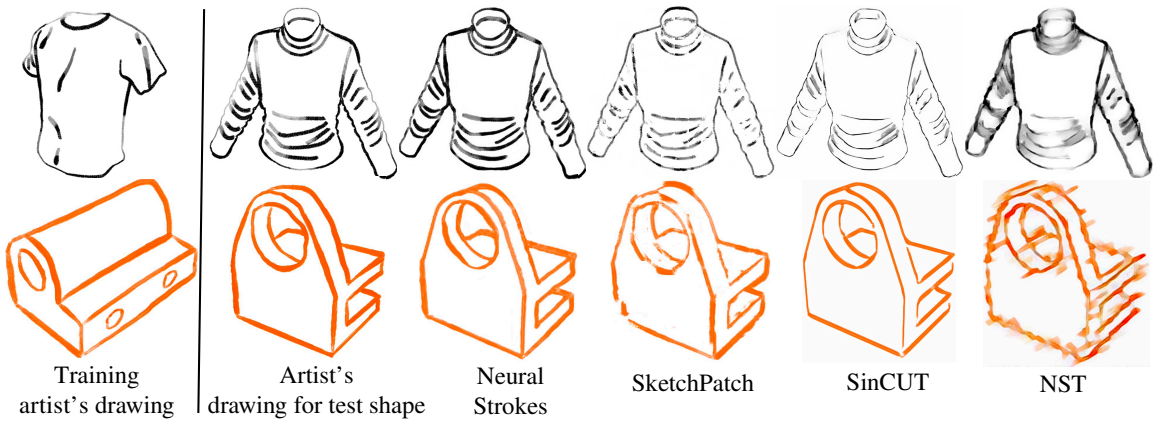

Figure 4.15: Comparisons with other methods. *Left to right:* training artist's drawing, artist's drawing for test shape, Neural Strokes, SketchPatch, SinCUT, NST result. Where possible, we retrained the other methods to incorporate the same geometry features present in the 3D shape as in our method. Our method produces strokes having more similar texture, intensity and thickness variation to the artist's drawing compared to other methods, which seem to miss the above style aspects.

except in terms of the feature maps that we provided, they often introduce gaps in strokes, or blur the stroke entirely. In the top row, observe that other methods do not accurately transfer stroke thicknesses from the example. Our method produces more precise stylized strokes, with fewer artifacts, agreeing with the artists' corresponding styles in terms of stroke thickness, shape, and texture.

Additional comparisons with Bénard *et al.*[5]. Figure 4.16 shows the training artist's drawing on the top, and results from Bénard *et al.*[5] in the bottom (zoom-in for details). They roughly capture the overall distribution of line properties, without matching the artist's choices well. Moreover, Bénard *et al.*[5] introduces many holes and cannot handle challenging cases, such as varying stroke thickness or large deformation (the rightmost style in Figure 4.16).

Additional comparisons with SketchPatch [32]. We experimented with using one SketchPatch model for stroke geometry prediction and another SketchPatch model for stroke texture prediction. Specifically, in the first step, we train a Sketch-

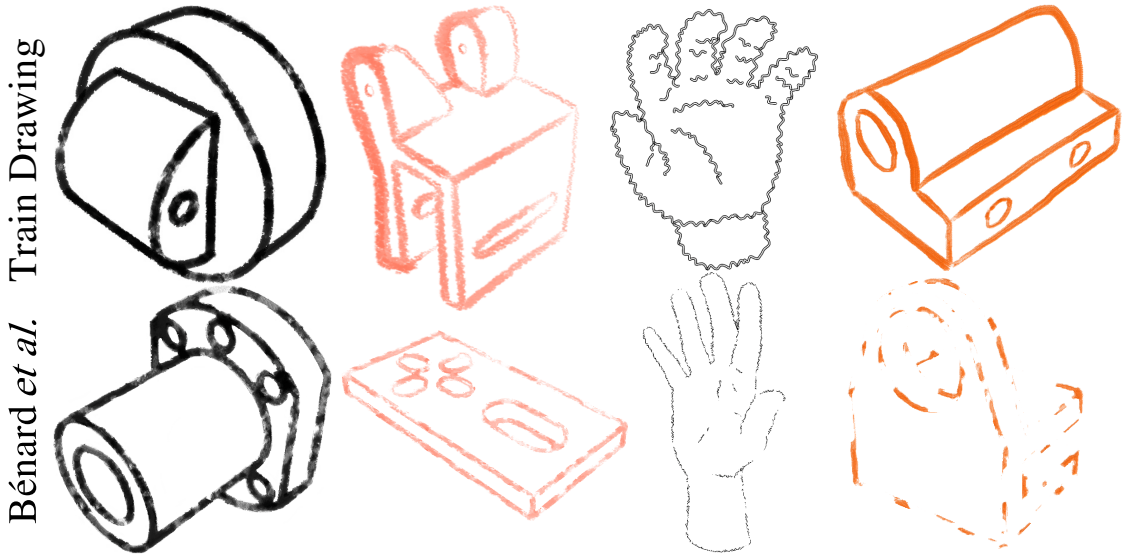

Figure 4.16: *Top:* artist-drawn training drawings. *Bottom:* results from Bénard *et al.*[5].

Method	LPIPS ↓	FID ↓
SketchPatch	0.1104	83.60
SketchPatch-texture	0.1142	86.96
Neural Strokes	**0.0956**	**62.40**

Table 4.12: Quantitative evaluation of SketchPatch variants.

Patch model (called *SketchPatch-geometry*) on the training stroke mask $\hat{\mathbf{I}}_b$ to predict stroke geometry as a grayscale raster image. In the second step, we train another SketchPatch model (called *SketchPatch-texture*) on the training drawing $\hat{\mathbf{I}}$ to generate a stylized line drawing given the output of *SketchPatch-geometry*. The results did not improve compared to *SketchPatch* in terms of our evaluation metrics (see Table 4.12). Figure 4.17 shows example output of *SketchPatch-geometry* and *SketchPatch-texture*.

User study. We also conducted an Amazon MTurk study as an additional perceptual evaluation. Each questionnaire page showed participants the stylized artist's drawing for the training shape, along with a randomly ordered pair of drawings: one

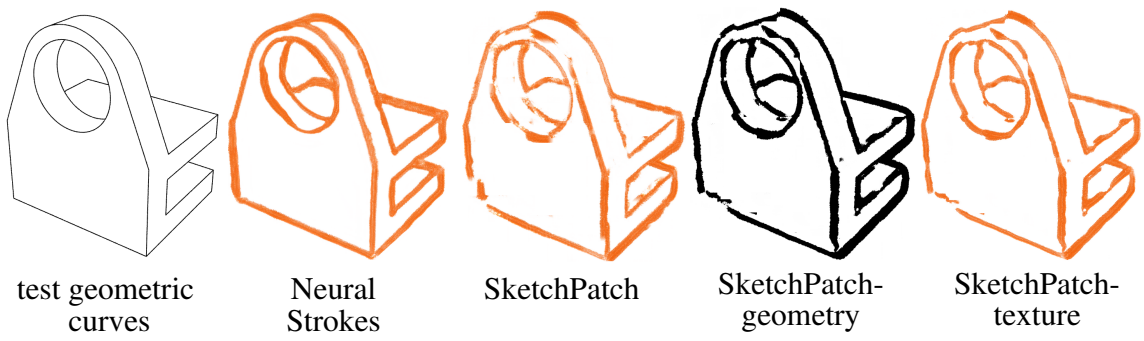

Figure 4.17: *Left to right:* test geometric curves, Neural Strokes, SketchPatch, SketchPatch-geometry, SketchPatch-texture result.

synthetic drawing from our method, and another from a different algorithm or from the same artist and style. We asked participants which drawing best mimicked the style of training drawing. Participants could pick either drawing, specify "none" or "both" drawings mimicked the training drawing equally well. We asked questions twice in a random order to verify participants' reliability. We had 93 reliable participants. Figure 4.18 summarizes the number of votes for the above options. The study shows that our method receives the most votes for better stylization compared to other methods, nearly seven times as many as the best alternative (SketchPatch). Moreover, our method receives similar number of votes with the artists' drawings. This indicates that our stylized drawings are comparable to artists' drawings.

