Im2Vec: Synthesizing Vector Graphics without Vector Supervision

Pradyumna Reddy[1] Michaël Gharbi[2] Michal Lukáč[2] Niloy J. Mitra[1,2]

[1]University College London [2]Adobe Research

Abstract

Vector graphics are widely used to represent fonts, logos, digital artworks, and graphic designs. But, while a vast body of work has focused on generative algorithms for raster images, only a handful of options exists for vector graphics. One can always rasterize the input graphic and resort to image-based generative approaches, but this negates the advantages of the vector representation. The current alternative is to use specialized models that require explicit supervision on the vector graphics representation at training time. This is not ideal because large-scale high-quality vector-graphics datasets are difficult to obtain. Furthermore, the vector representation for a given design is not unique, so models that supervise on the vector representation are unnecessarily constrained. Instead, we propose a new neural network that can generate complex vector graphics with varying topologies, and only requires indirect supervision from readily-available raster *training images (i.e., with no vector counterparts). To enable this, we use a differentiable rasterization pipeline that renders the generated vector shapes and composites them together onto a raster canvas. We demonstrate our method on a range of datasets, and provide comparison with state-of-the-art SVG-VAE and DeepSVG, both of which require explicit vector graphics supervision. Finally, we also demonstrate our approach on the MNIST dataset, for which no groundtruth vector representation is available. Source code, datasets and more results are available at* http://geometry.cs.ucl.ac.uk/projects/2020/Im2Vec/.

1. Introduction

In vector graphics, images are represented as collections of parametrised shape primitives rather than a regular raster of pixel values. This makes for a compact, infinitely scalable representation with appearance that may be varied at need simply by modifying stroke or colour parameters. As a result, it is favoured by graphic artists and designers.

Unfortunately, creating vector graphics still remains a difficult task largely limited to manual expert workflows,

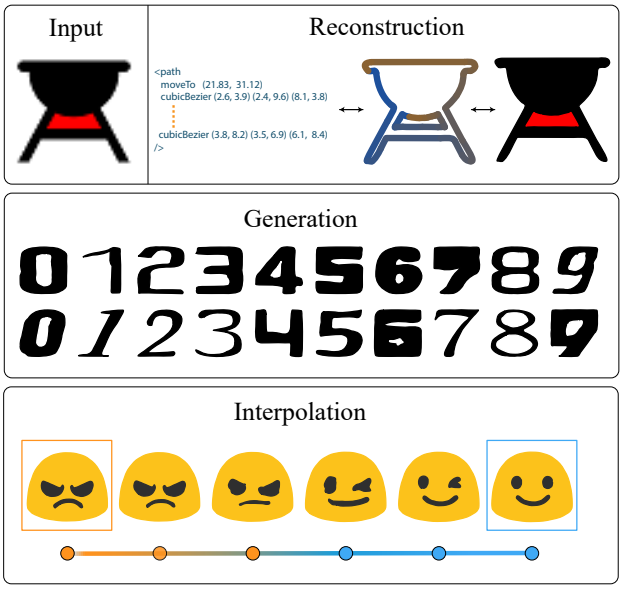


Figure 1: We present Im2Vec that can be trained with only image supervision to produce a latent space for vector graphics output. The learned space supports reprojection, sampling (i.e., generation), and interpolation.

because the same irregular structure makes it ill-suited for today's convolution-based generative neural architectures. There is demand for a generative approach suitable for this domain, but it is not yet well served by research because of the difficult design requirements. Suitable approaches should: (i) produce output in vector format; (ii) establish correspondence across elements of the same family; (iii) support reconstruction, sampling, and interpolation; (iv) give user control over accuracy versus compactness of the representation; and finally, (v) be trainable directly using images without the need for vector supervision.

SVG-VAE [24] and DeepSVG [5], the two leading generative algorithms for vector graphics, cast synthesis as a sequence prediction problem, where the graphic is a sequence of drawing instructions, mimicking how common formats actually represent vector art. Training these methods therefore requires supervision from ground truth vector graphics

sequences, which are difficult to collect in large volumes. Furthermore, the mapping from sequences of parametrised drawing instruction to actual images is highly non-linear with respect to the parameters and also non-injective, allowing a variety of different sequences to produce the same visual result. This makes it difficult to consider appearance as a criterion, and also causes the produced results to inherit any structural bias baked into the training sequences.

An approach aiming to do away with such vector supervision would need to overcome a number of challenges. First, the relationship between the representation and its appearance must be made explicit and differentiable. Second, it must operate on an internal representation that directly maps to a vector graphics representation and is flexible enough to support a large range of topologies and shape complexities. Finally, it should extract correspondences between related shapes, directly from unlabelled images.

In this paper, we propose such a method, called Im2Vec, based on a representation that mimics the compositing behaviour of complex vector graphics. It uses a variable-complexity closed Bézier path as the fundamental primitive, with the capability to composite a variable number of these to create shapes of arbitrary complexity and topology (shown in Figure 2).

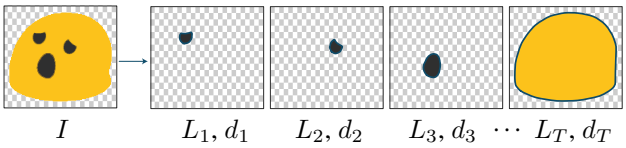


Figure 2: Im2Vec encodes a shape as a layered set of filled curves (or shapes). Each shape is obtained by deformation of a topological disk, differentiably rasterized into images L_i, then differentiably composited back-to-front according to scalar depth variables d_i.

The key insight that allows the handling of arbitrary complexity is that we can treat any primitive closed shape as a deformation of a unit circle, which is modelled as 1D convolution on samples from this circle conditioned on a common latent vector. By recombining these primitive paths through a differentiable rasterizer [22] and differentiable compositing [28], we can natively represent vector art while learning to generate it purely based on appearance, obviating the need for vector supervision.