In the Amazon Mechanical Turk perceptual evaluation where we showed participants (a) a stylized artist's drawing for a training shape (Figure 4.19, A) , (b) test geometric curves (Figure 4.19, B) , (c) a pair of stylized line drawings of the test shape placed in a randomized left/right position (Figure 4.19, X and Y): one line drawing was picked from our method, while the other came from *SketchPatch*, *SinCUT*, *NST*, Bénard *et al.*[5], or *Artists* (5 possible comparison cases). We asked participants to select the drawing that best mimicked the style of training drawing A. Participants could pick one of four options: drawing X, drawing Y, "neither of the drawings mimicked the style well", or "both drawings mimicked the style well". The study included

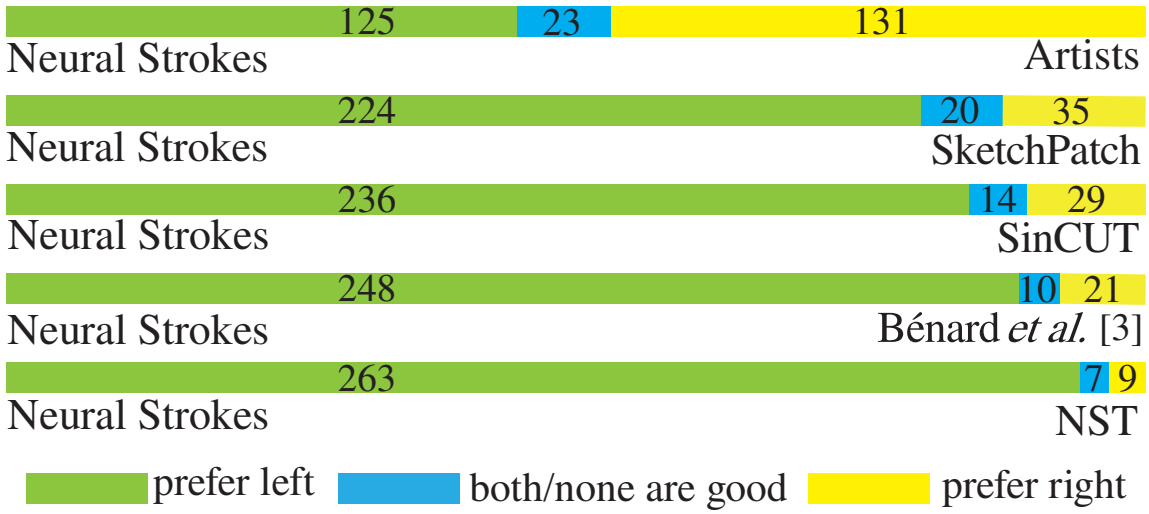

Figure 4.18: User study voting results.

the 31 styles from our dataset and each style consists of 3 test shapes. As a result, there were total 93 test cases, each involving the above-mentioned 5 comparisons (465 total comparisons).

Each questionnaire was released via the MTurk platform. It contained 15 unique questions, each asking for one comparison. Then these 15 questions were repeated in the questionnaire in a random order. In these repeated questions, the order of compared line drawings was flipped. If a worker gave more than 5 inconsistent answers for the repeated questions, then the worker was marked as "unreliable". Each participant was allowed to perform the questionnaire only once to ensure participant diversity. A total of 161 participants took part in the study. Among 161 participants, 68 workers were marked as "unreliable". For each of the 465 comparisons, we gathered votes from 3 different "reliable" users.

Ablation study. We also compare with the following reduced variants of our method. **(1)** ***No strokes:*** in this reduced variant, we remove the vector stroke representation from our method and use only the surface geometry module to predict

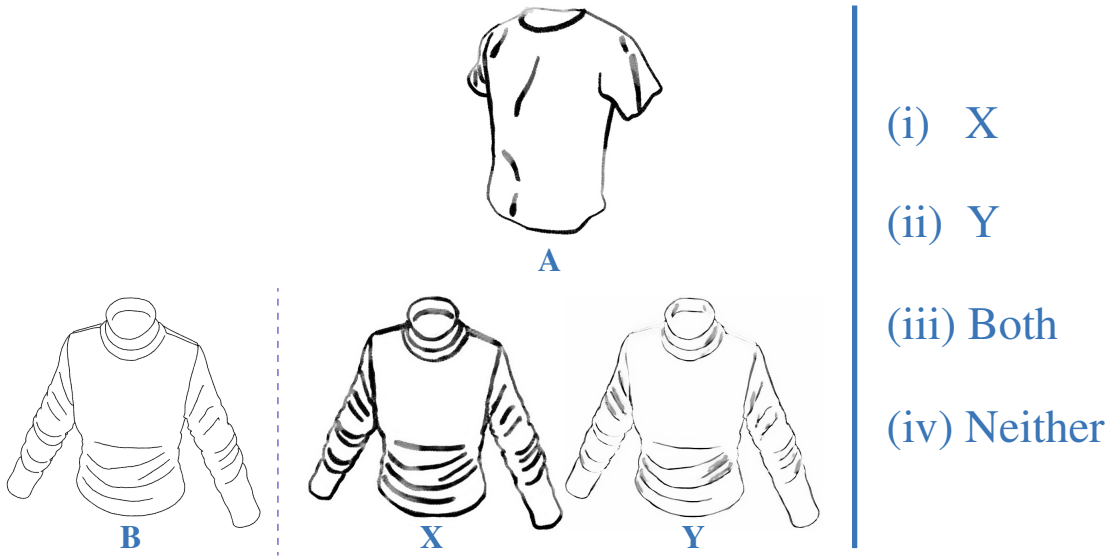

Figure 4.19: Layout shown to participants of our user study.

the stroke geometry as a raster image. Specifically, the surface geometry module (a 2D fully convolutional network) takes the map $\mathbf{V}$ as input and produces the raster drawing $\mathbf{I}_b$ directly without the use of the differentiable vector graphics renderer. Since this reduced variant does not predict thickness or displacement, we remove the displacement regularization $\mathcal{L}_s$ and only use $\mathcal{L}_b$ loss for the training of surface geometry module. **(2)** ***No curve features:*** we remove the raw curve features from our path geometry module. Specifically, we use the surface geometry module to produce a $768 \times 768 \times 3$ map, where each pixel contains the thickness and 2D displacement prediction. Then the stroke attributes predictions are propagated to the geometric curves directly without the use of raw curve features and our 1D CNN. **(3)** ***No surface features:*** we exclude the 3D shape features from our path geometry module by removing the $768 \times 768 \times 8$ surface geometry renderings from the input of the surface geometry module. We also remove them from the stroke texture module. **(4)** ***No multi-scale crops:*** during training, instead of randomly choosing a crop size from a set of scales, we use a fixed crop size 128×128 in this variant. **(5)** ***No regularization:*** we remove the displacement regularization $\mathcal{L}_s$ in this variant. For all these

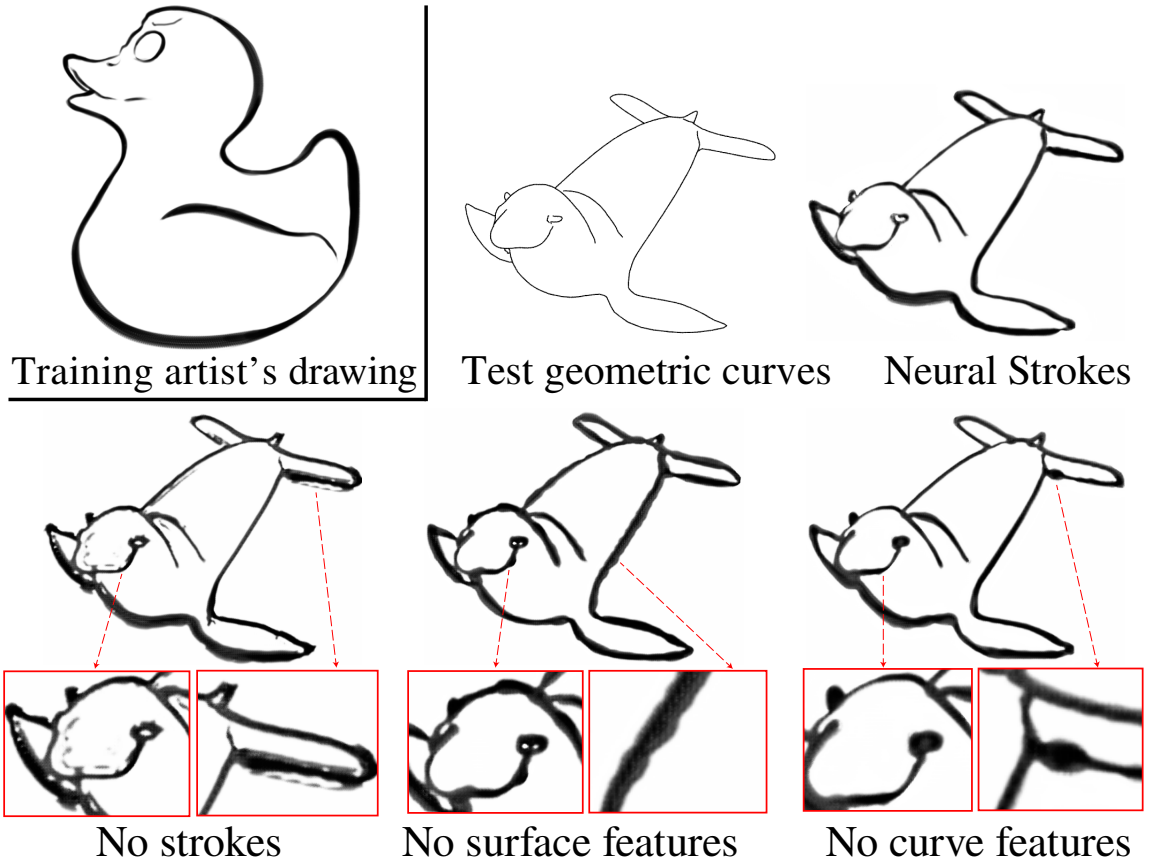

Figure 4.20: Comparisons with variants of our method. Removing features from our method result in noisy, incoherent strokes deviating from the training drawing style.

variants, the training and architecture of stroke texture module remain the same unless specified. Table 4.13 reports the evaluation measures for Neural Strokes and the abovementioned reduced variants. The reduced variants result in worse performance. Figure 4.20 shows characteristic comparisons with the reduced variants. We observe degraded results, especially in the case of "No strokes" where several broken, noisy strokes appear. In the case of "No surface features" and "No curve features", we observe incoherent strokes with unnatural thickness variation and deformation.

Intermediate results. Unlike purely raster image based methods that produce strokes as pixel values, Neural Strokes predict stroke attributes (thickness, displace-

Method	LPIPS ↓	FID ↓
No strokes	0.1082	75.65
No curve features	0.1006	67.74
No surface features	0.1022	72.86
No multi-scale crops	0.1056	73.70
No regularization	0.1107	71.73
Neural Strokes	**0.0956**	**62.40**

Table 4.13: Ablation study.

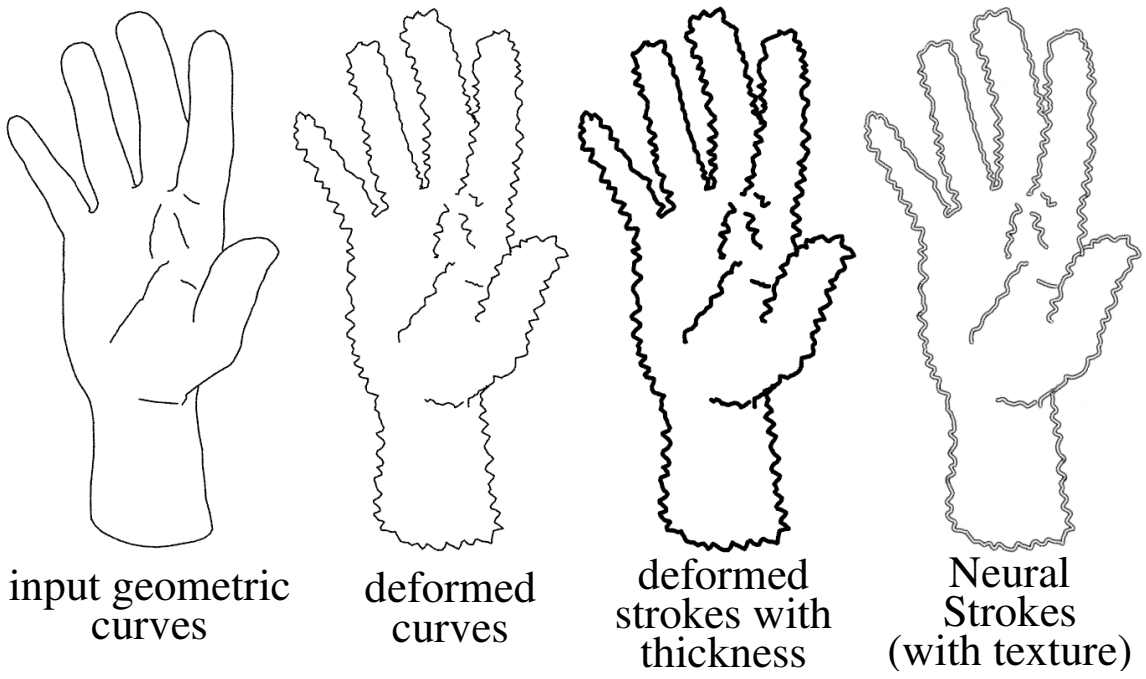

Figure 4.21: *Left to right:* Input geometric curves of a hand shape, deformed curves with predicted displacement from the path geometry module, strokes with predicted displacement and thickness from the path geometry module, final output textured strokes.

ment, texture) in intermediate stages that can be visualized separately. Figure 4.21 shows intermediate results of our method.

Vector Graphics editing. Since our method is able to output the stylized drawing in a vector representation, one can easily edit the strokes in vector graphics editing

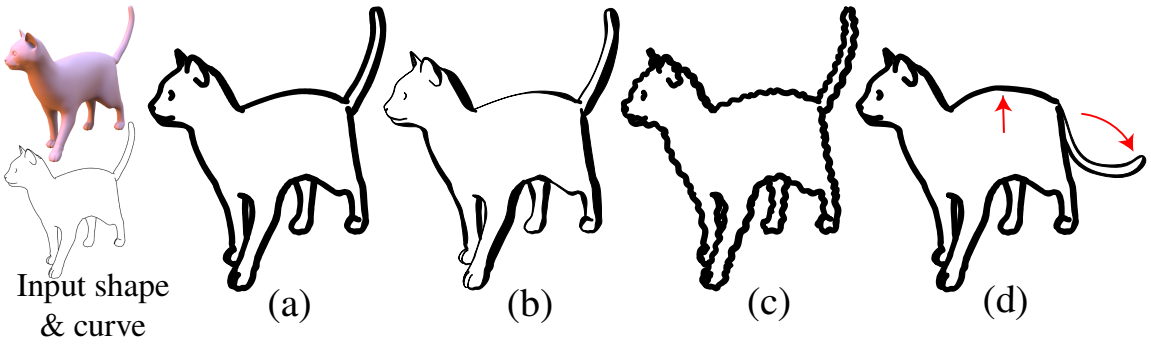

Figure 4.22: Given our output strokes (a) of a cat shape, we show three editing operations: (b) rescale thickness, (c) add wiggliness, (d) move control points of strokes.

applications. In Figure 4.22, we show three examples of vector editing operations on our output strokes: rescaling thickness, adding wiggliness, and move control points.