We evaluate Im2Vec on a variety of examples with varying complexity and topology including fonts, emojis, and icons. We demonstrate that Im2Vec, even without any vector supervision, consistently performs better reconstruction compared to SVG-VAE and DeepSVG when trained on the same dataset. We also compare our approach to a purely raster-based autoencoder, which we dub ImageVAE. While ImageVAE and Im2Vec produce comparable reconstruction quality, Im2Vec outputs vector graphics and hence enjoys the associated editability and compactness benefits. Finally, we quantify the compactness versus approximation power of our method, and demonstrate Im2Vec can be used to vectorize the MNIST dataset for which no groundtruth vector representation is available.

2. Related Work

Deep learning techniques for parametric vector shapes have recently garnered significant interest from the machine learning community [19, 11, 13, 40, 27].

Learning-based image vectorization. Our autoencoder encodes raster images. It can therefore address the single-image vectorization problem [3, 9, 31, 20, 1, 17], for which learning-based solutions have been proposed. Egiazarian et al. [7] vectorize technical line drawings. They predict the parameters of vector primitives using a transformer-based network, and refine them by optimization. DeepSpline [11] produces parametric curves of variable lengths from images using a pre-trained VGG network [33] for feature extraction followed by a hierarchical recurrent network. Guo et al. [14] use neural networks sub-divide line drawings and reconstruct the local topology at line junctions. The network predictions are used in a least squares curve fitting step to estimate Bézier curve parameters. Liu et al. [23] focus on vectorization of rasterized floorplans. They use a network to extract and label wall junctions, and use this information to solve an integer program that outputs the vectorized floor plans as a set of architectural primitives. These works produce high-quality vectorizations but, unlike ours, focus on the single image case. In contrast, our objective is to train a latent representation which can serve both for vectorization of existing raster images, and for generating new graphics by sampling with no post-processing.

Parametric shape estimation. Deep learning methods for parametric shape estimation typically encode shapes as an assembly of primitives, often with fixed topology and cardinality [13]. Smirnov et al. [36] fit rasterized fonts using quadratic Bézier curves, and 3D signed distance fields using cuboids. Their outputs have predetermined, fixed topologies that are specified as class-dependent templates. Zou et al. [41] train a recurrent network that predict shapes as a collection of cuboids from depth maps; they supervise directly on the shape parameters. Tulsiani et al [39] also use hierarchies of cuboids, but from occupancy volumes. Similar techniques have explored other primitives like superquadrics [27] and Coon patches [35] as primitives. Sinha et al. [34] represents watertight 3D shapes as continuous deformation of a sphere. This is analogous to our representation of closed 2D curves.

Shape-generating programs. Ganin et al. [10], Huang et al. [18], and Nakano [25] train Reinforcement Learning (RL) drawing agents. They circumvent the need for direct supervision on the drawing program by simulating a rendering engine to produce images from which they compute a reward signal. Ellis et al. [8] use program synthesis to generate graphics expressed using a subset of the LaTeX language from hand drawings. They do not work with complex parametric shapes like Bézier curves, which are the basic building block of most vector designs. Another notable work is the CSGNet [32] that present impressive performance in estimating constructive solid geometry programs. It uses the REINFORCE [37] algorithm to learn in an unsupervised manner, but runs into issues like drawing over previous predictions in the later stages of the generation process. Further, it can only output 32×32 raster images, which lacks the flexibility of vector graphics and is insufficient for applications that require high fidelity. Strokenet [40] trains an agent that draws strokes after observing a canvas image and a generator that maps stroke parameters to a new image.

Generative vector graphics model. Our goal is to obtain a generative model for vector graphics. Previous works in this area have focused predominantly on the case where direct vector supervision is available. In contrast, our model can be trained from raster data alone. SketchRNN [15] introduces a model for both conditional and unconditional sketch generation. Sketches are encoded as a sequence of pen position and on/off states. An LSTM is then trained to predict the parameters of a density function over the sketch parameter space, which can then be sampled to produce a new sketches. Similarly, Sketchformer [29] proposed a transformer based architecture for encoding vector form sketches. They show how the encoding can be used for sketch classification, image retrieval, and interpolation.

SVG-VAE [24] is the first method that attempts to estimate vector graphics parameters for generative tasks. They follow a two stage training process. First, they train an image Variational Auto Encoder (VAE). Second, they freeze the VAE's weights and train a decoder that predicts vector parameters from the latent variable learned on images. They show a style-transfer application from one vector graphic to another. Unlike ours, their method is not end-to-end, and it requires vector supervision. More recently, DeepSVG [5] showed that models operating on vector graphics benefit from a hierarchical architecture; they demonstrate interpolation and generation tasks. Prior works [2, 12] can generate new font glyphs from partial observations, but they only work in a low-resolution raster domain. Li et al. [22] have recently proposed a differentiable rasterizer that enables gradient based optimization and learning on vector graphics, using raster-based objectives. This is a key building block for our method. However, we go beyond the generative models they demonstrate. In particular, our network can generate graphics made up of closed curves with complex and varying topologies; it does not produce artifacts like overlapping paths.

3. Method

Our goal is to build a generative model for vector graphics that does not require vector supervision, i.e., that only requires raster images at training time. Our model follows an encoder–decoder architecture (Fig. 3). The encoder has a standard design [16]; it maps a raster image I to a latent variable $z \in \mathbb{R}^d$, which is then decoded into a vector graphic structure. Our decoder has been carefully designed so that it can generate complex graphics, made of a variable number T of paths, with varying lengths and no predetermined topology (§ 3.1). We also train an auxiliary model to predict the optimal number of control points for each path (§ 3.2). Finally, each vector shape is rasterized using a differentiable rasterizer [22] and composited into a final rendering [28], which we compare to a raster ground truth for training (§ 3.3).

3.1. Vector Graphics Decoder

We choose to represent a vector graphic as a depth-ordered set of T closed Bézier paths, or equivalently, a set of T simply connected solid 2D shapes. The first operator in our decoder is a recurrent neural network (RNN) that consumes the global latent code z representing the graphic as a whole (§ 3.1.3). At each time step t, the RNN outputs a per-path latent code z_t. This mechanism lets us generate graphics with arbitrary numbers of paths, and arbitrary topology (using fill rules to combine the shapes). The path-specific codes are then individually processed by a *path decoder* module (§ 3.1.1) which outputs the parameters of a closed path of arbitrary length using cubic Bézier segments.

3.1.1 Single path decoder with circular convolutions

To ensure the individual paths are closed, we obtain them by continuous deformation of the unit circle. Specifically, for each shape, we sample $3k$ points along the circle, corresponding to the control points of k cubic Bézier segments. We compute the 2D cartesian coordinates p_i of each of these points, and annotate them with a 1-hot binary variable c_i to distinguish between the segment endpoints — every third point, which the Bézier path interpolates — and the other control points.

We replicate the path's latent code z_t and concatenate it with the sample position and point type label, so that each sample on the circle is represented as a vector $\begin{bmatrix} p_i & c_i & z_t \end{bmatrix}$, $i \in \{1, \ldots, 3k\}$, which we call a *fused latent vector*. These

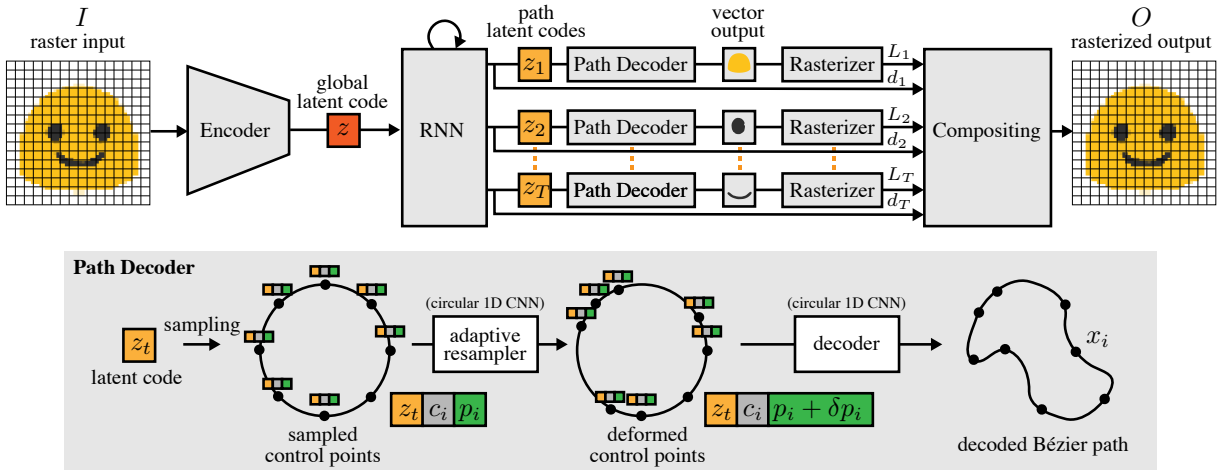


Figure 3: **Architecture overview.** We train an end-to-end variational autoencoder that encodes a raster image to a latent code z, which is then decoded to a set of ordered closed vector paths (**top**). We then rasterize the paths using DiffVG [22] and composite them together using DiffComp to obtain a rasterized output, which we compare to the ground truth raster target for supervision at training time. Our model can handle graphics with multiple component paths. It uses an RNN to produce a latent code z_t for each path, from the global latent code z representing the graphic as a whole. Our path decoder (**bottom**) decodes the path codes into closed Bézier paths. Our representation ensures the paths are closed by sampling the path control points uniformly on the unit circle. These control positions are then deformed using a 1D convolutional network with circular boundary conditions to enable adaptive control over the point density. Finally, another 1D circular CNN processes the adjusted points on the circle to output the final path control points in the absolute coordinate system of the drawing canvas. The auxiliary network that predicts the optimal number of control points per path is trained independently from our main model; it is not shown here.

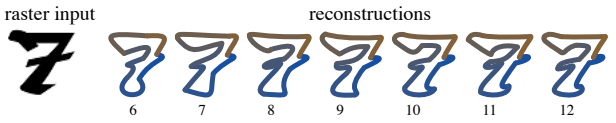
are then arranged into a cyclic buffer, which is then processed by a neural network performing 1D convolutions with cyclic boundary conditions (along the sample dimension) to obtain the final spatial locations of the path's control points: $x_1, \ldots, x_{3k}$. The cyclic convolution along the sample axis corresponds to convolution along the perimeter of the unit circle. It is a crucial component of our method because it enables information sharing between neighbouring samples, while respecting the closed topology of the shape. We use 3-tap filters for all convolutions and ReLU activations.

Sampling the unit circle rather than using a fixed-length input array allows us to adjust the complexity (i.e., the number of segments k) of the Bézier path by simply changing the sampling density. In Section 3.2, we show this sampling density can be determined automatically, based on complexity of the shape to match, using an auxiliary network. Figure 4 shows the impact of the number of segments on the reconstruction quality.

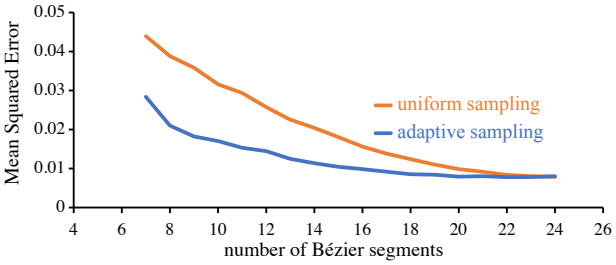
3.1.2 Adaptive control point density

The most natural choice for our control point parameterization would be to choose equally spaced sample points along the unit circle (in angle). We found this uniform control points allocation was often sub-optimal. Ideally, more control points should be allocated to sections of the path with higher complexity (e.g., sharp creases or serifs for fonts). To address this, we propose an adaptive sampling mechanism, which we call the *sample deformation* subnetwork. This module is a 1D convolutional network with cyclic boundary condition acting on the fused latent vectors $\begin{bmatrix} p_i & c_i & z_t \end{bmatrix}$, where the p_i are uniformly spaced along the circle. It outputs a displacement δp_i for each sample point. We parameterize this output in polar coordinates so that $p_i + \delta p_i$ remains on the circle.