4.2.4 Summary

In this section, we presented a method that learns to stylize line drawings for 3D models by predicting stroke thickness, displacement and texture. The model is trained from a single raster drawing and produces output strokes in a vector graphics format. Our experiments demonstrate that our method significantly improves over existing image-based stylization methods and that our generated drawings are comparable to artists' drawing. There are still avenues for further improvements. An artist may sometimes vary the style within the same drawing, make random choices, or the style might be uncorrelated with any of the features we use. In this case, our result may not reproduce well such stylistic choices. In addition, when there is a large mismatch between input geometric curves and training drawing, the network may fail to reproduce correctly the stroke thickness and displacement. Learning to predict the correspondence between feature curves and the training drawing could help dealing with this issue. Learning to transfer style for other types of drawings from a single or few examples, such as hatching illustrations and cartoons, would also be another interesting research direction.

CHAPTER 5

CONCLUSION

The thesis explored two approaches to controllable neural image synthesis: a transformer-based model for synthesizing natural images with long-range interactions, and a convolutional neural network for vector art synthesis and stylization with the guidance of 3D shapes.

In Chapter 3, I presented an approach for automatically editing an input high-resolution image according to a user's edits on its semantic segmentation map. To obtain realistic and consistent editing results, an effective system needs to consider global context from across the full image. The proposed approach is based on transformers that are well equipped to handle the long-range dependencies through their attention mechanism allowing them to focus on distant image areas. To avoid paying prohibitive computational costs of the attention operation at high resolutions, I proposed an adaptively sparsified attention mechanism. The core idea is to efficiently determine a small list of relevant locations that are worth attending to, and compute the attention map only over these locations. This leads to a large reduction in computational cost due to the highly sparse attention matrix. Compared to the state-of-the-art, the proposed approach can obtain more realistic and consistent edits while achieving high diversity in the outputs.

In Chapter 4, I presented a system for synthesizing editable vector drawings from 3D shapes. The proposed system consists of two stages. In the first stage, I combine traditional geometric algorithms and modern neural architectures to produce a state-of-the-art line drawing algorithm. A neural ranking network is trained to assess

the plausibility of the output drawing. To maximize the plausibility during testing, the parameters of geometric algorithms are optimized through a novel differentiable renderer for line drawings. In the second stage, the plain vector drawing from the first stage is converted to a stylized vector drawing. The resulting drawings are in the format of vector strokes. The style of vector drawings is represented by stroke attributes including thickness, deformation and texture, capturing a variety of styles such as wiggly strokes and spatially varying thickness. To learn the stylization of vector art from raster supervision, our model leverages a differentiable vector renderer to propagate the gradient from raster mages to stroke attributes. Moreover, the proposed model is trained on a single training drawing by using the multi-scale cropping strategy. The better performance of the proposed system is demonstrated by comparing to the state-of-the-art methods and baselines via both numerical experiments and perceptual user studies.

5.1 Future Work

Although this thesis tackled significant challenges in creating high-quality content in both the natural and artistic domain via exploiting model-efficient and data-efficient methods, there are still many exciting directions for future work.

5.1.1 Diverse Synthesis of 3D Data and Video

In Chapter 3, the proposed method modeled long-range interactions in high-resolution images. A promising future work direction is to investigate transformers for long-term video editing and high-resolution 3D shape synthesis. In video editing, long-range dependencies are important to capture, e.g., editing of an object in a keyframe may need to be propagated to the same object in a distant frame. In 3D shape synthesis, the architectural style of a door in a building should be consistent with other distant parts of the building, e.g., windows, balconies, and so on. More-

over, many tasks for 3D data and video are inherently multi-modal. The proposed sparsified attention mechanism in Chapter 3 provides new insights for avoiding high computational cost in transformers. To capture long-range dependencies in 3D data and video, it will be useful to investigate more efficient representations and attention mechanisms, such as coordinated-based representations and softmax-free attention with linear complexity.

5.1.2 Generative Modeling of Vector Art

In Chapter 4, the vector drawing is generated from a 3D shape. Apart from creating vector art from 3D shapes, there are many other interesting and challenging scenarios, such as converting raster images to vector art, as well as unconditional generation of vector art.

Vector art can represent a diverse range of objects and scenes in a non-photorealistic style, involving different number of drawing curves and attributes, which can not easily modeled by VAEs or GANs. Recently, diffusion models have achieved wide success in raster image generation, outperforming GANs in fidelity and diversity, without training instability and mode collapse issues. The advantages of diffusion models can also benefit generative modeling of vector graphics that tend to be quite diverse. However, diffusion models for vector graphics generation remain largely unexplored. There are several aspects that require further research: (i) neural representations of vector graphics, (ii) the design of an effective denoising architecture, (iii) the conditioning mechanism to fuse raster and curve representations.

An effective diffusion model for generative modeling of vector art can lead to more creative applications such as translation of raster images into vector graphics (see preliminary results in Figure 5.1), automatic vector graphics completion, vector drawing beautification, and so on.

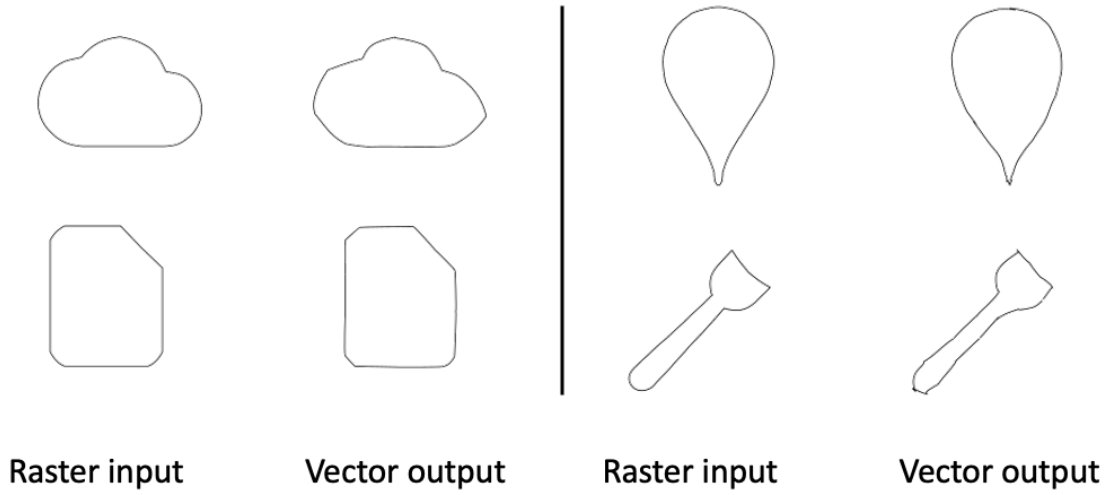

Figure 5.1: *Preliminary results:* translation of raster images into vector graphics with a diffusion model based on [55] operating on control points of the curves.

5.1.3 Image Editing with Sketches

Sketching (Chapter 4) represents a more expressive and editable guidance for image editing as opposed to semantic painting (Chapter 3). While a lot of progress has been made in sketch-based image editing, there are still significant challenges. First, these approaches are often trained on synthetic sketches in a single style. This leads to poor generalization on real sketches that are usually drawn in various styles. Second, sketches often convey the structure of the underlying 3D world, whereas most approaches regard them as flat images. Leveraging users' implicit guidance often requires understanding the underlying 3D structure of the input sketches. I believe the combination of 3D-aware sketch stylization (Chapter 4) and transformer-based image synthesis (Chapter 3) is a promising future work direction for image editing with sketches.