With our adaptive sampling mechanism turned on, the path decoder now operates on the fused latent vector with sample deformation, $\begin{bmatrix} p_i + \delta p_i & c_i & z_t \end{bmatrix}$, instead of the regularly-spaced positions. In Figure 4b, we show the sample deformation module improves the reconstruction accuracy, especially when few segments are used. The benefit over the uniform sampling distribution diminishes as more curve segments are added.



(a) visual fidelity vs. number of segments



(b) error vs. number of segments

Figure 4: **Uniform vs. adaptive sampling.** Our decoder provides a natural control over the complexity of the vector graphics it produces. By adjusting the sampling density on the unit circle, we can increase the number of Bézier segments and obtain a finer or vector representation of a target raster image (a). Our adaptive sampling mechanism (§ 3.1.2) improves reconstruction accuracy, compared to a uniform distribution of the control points with the same number of segments (b). This adaptive scheme achieves good reconstructions with as few as 7–8 segments, while uniform sampling requires 12–14.

3.1.3 Decoding multi-part shapes using an RNN

So far, we have discussed a decoder architecture for a single shape, but our model can represent vector graphics made of multiple parts. This is achieved using a bidirectional LSTM [30] that acts on the graphic's latent code z. To synthesize a graphic with multiple component shapes, we run the recurrent network for T steps, in order to obtain shape latent codes for each shape: $z_1, \ldots, z_T$. We set T to a fixed value, computed before training, equal to the maximum number of components a graphic in our training dataset can have. When a graphic requires fewer than T shapes, the extra paths produced by the RNN are degenerate and collapse to a single point; we discard them before rendering.

In addition to the shape latent codes z_i, the recurrent network outputs an unbounded scalar depth value d_i for each path which is used by our differentiable compositing module when rasterizing the shapes onto the canvas.

3.2. Predicting the number of path control points

Each path (shape) in our vector output can be made of a variable number of segments. Figure 4a shows how the reconstruction loss decreases as we increase the number of curve segments from 6-25, for multiple designs. It also shows that, depending on the design's complexity, not all paths need many segments to be represented accurately. We train an auxiliary network conditioned on a path latent variable z_t to model the complexity–fidelity trade-off and automatically determine the optimal number of segments for a path. This auxiliary network has 3 fully connected layers. It outputs 3 parameters a, b, and c of a parametric curve $x \mapsto ae^{-bx} + c$ that approximates the loss graph of a given shape, with respect to the number of segments. Given this parametric approximation, we allow the user to set the quality trade-off as a threshold on the derivative of the parametric curve. Specifically, we solve for x in the derivative expression and round up to obtain the number of segments to sample. This threshold defines what improvement in the reconstruction error is worth the added complexity of an additional Bézier segment. Please refer to our supplementary for more information on the auxiliary network.

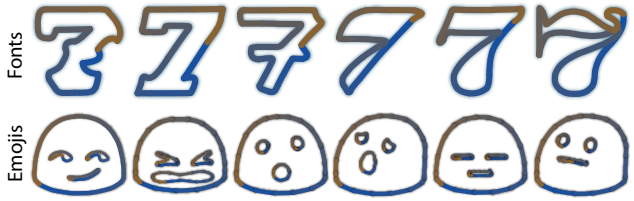


Figure 5: **Latent space correspondences.** Im2Vec encodes shapes as deformation of a topological disk. This naturally gives a point-to-point correspondence between shapes across graphics design once we encode them in our latent space. Graphics can be made of a single path (top), or multiple paths (bottom). In both cases, our model establish meaningful geometric correspondences between the designs, indicated by the blue–orange color coding.

3.3. Multi-resolution raster loss

Given a raster input image I, our model encodes the design into a global latent code z, which the RNN decomposes into path latent codes $z_1, \ldots, z_T$. Our path decoder maps each path latent code to a closed Bézier path. We rasterize each path individually, as a solid shape using the differentiable rasterizer of Li et al. [22], and composite them together into a final raster image O using the differentiable compositing algorithm of Reddy et al [28]. Since every step of the pipeline is differentiable, we can compute a loss between input image I and raseterized generated vector graphic O, and backpropagate the error to train our model using gradient descent.

When we differentiate O with respect to the Bézier parameters, the gradients have a small area of influence, corresponding to the support of the rasterization prefiltering kernel. This adversely affects convergence especially when the mismatch between I and O is high (e.g., at the early stages

of the training). We alleviate this issue by rasterizing our graphics at multiple resolutions. That is, we render an image pyramid instead of a single image, and aggregate the loss at each pyramid level. We obtain the ground truth supervision for each level by decomposing the target image into a Gaussian pyramid, where each level is downsampled by a factor 2 along each dimension from the previous level. The gradients at the coarsest level are more stable and provide a crucial signal when the images differ significantly, while the fine-scale gradients are key to obtaining high spatial accuracy. The loss we minimize is given by:

$$\mathbb{E}_{I\sim\mathcal{D}} \sum_{l=1}^{L} \|pyr_l(I) - O_l\|^2, \tag{1}$$

where L is the number of pyramid levels, $pyr_l(I)$ the l-th pyramid level, O_l our output rasterized at the corresponding spatial resolution, and $\mathcal{D}$ the training dataset.

3.4. Shape correspondences by segmentation

When specializing a generative models to a single class, e.g., the same glyph or digit across multiple fonts, it is often desirable that the model's latent space capture correspondences between parts of the instance, like the opening in the capital letter 'A', or the eyes and mouth of an emoji face. To enable this, we segment our raster training dataset using an automatic off-the-shelf tool [20]. We cluster these segments across the dataset based on spatial position, and assign to each cluster a unique RGB colour. This consistent labeling helps learn a more interpretable latent space for purposes of interpolation, but is not itself critical; we show in supplementary material that our reconstruction is robust to inconsistent labeling thanks to the differentiable compositing step.

3.5. Training details

We train our model end-to-end for 100 – 1000 epochs, using a batch size between 2 – 256 and the Ranger optimizer [38] with learning rate between 10^{-3} and 10^{-4}, depending on the dataset. To evaluate path decoder's generalization to variable number of segments, we randomly chose the number of segments $k \in \{7, \ldots, 25\}$ at every iteration.

4. Evaluation

We demonstrate Im2Vec's quantitative performance in 3 tasks: reconstruction, generation, and interpolation. We compare it with raster based ImageVAE and vector based SVG-VAE, DeepSVG on all the tasks.

Reconstruction We measure the reconstruction performance of the baselines and Im2Vec using L_2 loss in image space. This quantifies how accurately the latent space of the different methods captures the training dataset. Since SVG-VAE and DeepSVG work in vector domain, we rasterize their vector estimates using CairoSVG [4].

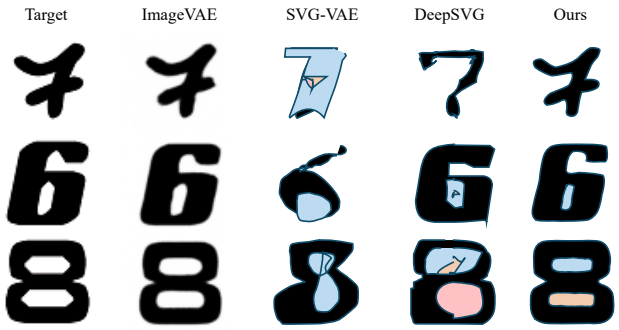


Figure 6: **Reconstructions on FONTS.** Our model, Im2Vec, captures complex topologies and produces vector outputs. ImageVAE has good fidelity but produces raster outputs with limited resolution (see Table 1). SVG-VAE and DeepSVG produce vector outputs but often fail to accurately reproduce complex fonts. All the methods were trained on the same set of fonts. Please use digital zoom to better appreciate the quality of the vector graphics.

Table 1: **Reconstruction quality.** Comparison of pixel-space reconstruction losses for various methods and datasets. Note that neither SVG-VAE nor DeepSVG operate on datasets without vector supervision.

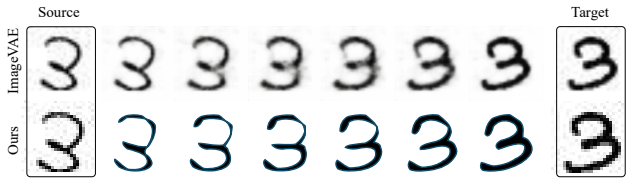
	FONTS	MNIST	EMOJIS	ICONS
ImageVAE	0.0116	0.0033	0.0016	0.0002
SVG-VAE	0.1322	×	-	-
DeepSVG	0.0938	×	-	-
Im2Vec (Ours)	0.0284	0.0036	0.0014	0.0003

Table 1 shows reconstruction quality of the Im2Vec and other baselines on FONTS [24], MNIST [21], EMOJIS [26], and ICONS [6]. While vector based methods have the advantage of being able reproduce the exact intended vector parametrization, they are adversely effected by the non-linear relationship between vector parameters and image appearance. Therefore what seems like a small error in the vector parameters estimated by SVG-VAE and DeepSVG may result in dramatic changes in appearance. Unlike vector domain methods, Im2Vec is not affected by the objective mismatch between the vector parameter and pixel spaces, thereby achieving significant improvement in the reconstruction task.

Refer to our supplementary for a chamfer distance based reconstruction comparison between SVG-VAE, DeepSVG and our method.

We show qualitative comparisons of input shape re-

(a) reconstruction



(b) interpolation

Figure 7: **MNIST results.** The MNIST dataset only provides raster data. Since no vector graphics ground truth is available, neither SVG-VAE nor DeepSVG can be trained on this dataset. We trained both ImageVAE and Im2Vec on the full dataset, with no digit class specialization or conditioning. Our model produces vector outputs, while ImageVAE is limited to low-resolution raster images (**top**). Both models produce convincing interpolation (**bottom**).

construction between methods in Figures 6 and 7a. We also show reconstruction output of Im2Vec on EMOJIS and ICONS in Fig. 8.

Generation and Interpolation We present a random sample of font glyphs generated using Im2Vec in Figure 10. A qualitative comparison of latent space interpolation between baselines and Im2Vec is presented in Figures 9b and 7b. We also present latent space interpolation between 4 input images of EMOJIS and ICONS in Fig. 9a.

Table 2: **Generation and Interpolation quality.** Results on the FONTS and the MNIST are more accurate than both previous techniques that require vector supervision, and an image-based baseline autoencoder.

	Generation		**Interpolation**	
	FONTS	MNIST	FONTS	MNIST
ImageVAE	0.171	0.058	0.184	0.072
SVG VAE	0.206	×	0.206	×
DeepSVG	0.210	×	0.202	×
Im2Vec (Ours)	0.187	0.069	0.188	0.0872

To quantitatively evaluate our generation results with others, we quantify how realistic the intermediate shapes in the latent shape as the average closest distance between the

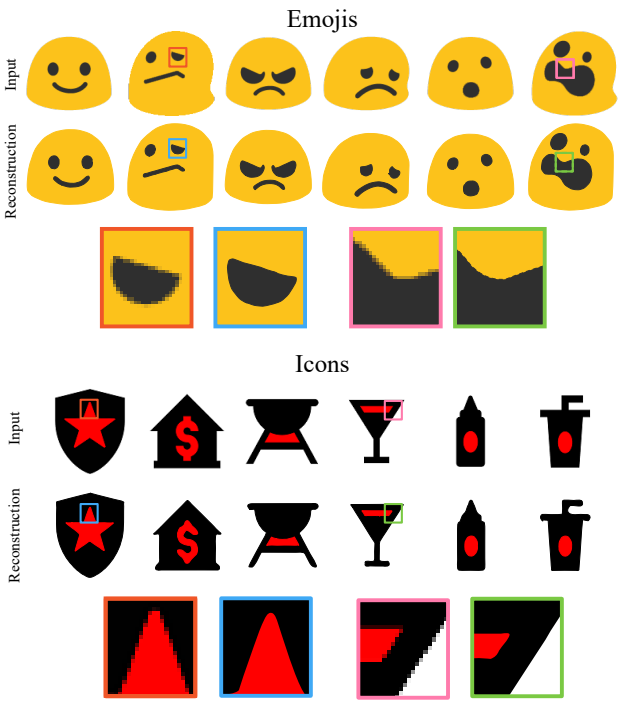


Figure 8: **Reconstructions.** Results on the EMOJIS and the ICONS datasets. In each case, we show the input image (128×128) and the corresponding vector graphics output, which can be rasterized at arbitrary resolutions.