BIBLIOGRAPHY

[1] Abnar, Samira, and Zuidema, Willem. Quantifying attention flow in transformers. *arXiv preprint arXiv:2005.00928* (2020).

[2] Barnes, Connelly, Shechtman, Eli, Finkelstein, Adam, and Goldman, Dan B. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3 (2009).

[3] Bau, David, Strobelt, Hendrik, Peebles, William, Wulff, Jonas, Zhou, Bolei, Zhu, Jun-Yan, and Torralba, Antonio. Semantic photo manipulation with a generative image prior. *arXiv preprint arXiv:2005.07727* (2020).

[4] Beltagy, Iz, Peters, Matthew E, and Cohan, Arman. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[5] Bénard, Pierre, Cole, Forrester, Kass, Michael, Mordatch, Igor, Hegarty, James, Senn, Martin Sebastian, Fleischer, Kurt, Pesare, Davide, and Breeden, Katherine. Stylizing animation by example. *ACM Trans. Graph. 32*, 4 (2013).

[6] Bénard, Pierre, and Hertzmann, Aaron. Line drawings from 3D models. *Foundations and Trends in Computer Graphics and Vision 11*, 1-2 (2019).

[7] Caesar, Holger, Uijlings, Jasper, and Ferrari, Vittorio. Coco-stuff: Thing and stuff classes in context. In *CVPR* (2018), pp. 1209–1218.

[8] Cao, Chenjie, Hong, Yuxin, Li, Xiang, Wang, Chengrong, Xu, Chengming, Xue, XiangYang, and Fu, Yanwei. The image local autoregressive transformer. In *NeurIPS* (2021).

[9] Chan, Caroline, Durand, Frédo, and Isola, Phillip. Learning to generate line drawings that convey geometry and semantics. In *Proc. CVPR* (2022).

[10] Chang, Angel X, et al. Shapenet: An information-rich 3d model repository. arXiv:1512.03012, 2015.

[11] Chen, Jiawen, Adams, Andrew, Wadhwa, Neal, and Hasinoff, Samuel W. Bilateral guided upsampling. *ACM Trans. Graph. 35*, 6 (nov 2016).

[12] Chen, Liang-Chieh, Papandreou, George, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence 40*, 4 (2017), 834–848.

[13] Chen, Mark, Radford, Alec, Child, Rewon, Wu, Jeffrey, Jun, Heewoo, Luan, David, and Sutskever, Ilya. Generative pretraining from pixels. In *icml* (2020), pp. 1691–1703.

[14] Chen, Qifeng, and Koltun, Vladlen. Photographic image synthesis with cascaded refinement networks. In *iccv* (2017), pp. 1511–1520.

[15] Cheng, Yu, Gan, Zhe, Li, Yitong, Liu, Jingjing, and Gao, Jianfeng. Sequential attention gan for interactive image editing. In *Proceedings of the 28th ACM International Conference on Multimedia* (2020), pp. 4383–4391.

[16] Child, Rewon, Gray, Scott, Radford, Alec, and Sutskever, Ilya. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).

[17] Choi, Jooyoung, Kim, Sungwon, Jeong, Yonghyun, Gwon, Youngjune, and Yoon, Sungroh. Ilvr: Conditioning method for denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 14367–14376.

[18] Chu, Xiangxiang, Tian, Zhi, Wang, Yuqing, Zhang, Bo, Ren, Haibing, Wei, Xiaolin, Xia, Huaxia, and Shen, Chunhua. Twins: Revisiting the design of spatial attention in vision transformers. *arXiv preprint arXiv:2104.13840 1*, 2 (2021), 3.

[19] Chu, Xiangxiang, Tian, Zhi, Zhang, Bo, Wang, Xinlong, Wei, Xiaolin, Xia, Huaxia, and Shen, Chunhua. Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882* (2021).

[20] Cole, Forrester, Golovinskiy, Aleksey, Limpaecher, Alex, Barros, Heather Stoddart, Finkelstein, Adam, Funkhouser, Thomas, and Rusinkiewicz, Szymon. Where do people draw lines? *ACM Trans. Graph. 27*, 3 (2008).

[21] Cole, Forrester, Sanik, Kevin, DeCarlo, Doug, Finkelstein, Adam, Funkhouser, Thomas, Rusinkiewicz, Szymon, and Singh, Manish. How well do line drawings depict shape? *ACM Trans. Graph. 28*, 3 (2009).

[22] Coleman, Patrick, Murphy, Laura, Kranzler, Markus, and Gilbert, Max. Making souls: Methods and a pipeline for volumetric characters. In *SIGGRAPH Talks* (2020).

[23] DeCarlo, Doug. Depicting 3d shape using lines. In *Proc. SPIE Human Vision and Electronic Imaging XVII* (2012).

[24] DeCarlo, Doug, Finkelstein, Adam, Rusinkiewicz, Szymon, and Santella, Anthony. Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3 (2003).

[25] Dehghani, Mostafa, Zamani, Hamed, Severyn, Aliaksei, Kamps, Jaap, and Croft, W. Bruce. Neural ranking models with weak supervision. In *Proc. SIGIR* (2017).

[26] Dhamo, Helisa, Farshad, Azade, Laina, Iro, Navab, Nassir, Hager, Gregory D, Tombari, Federico, and Rupprecht, Christian. Semantic image manipulation using scene graphs. In *cvpr* (2020), pp. 5213–5222.

[27] Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[28] Esser, Patrick, Rombach, Robin, Blattmann, Andreas, and Ommer, Björn. Imagebart: Bidirectional context with multinomial diffusion for autoregressive image synthesis. In *NeurIPS* (2021).

[29] Esser, Patrick, Rombach, Robin, and Ommer, Bjorn. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 12873–12883.

[30] Fankbonner, Edgar Loy. *Art of Drawing the Human Body*. Inc. Sterling Publishing Co., 2004.

[31] Ferstl, David, Reinbacher, Christian, Ranftl, Rene, Ruether, Matthias, and Bischof, Horst. Image guided depth upsampling using anisotropic total generalized variation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (December 2013).

[32] Fish, Noa, Perry, Lilach, Bermano, Amit, and Cohen-Or, Daniel. Sketchpatch: sketch stylization via seamless patch-level synthesis. *ACM Trans. Graph. 39*, 6 (2020).

[33] Fišer, Jakub, Jamriška, Ondřej, Lukáč, Michal, Shechtman, Eli, Asente, Paul, Lu, Jingwan, and Sýkora, Daniel. Stylit: Illumination-guided example-based stylization of 3d renderings. *ACM Trans. Graph. 35*, 4 (2016).

[34] Ganin, Yaroslav, Kulkarni, Tejas, Babuschkin, Igor, Eslami, SM Ali, and Vinyals, Oriol. Synthesizing programs for images using reinforced adversarial learning. In *Proc. ICML* (2018).

[35] Gastal, Eduardo SL, and Oliveira, Manuel M. Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30*, 4 (2011).

[36] Gatys, Leon A, Ecker, Alexander S, and Bethge, Matthias. Image style transfer using convolutional neural networks. In *Proc. CVPR* (2016).

[37] Gombrich, E. H. *Art and Illusion: A Study in the Psychology of Pictorial Representation*. Princeton U. Press, 1961.

[38] Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. *Advances in neural information processing systems 27* (2014).

[39] Goodman, Nelson. *Languages of Art: An Approach to a Theory of Symbols*. Bobbs-Merrill Company, 1968.

[40] Goodwin, Todd, Vollick, Ian, and Hertzmann, Aaron. Isophote distance: A shading approach to artistic stroke thickness. In *Proc. NPAR* (2007).

[41] Grabli, Stéphane, Turquin, Emmanuel, Durand, Frédo, and Sillion, François X. Programmable rendering of line drawing from 3d scenes. *ACM Trans. Graph. 29*, 2 (2010).

[42] Gryaditskaya, Yulia, Hähnlein, Felix, Liu, Chenxi, Sheffer, Alla, and Bousseau, Adrien. Lifting freehand concept sketches into 3d. *ACM Trans. Graph. 39*, 6 (2020).