intermediate shapes to any sample in the training dataset:

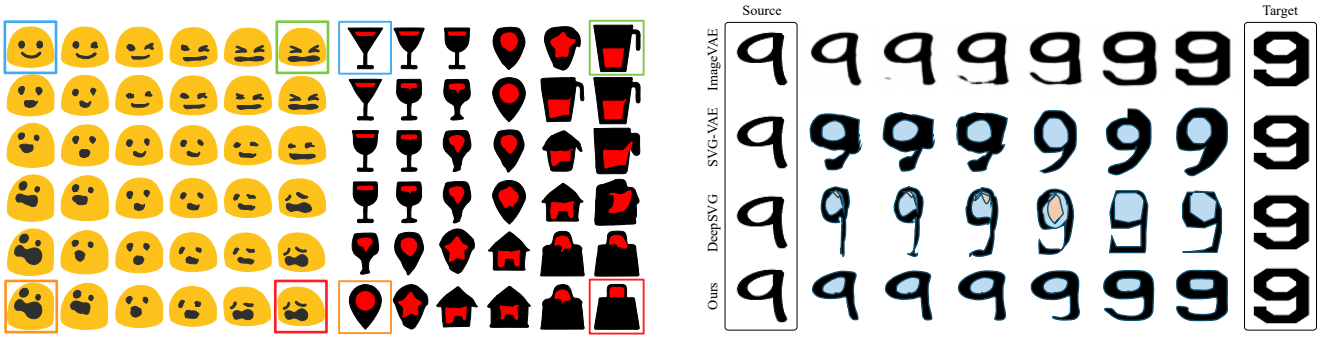
$$\sum_{O \in O_G} \min_{I \in \text{dataset}} (\|I, O\|^2), \tag{2}$$

where O_G is the set of all generated shapes. We variationally sample 1000 shapes from all the methods and present the quality of the generated shapes in Table 2.

We perform similar evaluation to quantify the quality of our interpolations. For comparison we sample 4 evenly spaced interpolations between 250 random pairs of images from the training dataset to create interpolation samples. The results of the quality of interpolation between different methods is presented in Table 2.

5. Limitations

The raster-based nature of the training imposes the principal limitations of our method (see Figure 11). It is possible for some very fine features to underflow the training resolution, in which case they may be lost. This could be addressed by increasing the resolution at the expense of computational efficiency, or perhaps by developing a more involved image-space loss. Secondly, in particularly difficult cases it is possible for the generated shape to go to a local optimum that contains degenerate features or semanti-



(a) EMOJIS and ICONS interpolations using Im2Vec

(b) Comparison to baselines on FONTS

Figure 9: **Interpolations.** Our learned latent space enables plausible interpolation between samples. In (a), we show interpolations between source–target pairs on the EMOJIS and ICONS datasets. In (b) we show interpolations on the FONTS dataset. Unlike previous work, Im2Vec enables plausible interpolation even across significant changes in shape. For instance, the stem of the digit '9' naturally curls along the interpolation path.

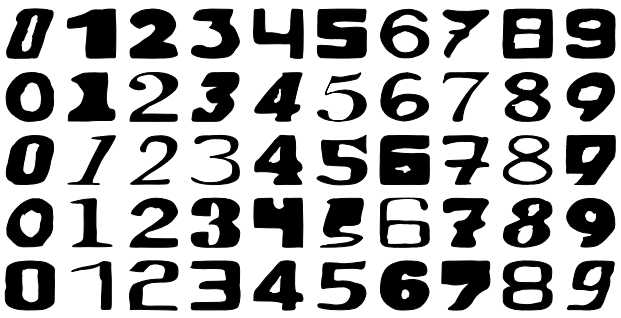


Figure 10: **Random samples.** We show a random selection of digits generated by Im2Vec. The latent space was trained on the full Fonts dataset. Our model is capable of generating samples with significant topological variations across the different font types. In the supplemental material, we include 1000 random samples from the latent space. Please use digital zoom to better evaluate the quality.

Figure 11: **Limitations.** Im2Vec is only supervised by an image-space loss, so it can sometimes miss small topological features (**Left**), or produce semantically meaningless or degenerate geometries (**Right**). While the former can be resolved by providing higher resolution supervision, the later could be mitigated by using local geometric priors.

cally non-meaningful parts which nonetheless still result in a plausible rasterised image. This is a consequence of lack of vector supervision, but could possibly be addressed by imposing geometric constraints on the generated paths.

6. Conclusion

We presented Im2Vec as a generative network that can be trained to produce vector graphics output of varying complexity and topology using only image supervision, without requiring vector sequence guidance. Our generative setup supports projection (i.e., converting images to vector sequences), sampling (i.e., generating new shape variations directly in vector form), as well as interpolation (i.e., morphing from one vector sequence to another, even with topological variations). Our evaluations show that Im2Vec achieves better reconstruction fidelity compared to methods requiring vector supervision.

We hope that this method can become the fundamental building block for neural processing of vector graphics and similar parametric shapes.

S.1. Network Architecture

Our Encoder network contains 5 2D convolution residual blocks, with [32, 64, 128, 256, 512] filters respectively. All the convolution layers have kernel size 3, stride 2 and zero pad the input by 1 pixel in both spatial dimensions. The convolutional layers are followed by two parallel fully-connecter layers that each output a vector of size 128; they represent the mean and variance for the latent embedding. Our Path decoder has 6 1D convolution layers with [170, 340, 340, 340, 340, 340, 2] channels. All the 1D convolutions have kernel size 3, stride 1 and circular padding of the input by 1 tap. Our auxiliary network contains 4 fully-connected layers with [256, 256, 256, 3] channels, respectively. The sample deformation network contains 3 1D convolution layers with [340, 340, 1] channels, all the convo-

Table S1: **Reconstruction quality.** Comparison of Bidirectional Chamfer Distance reconstruction losses for various methods and datasets.

	FONTS
ImageVAE	×
SVG-VAE	0.168
DeepSVG	0.136
Im2Vec (Ours)	0.279

lution layers have the same kernel size, stride and padding as the 1D convolution layers in path decoder. All layers are followed by ReLU activations, except the last layer of the path decoder which is followed by a sigmoid activation.

S.2. Chamfer Distance

An alternative, commonly used metric to measure the reconstruction accuracy for geometric objects is the Chamfer distance. For two point sets X, Y, the Chamfer distance is defined as:

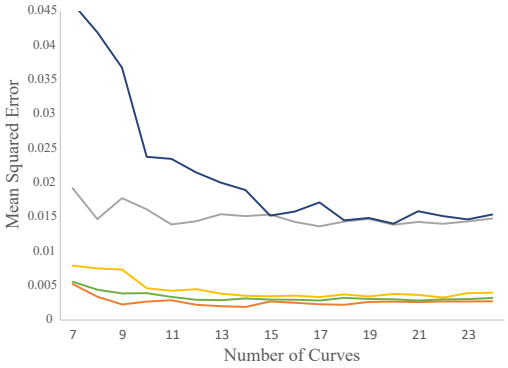
$$\sum_{x \in X} \min_{y \in Y} \|x - y\|^2 + \sum_{y \in Y} \min_{x \in X} \|x - y\|^2. \quad (3)$$

In Table S1, we show the bidirectional Chamfer distance computed between the synthesized and ground truth geometries, with points sampled uniformly along the shape boundaries (according to the path parameterization). Unlike the pixel-space metrics we report in the main paper, this evaluation suggests our model underperforms the baselines. This is misleading. As shown by Smirnov *et al*. [36], the Chamfer distance varies wildly, depending on the sampling pattern (and the parameterization, by extension). This adversely impacts our method. The baselines (DeepSVG and SVG-VAE) are optimized to regress the ground truth vector parameterization, which leads to a lower Chamfer loss, despite worse perceptual fidelity. Conversely, our method is trained on raster data only. It does not seek to retrieve the ground truth parameterization of the curve, but rather to faithfully capture the (rasterized) appearance. Therefore, our model achieves higher fidelity, despite a higher Chamfer distance.

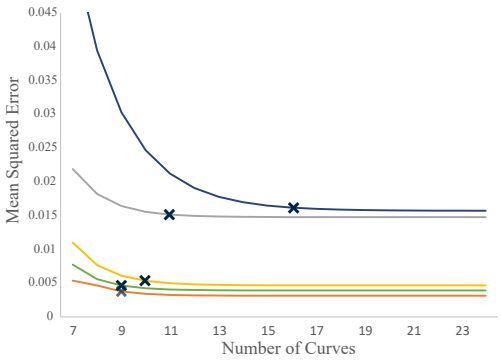
S.3. Auxiliary Network

In Figure S1b, we plot the reconstruction error vs. number of path segments for 5 randomly sampled instances from the FONTS dataset, as estimated by our trained Im2Vec model. In Figure S1c, we show a plot of the reconstruction error vs. the number of segments modelled by our auxiliary network. We use Equation (4) to select the appropriate sampling rate for new shapes generated using our method. This

(a) visual fidelity vs. number of segments



(b) error vs. number of segments; ground truth



(c) error vs. number of segments; modelled

Figure S1: **Auxillary network output.** Our auxiliary network helps us choose the best sampling density on the unit circle, such that we express the generated shape with the fewest number of Bézier curves based on the user defined reconstruction error threshold.

enables us represent each of the generated shape in the most compact way. We mark 'x' in the Figure S1c for k = 0.005.

N = number of curves and **k = Rate of change of loss.**

$$\text{loss} = c + \exp(b - a * N) \quad (4)$$

$$d\text{loss}/dN = -a \exp(b - a * N) \quad (5)$$

$$\implies k > -a \exp(b - a * N) \quad (6)$$

$$\implies k/a < \exp(b - a * N) \quad (7)$$

$$\implies \log(k/a) < b - a * N \quad (8)$$

$$\implies -b + \log(k/a) < -a * N \quad (9)$$

$$\implies -\log(k/a) + b > a * N \quad (10)$$

$$\implies \frac{\log(a/k) + b}{a} > N \quad (11)$$

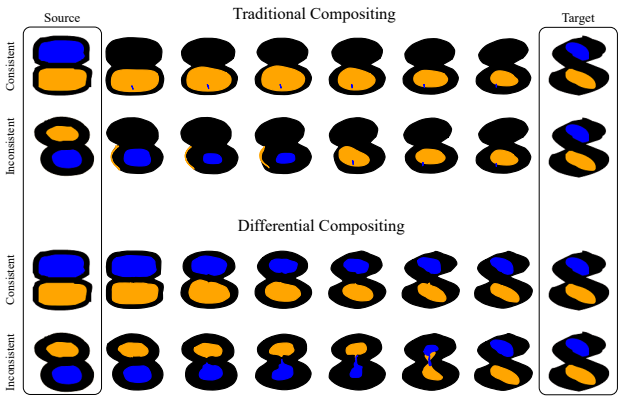


Figure S2: **Training on inconsistent dataset.** We show latent space interpolations using models trained with traditional compositing versus differential compositing. In both the scenarios, we show examples of interpolations between consistently labelled instances and inconsistently labelled instances. When trained with differential compositing, in both the scenarios our model is robust to pre-processing inconsistencies.