[43] Gryaditskaya, Yulia, Sypesteyn, Mark, Hoftijzer, Jan Willem, Pont, Sylvia, Durand, Frédo, and Bousseau, Adrien. Opensketch: A richly-annotated dataset of product design sketches. *ACM Trans. Graph. 38*, 6 (2019).

[44] Gu, Shuyang, Bao, Jianmin, Yang, Hao, Chen, Dong, Wen, Fang, and Yuan, Lu. Mask-guided portrait editing with conditional gans. In *cvpr* (2019), pp. 3436–3445.

[45] Guptill, Arthur Leighton. *Rendering in Pen and Ink*. Watson-Guptill Publications, 1997.

[46] Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer, Bernhard, Reutemann, Peter, and Witten, Ian H. The weka data mining software: An update. *SIGKDD Explor. Newsl. 11*, 1 (2009).

[47] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proc. CVPR* (2016).

[48] Herbrich, R., Graepel, T., and Obermayer, K. Support vector learning for ordinal regression. In *Proc. ICANN* (1999).

[49] Hertzmann, Aaron. Why do line drawings work? a realism hypothesis. *Perception 49*, 4 (2020).

[50] Hertzmann, Aaron, Jacobs, Charles E., Oliver, Nuria, Curless, Brian, and Salesin, David H. Image analogies. In *Proc. SIGGRAPH* (2001).

[51] Hertzmann, Aaron, Oliver, Nuria, Curless, Brian, and Seitz, Steven M. Curve analogies. In *Proceedings of the 13th Eurographics Workshop on Rendering* (2002).

[52] Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard, and Hochreiter, Sepp. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS 30* (2017).

[53] Hinz, Tobias, Fisher, Matthew, Wang, Oliver, and Wermter, Stefan. Improved techniques for training single-image gans. In *wacv* (2021), pp. 1300–1309.

[54] Hinz, Tobias, Heinrich, Stefan, and Wermter, Stefan. Generating multiple objects at spatially distinct locations. In *ICLR* (2019).

[55] Ho, Jonathan, Jain, Ajay, and Abbeel, Pieter. Denoising diffusion probabilistic models. In *Proc. NeurIPS* (2020).

[56] Hodges, Elaine. *The Guild Handbook of Scientific Illustration*. Wiley, 2003.

[57] Holtzman, Ari, Buys, Jan, Du, Li, Forbes, Maxwell, and Choi, Yejin. The curious case of neural text degeneration. In *iclr* (2019).

[58] Hong, Seunghoon, Yan, Xinchen, Huang, Thomas, and Lee, Honglak. Learning hierarchical semantic image manipulation through structured representations. In *NeurIPS* (2018), pp. 2713–2723.

[59] Huang, Xun, and Belongie, Serge. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proc. ICCV* (2017).

[60] Hui, Tak-Wai, Loy, Chen Change, and Tang, Xiaoou. Depth map super-resolution by deep multi-scale guidance. In *Proceedings of European Conference on Computer Vision (ECCV)* (2016).

[61] Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* (2017).

[62] Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR* (2017).

[63] Jiang, Yifan, Chang, Shiyu, and Wang, Zhangyang. Transgan: Two transformers can make one strong gan. *arXiv preprint arXiv:2102.07074* (2021).

[64] Jo, Youngjoo, and Park, Jongyoul. Sc-fegan: Face editing generative adversarial network with user's sketch and color. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 1745–1753.

[65] Johnson, Justin, Alahi, Alexandre, and Fei-Fei, Li. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV* (2016).

[66] Jones, Thouis R., Durand, Frédo, and Desbrun, Mathieu. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph. 22*, 3 (2003).

[67] Jongejan, J., Rowley, H., Kawashima, T., Kim, J., and Fox-Gieg, N. The quick, draw dataset. https://quickdraw.withgoogle.com/, 2016.

[68] Judd, Tilke, Durand, Frédo, and Adelson, Edward. Apparent ridges for line drawing. *ACM Trans. Graph.* (2007).

[69] Kalnins, Robert D., Markosian, Lee, Meier, Barbara J., Kowalski, Michael A., Lee, Joseph C., Davidson, Philip L., Webb, Matthew, Hughes, John F., and Finkelstein, Adam. Wysiwyg npr: Drawing strokes directly on 3d models. In *Proc. SIGGRAPH* (2002).

[70] Kalogerakis, Evangelos, Nowrouzezahrai, Derek, Simari, Patricio, McCrae, James, Hertzmann, Aaron, and Singh, Karan. Data-driven curvature for real-time line drawing of dynamic scene. *ACM Trans. Graph. 28*, 1 (2009).

[71] Kennedy, John M. *A Psychology of Picture Perception: Images and Information*. Jossey-Bass Publishers, 1974.

[72] Kingma, Diederik P, and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[73] Kitaev, Nikita, Kaiser, Lukasz, and Levskaya, Anselm. Reformer: The efficient transformer. In *ICLR* (2020).

[74] Koch, Sebastian, Matveev, Albert, Jiang, Zhongshi, Williams, Francis, Artemov, Alexey, Burnaev, Evgeny, Alexa, Marc, Zorin, Denis, and Panozzo, Daniele. Abc: A big cad model dataset for geometric deep learning. In *Proc. CVPR* (2019).

[75] Koenderink, Jan J, and van Doorn, Andrea J. The shape of smooth objects and the way contours end. *Perception 11*, 2 (1982).

[76] Kopf, Johannes, Cohen, Michael F, Lischinski, Dani, and Uyttendaele, Matt. Joint bilateral upsampling. *ACM Transactions on Graphics (ToG) 26*, 3 (2007), 96–es.

[77] Lee, Cheng-Han, Liu, Ziwei, Wu, Lingyun, and Luo, Ping. Maskgan: Towards diverse and interactive facial image manipulation. In *cvpr* (2020), pp. 5549–5558.

[78] Lee, Yunjin, Markosian, Lee, Lee, Seungyong, and Hughes, John F. Line drawings via abstracted shading. *ACM Trans. Graph. 26*, 3 (2007).

[79] Lewis, Mike, Liu, Yinhan, Goyal, Naman, Ghazvininejad, Marjan, Mohamed, Abdelrahman, Levy, Omer, Stoyanov, Ves, and Zettlemoyer, Luke. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[80] Li, Tzu-Mao, Lukáč, Michal, Gharbi, Michaël, and Ragan-Kelley, Jonathan. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. 39*, 6 (2020).

[81] Li, Yijun, Fang, Chen, Hertzmann, Aaron, Shechtman, Eli, and Yang, Ming-Hsuan. Im2pencil: Controllable pencil illustration from photographs. In *Proc. CVPR* (2019).

[82] Li, Yijun, Fang, Chen, Yang, Jimei, Wang, Zhaowen, Lu, Xin, and Yang, Ming-Hsuan. Universal style transfer via feature transforms. In *Proc. NeurIPS* (2017).

[83] Ling, Huan, Kreis, Karsten, Li, Daiqing, Kim, Seung Wook, Torralba, Antonio, and Fidler, Sanja. Editgan: High-precision semantic image editing. In *NeurIPS* (2021).

[84] Liu, Difan, Fisher, Matthew, Hertzmann, Aaron, and Kalogerakis, Evangelos. Neural strokes: Stylized line drawing of 3d shapes. In *Proc. ICCV* (2021).

[85] Liu, Difan, Nabail, Mohamed, Hertzmann, Aaron, and Kalogerakis, Evangelos. Neural contours: Learning to draw lines from 3d shapes. In *Proc. CVPR* (2020).

[86] Liu, Difan, Shetty, Sandesh, Hinz, Tobias, Fisher, Matthew, Zhang, Richard, Park, Taesung, and Kalogerakis, Evangelos. Asset: Autoregressive semantic scene editing with transformers at high resolutions. *ACM Trans. Graph. 41*, 4 (2022).

[87] Liu, Guilin, Reda, Fitsum A, Shih, Kevin J, Wang, Ting-Chun, Tao, Andrew, and Catanzaro, Bryan. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 85–100.

[88] Liu, Hongyu, Wan, Ziyu, Huang, Wei, Song, Yibing, Han, Xintong, and Liao, Jing. Pd-gan: Probabilistic diverse gan for image inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 9371–9381.

[89] Liu, Hongyu, Wan, Ziyu, Huang, Wei, Song, Yibing, Han, Xintong, Liao, Jing, Jiang, Bin, and Liu, Wei. Deflocnet: Deep image editing via flexible low-level controls. In *cvpr* (2021), pp. 10765–10774.

[90] Liu, Ming-Yu, Tuzel, Oncel, and Taguchi, Yuichi. Joint geodesic upsampling of depth images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2013), pp. 169–176.

[91] Liu, Ze, Lin, Yutong, Cao, Yue, Hu, Han, Wei, Yixuan, Zhang, Zheng, Lin, Stephen, and Guo, Baining. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030* (2021).

[92] Loshchilov, Ilya, and Hutter, Frank. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[93] Lu, Jingwan, Barnes, Connelly, DiVerdi, Stephen, and Finkelstein, Adam. Realbrush: Painting with examples of physical media. *ACM Trans. Graph. 32*, 4 (2013).

[94] Lu, Jingwan, Yu, Fisher, Finkelstein, Adam, and DiVerdi, Stephen. Helpinghand: Example-based stroke stylization. *ACM Trans. Graph. 31*, 4 (2012).

[95] Lutio, Riccardo de, D'aronco, Stefano, Wegner, Jan Dirk, and Schindler, Konrad. Guided super-resolution as pixel-to-pixel transformation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 8829–8837.

[96] Mao, Xudong, Li, Qing, Xie, Haoran, Lau, Raymond YK, Wang, Zhen, and Paul Smolley, Stephen. Least squares generative adversarial networks. In *Proc. ICCV* (2017).

[97] Meng, Chenlin, Song, Yang, Song, Jiaming, Wu, Jiajun, Zhu, Jun-Yan, and Ermon, Stefano. Sdedit: Image synthesis and editing with stochastic differential equations. In *Proc. ICLR* (2022).

[98] Nam, Seonghyeon, Kim, Yunji, and Kim, Seon Joo. Text-adaptive generative adversarial networks: manipulating images with natural language. In *NeurIPS* (2018), pp. 42–51.

[99] Ntavelis, Evangelos, Romero, Andrés, Kastanis, Iason, Van Gool, Luc, and Timofte, Radu. Sesame: semantic editing of scenes by adding, manipulating or erasing objects. In *eccv* (2020), pp. 394–411.

[100] Ohtake, Yutaka, Belyaev, Alexander, and Seidel, Hans-Peter. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3 (2004).

[101] Ohtake, Yutaka, Belyaev, Alexander, and Seidel, Hans-Peter. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3 (2004).

[102] Park, Jaesik, Kim, Hyeongwoo, Tai, Yu-Wing, Brown, Michael S., and Kweon, Inso. High quality depth map upsampling for 3d-tof cameras. In *2011 International Conference on Computer Vision* (2011), pp. 1623–1630.

[103] Park, Taesung, Efros, Alexei A., Zhang, Richard, and Zhu, Jun-Yan. Contrastive learning for unpaired image-to-image translation. In *Proc. ECCV* (2020).

[104] Park, Taesung, Liu, Ming-Yu, Wang, Ting-Chun, and Zhu, Jun-Yan. Semantic image synthesis with spatially-adaptive normalization. In *cvpr* (2019), pp. 2337–2346.

[105] Park, Taesung, Zhu, Jun-Yan, Wang, Oliver, Lu, Jingwan, Shechtman, Eli, Efros, Alexei, and Zhang, Richard. Swapping autoencoder for deep image manipulation. In *Proc. NeurIPS* (2020).

[106] Parmar, Niki, Vaswani, Ashish, Uszkoreit, Jakob, Kaiser, Lukasz, Shazeer, Noam, Ku, Alexander, and Tran, Dustin. Image transformer. In *icml* (2018), PMLR, pp. 4055–4064.

[107] Patashnik, Or, Wu, Zongze, Shechtman, Eli, Cohen-Or, Daniel, and Lischinski, Dani. Styleclip: Text-driven manipulation of stylegan imagery. In *ICCV* (2021), pp. 2085–2094.

[108] Phong, Bui Tuong. Illumination for computer generated pictures. *Commun. ACM 18*, 6 (1975).

[109] Ramesh, Aditya, Dhariwal, Prafulla, Nichol, Alex, Chu, Casey, and Chen, Mark. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* (2022).

[110] Ramesh, Aditya, Pavlov, Mikhail, Goh, Gabriel, Gray, Scott, Voss, Chelsea, Radford, Alec, Chen, Mark, and Sutskever, Ilya. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092* (2021).

[111] Resource, VG. The models resource, https://www.models-resource.com/, 2019.

[112] Rusinkiewicz, Szymon. Estimating curvatures and their derivatives on triangle meshes. In *Proc. 3DPVT* (2004).

[113] Rusinkiewicz, Szymon, and DeCarlo, Doug. rtsc library. http://www.cs.princeton.edu/gfx/proj/sugcon/, 2007.

[114] Saharia, Chitwan, Chan, William, Saxena, Saurabh, Li, Lala, Whang, Jay, Denton, Emily, Ghasemipour, Seyed Kamyar Seyed, Ayan, Burcu Karagol, Mahdavi, S Sara, Lopes, Rapha Gontijo, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487* (2022).

[115] Sangkloy, Patsorn, Burnell, Nathan, Ham, Cusuh, and Hays, James. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Trans. Graph. 35*, 4 (2016).

[116] Sayim, Bilge, and Cavanagh, Patrick. What line drawings reveal about the visual brain. *Frontiers in Human Neuroscience 5* (2011).

[117] Shaham, Tamar Rott, Dekel, Tali, and Michaeli, Tomer. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 4570–4580.

[118] Shaham, Tamar Rott, Gharbi, Michaël, Zhang, Richard, Shechtman, Eli, and Michaeli, Tomer. Spatially-adaptive pixelwise networks for fast image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 14882–14891.

[119] Sharma, Gopal, Goyal, Rishabh, **Difan Liu**, Kalogerakis, Evangelos, and Maji, Subhransu. Csgnet: Neural shape parser for constructive solid geometry. In *Proc. CVPR* (2018).

[120] Sharma, Gopal, Goyal, Rishabh, **Difan Liu**, Kalogerakis, Evangelos, and Maji, Subhransu. Neural shape parsers for constructive solid geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[121] Sharma, Gopal, **Difan Liu**, Maji, Subhransu, Kalogerakis, Evangelos, Chaudhuri, Siddhartha, and Měch, Radomír. Parsenet: A parametric surface fitting network for 3d point clouds. In *Proc. ECCV* (2020).

[122] Shocher, Assaf, Cohen, Nadav, and Irani, Michal. "zero-shot" super-resolution using deep internal learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 3118–3126.

[123] Su, Sitong, Gao, Lianli, Zhu, Junchen, Shao, Jie, and Song, Jingkuan. Fully functional image manipulation using scene graphs in a bounding-box free way. In *Proceedings of the 29th ACM International Conference on Multimedia* (2021), pp. 1784–1792.

[124] Suvorov, Roman, Logacheva, Elizaveta, Mashikhin, Anton, Remizova, Anastasia, Ashukha, Arsenii, Silvestrov, Aleksei, Kong, Naejin, Goka, Harshith, Park, Kiwoong, and Lempitsky, Victor. Resolution-robust large mask inpainting with fourier convolutions. *arXiv preprint arXiv:2109.07161* (2021).