S.4. Robustness to Inconsistent Correspondence

In Section 3.4 of the main paper, we describe a pipeline that segments the sub-components of a design or font, using off-the-shelf tools, which improves the interpretability and consistency of latent space interpolations. In Figure S2, we show that differential compositing makes our method robust to potential inconsistencies in this automatic labelling step. For this experiment, we specifically created training and test datasets of font character '8' where the openings are colored inconsistently. Note that our model still manages to capture meaningful interpolations for instances that have consistent labels.

References

[1] E. Alberto Dominici, N. Schertler, J. Griffin, S. Hoshyari, L. Sigal, and A. Sheffer. Polyfit: Perception-aligned vectorization of raster clip-art via intermediate polygonal fitting. *ACM Transaction on Graphics*, 39(4), 2020. 2

[2] S. Azadi, M. Fisher, V. G. Kim, Z. Wang, E. Shechtman, and T. Darrell. Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7564–7573, 2018. 3

[3] M. Bessmeltsev and J. Solomon. Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics (TOG)*, 38(1):1–12, 2019. 2

[4] CairoSVG. https://github.com/kozea/cairosvg. 6

[5] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte. Deepsvg: A hierarchical generative network for vector graphics animation, 2020. 1, 3

[6] creativeStall. https://thenounproject.com/creativestall/. 6

[7] V. Egiazarian, O. Voynov, A. Artemov, D. Volkhonskiy, A. Safin, M. Taktasheva, D. Zorin, and E. Burnaev. Deep vectorization of technical drawings. *arXiv preprint arXiv:2003.05471*, 2020. 2

[8] K. Ellis, D. Ritchie, A. Solar-Lezama, and J. B. Tenenbaum. Learning to infer graphics programs from hand-drawn images. corr abs/1707.09627 (2017). *arXiv preprint arXiv:1707.09627*, 2017. 3

[9] J.-D. Favreau, F. Lafarge, and A. Bousseau. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)*, 35(4):1–10, 2016. 2

[10] Y. Ganin, T. Kulkarni, I. Babuschkin, S. Eslami, and O. Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018. 3

[11] J. Gao, C. Tang, V. Ganapathi-Subramanian, J. Huang, H. Su, and L. J. Guibas. Deepspline: Data-driven reconstruction of parametric curves and surfaces. *arXiv preprint arXiv:1901.03781*, 2019. 2

[12] Y. Gao, Y. Guo, Z. Lian, Y. Tang, and J. Xiao. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019. 3

[13] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018. 2

[14] Y. Guo, Z. Zhang, C. Han, W. Hu, C. Li, and T.-T. Wong. Deep line drawing vectorization via line subdivision and topology reconstruction. In *Computer Graphics Forum*, volume 38, pages 81–90. Wiley Online Library, 2019. 2

[15] D. Ha and D. Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017. 3

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015. 3

[17] S. Hoshyari, E. Alberto Dominici, A. Sheffer, N. Carr, D. Ceylan, Z. Wang, and I.-C. Shen. Perception-driven semi-structured boundary vectorization. *ACM Transaction on Graphics*, 37(4), 2018. 2

[18] Z. Huang, W. Heng, and S. Zhou. Learning to paint with model-based deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8709–8718, 2019. 3

[19] P. K. Jayaraman, A. Sanghi, J. Lambourne, T. Davies, H. Shayani, and N. Morris. Uv-net: Learning from curve-networks and solids. *arXiv preprint arXiv:2006.10211*, 2020. 2

[20] J. Kopf and D. Lischinski. Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4):99:1 – 99:8, 2011. 2, 6

[21] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. 6

[22] T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):193:1–193:15, 2020. 2, 3, 4, 5

[23] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2195–2203, 2017. 2

[24] R. G. Lopes, D. Ha, D. Eck, and J. Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7930–7939, 2019. 1, 3, 6

[25] R. Nakano. Neural painters: A learned differentiable constraint for generating brushstroke paintings. *arXiv preprint arXiv:1904.08410*, 2019. 3

[26] notoEmoji. https://github.com/googlefonts/noto-emoji. 6

[27] D. Paschalidou, A. O. Ulusoy, and A. Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10344–10353, 2019. 2

[28] P. Reddy, P. Guerrero, M. Fisher, W. Li, and N. J. Mitra. Discovering pattern structure using differentiable compositing. *ACM Trans. Graph. (Proceedings of SIGGRAPH Asia 2020)*, 39(6):262:1–262:15, 2020. 2, 3, 5

[29] L. S. F. Ribeiro, T. Bui, J. Collomosse, and M. Ponti. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14162, 2020. 3

[30] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997. 5

[31] P. Selinger. Potrace: a polygon-based tracing algorithm. *Potrace (online), http://potrace. sourceforge. net/potrace. pdf (2009-07-01)*, 2003. 2

[32] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji. Csgnet: Neural shape parser for constructive solid geometry. corr abs/1712.08290 (2017). *arXiv preprint arXiv:1712.08290*, 2017. 3

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 2

[34] A. Sinha, A. Unmesh, Q. Huang, and K. Ramani. Surfnet: Generating 3d shape surfaces using deep residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6040–6049, 2017. 2

[35] D. Smirnov, M. Bessmeltsev, and J. Solomon. Deep sketch-based modeling of man-made shapes. *arXiv preprint arXiv:1906.12337*, 2019. 2

[36] D. Smirnov, M. Fisher, V. G. Kim, R. Zhang, and J. Solomon. Deep parametric shape predictions using distance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 561–570, 2020. 2, 9

[37] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 3

[38] Q. Tong, G. Liang, and J. Bi. Calibrating the adaptive learning rate to improve convergence of adam. *arXiv*, pages arXiv–1908, 2019. 6

[39] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2635–2643, 2017. 2

[40] N. Zheng, Y. Jiang, and D. Huang. Strokenet: A neural painting environment. In *International Conference on Learning Representations*, 2018. 2, 3

[41] C. Zou, E. Yumer, J. Yang, D. Ceylan, and D. Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 900–909, 2017. 2