[125] Tan, Zhentao, Chai, Menglei, Chen, Dongdong, Liao, Jing, Chu, Qi, Liu, Bin, Hua, Gang, and Yu, Nenghai. Diverse semantic image synthesis via probability distribution modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 7962–7971.

[126] Tay, Yi, Dehghani, Mostafa, Bahri, Dara, and Metzler, Donald. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732* (2020).

[127] Thibault, Aaron, and Cavanaugh, Sean. Making concept art real for borderlands. In *ACM SIGGRAPH 2010 Courses* (2010).

[128] Tulsiani, Shubham, and Gupta, Abhinav. Pixeltransformer: Sample conditioned signal generation. *arXiv preprint arXiv:2103.15813* (2021).

[129] turbosquid. Turbosquid, https://www.turbosquid.com/, 2021.

[130] Ulyanov, Dmitry, Vedaldi, Andrea, and Lempitsky, Victor. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (2016).

[131] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Lukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[132] Vinker, Yael, Horwitz, Eliahu, Zabari, Nir, and Hoshen, Yedid. Image shape manipulation from a single augmented training sample. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 13769–13778.

[133] Wan, Ziyu, Zhang, Jingbo, Chen, Dongdong, and Liao, Jing. High-fidelity pluralistic image completion with transformers. *arXiv preprint arXiv:2103.14031* (2021).

[134] Wang, Sinong, Li, Belinda Z, Khabsa, Madian, Fang, Han, and Ma, Hao. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).

[135] Wang, Ting-Chun, Liu, Ming-Yu, Zhu, Jun-Yan, Tao, Andrew, Kautz, Jan, and Catanzaro, Bryan. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proc. CVPR* (2018).

[136] Wang, Wenhai, Xie, Enze, Li, Xiang, Fan, Deng-Ping, Song, Kaitao, Liang, Ding, Lu, Tong, Luo, Ping, and Shao, Ling. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122* (2021).

[137] Wang, Xiaolong, Girshick, Ross, Gupta, Abhinav, and He, Kaiming. Non-local neural networks. In *cvpr* (2018), pp. 7794–7803.

[138] Wang, Zhou, Bovik, Alan C, Sheikh, Hamid R, and Simoncelli, Eero P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing 13*, 4 (2004), 600–612.

[139] Wilber, Michael J, Kwak, Iljung S, and Belongie, Serge J. Cost-effective hits for relative similarity comparisons. In *Proc. HCOMP* (2014).

[140] Winkenbach, Georges, and Salesin, David H. Computer-generated pen-and-ink illustration. In *Proc. SIGGRAPH* (1994).

[141] Winkenbach, Georges, and Salesin, David H. Rendering parametric surfaces in pen and ink. In *Proc. SIGGRAPH* (1996).

[142] Yang, Chao, Lu, Xin, Lin, Zhe, Shechtman, Eli, Wang, Oliver, and Li, Hao. High-resolution image inpainting using multi-scale neural patch synthesis. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 6721–6729.

[143] Yang, Jianwei, Li, Chunyuan, Zhang, Pengchuan, Dai, Xiyang, Xiao, Bin, Yuan, Lu, and Gao, Jianfeng. Focal self-attention for local-global interactions in vision transformers. In *NeurIPS* (2021).

[144] Yang, Jingyu, Ye, Xinchen, Li, Kun, Hou, Chunping, and Wang, Yao. Color-guided depth recovery from rgb-d data using an adaptive autoregressive model. *IEEE Transactions on Image Processing 23* (2014), 3443–3458.

[145] Yang, Qingxiong, Yang, Ruigang, Davis, James, and Nister, David. Spatial-depth super resolution for range images. In *2007 IEEE Conference on Computer Vision and Pattern Recognition* (2007), pp. 1–8.

[146] Yi, Li, Huang, Haibin, **Difan Liu**, Kalogerakis, Evangelos, Su, Hao, and Guibas, Leonidas. Deep part induction from articulated object pairs. *ACM Trans. Graph. 37*, 6 (2019).

[147] Yu, Jiahui, Lin, Zhe, Yang, Jimei, Shen, Xiaohui, Lu, Xin, and Huang, Thomas S. Generative image inpainting with contextual attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 5505–5514.

[148] Yu, Jiahui, Lin, Zhe, Yang, Jimei, Shen, Xiaohui, Lu, Xin, and Huang, Thomas S. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 4471–4480.

[149] Yu, Tao, Guo, Zongyu, Jin, Xin, Wu, Shilin, Chen, Zhibo, Li, Weiping, Zhang, Zhizheng, and Liu, Sen. Region normalization for image inpainting. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020).

[150] Yu, Yingchen, Zhan, Fangneng, Wu, Rongliang, Pan, Jianxiong, Cui, Kaiwen, Lu, Shijian, Ma, Feiying, Xie, Xuansong, and Miao, Chunyan. Diverse image inpainting with bidirectional and autoregressive transformers. *arXiv preprint arXiv:2104.12335* (2021).

[151] Zaheer, Manzil, Guruganesh, Guru, Dubey, Kumar Avinava, Ainslie, Joshua, Alberti, Chris, Ontanon, Santiago, Pham, Philip, Ravula, Anirudh, Wang, Qifan, Yang, Li, et al. Big bird: Transformers for longer sequences. In *NeurIPS* (2020).

[152] Zhang, Pan, Zhang, Bo, Chen, Dong, Yuan, Lu, and Wen, Fang. Cross-domain correspondence learning for exemplar-based image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 5143–5153.

[153] Zhang, Pengchuan, Dai, Xiyang, Yang, Jianwei, Xiao, Bin, Yuan, Lu, Zhang, Lei, and Gao, Jianfeng. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. *arXiv preprint arXiv:2103.15358* (2021).

[154] Zhang, Richard, Isola, Phillip, Efros, Alexei A, Shechtman, Eli, and Wang, Oliver. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR* (2018), pp. 586–595.

[155] Zhang, Richard, Isola, Phillip, Efros, Alexei A, Shechtman, Eli, and Wang, Oliver. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR* (2018).

[156] Zheng, Chuanxia, Cham, Tat-Jen, and Cai, Jianfei. Pluralistic image completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 1438–1447.

[157] Zheng, Haitian, Lin, Zhe, Lu, Jingwan, Cohen, Scott, Zhang, Jianming, Xu, Ning, and Luo, Jiebo. Semantic layout manipulation with high-resolution sparse attention. *arXiv preprint arXiv:2012.07288* (2020).

[158] Zhou, Bolei, Zhao, Hang, Puig, Xavier, Fidler, Sanja, Barriuso, Adela, and Torralba, Antonio. Scene parsing through ade20k dataset. In *CVPR* (2017), pp. 633–641.

[159] Zhou, Qingnan, and Jacobson, Alec. Thingi10k: A dataset of 10,000 3d-printing models. arXiv:1605.04797, 2016.

[160] Zhou, Xingran, Zhang, Bo, Zhang, Ting, Zhang, Pan, Bao, Jianmin, Chen, Dong, Zhang, Zhongfei, and Wen, Fang. Cocosnet v2: Full-resolution correspondence learning for image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 11465–11475.

[161] Zhu, Jun-Yan, Park, Taesung, Isola, Phillip, and Efros, Alexei A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on* (2017).

[162] Zhu, Jun-Yan, Park, Taesung, Isola, Phillip, and Efros, Alexei A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. ICCV* (2017).

[163] Zhu, Jun-Yan, Zhang, Zhoutong, Zhang, Chengkai, Wu, Jiajun, Torralba, Antonio, Tenenbaum, Josh, and Freeman, Bill. Visual object networks: Image generation with disentangled 3d representations. *Advances in Neural Information Processing Systems 31* (2018), 118–129.

[164] Zhu, Peihao, Abdal, Rameen, Qin, Yipeng, and Wonka, Peter. Sean: Image synthesis with semantic region-adaptive normalization. In *cvpr* (2020), pp. 5104–5113.