

DeepSpline: Data-Driven Reconstruction of Parametric Curves and Surfaces

Jun Gao^{1,2†}, Chengcheng Tang⁴, Vignesh Ganapathi-Subramanian⁴, Jiahui Huang³, Hao Su⁵, Leonidas J. Guibas⁴

¹University of Toronto; ²Vector Institute;
³Tsinghua University; ⁴Stanford University; ⁵UC San Diego

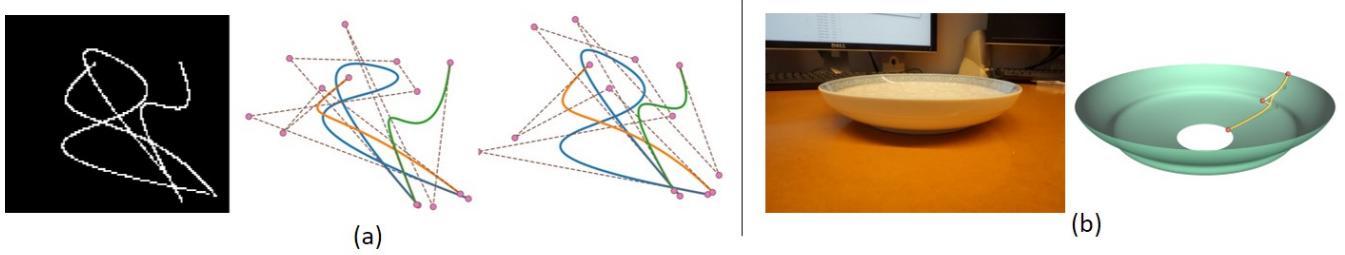


Figure 1: 2D and 3D Parametric Reconstruction. (a) Given a binary image as input on the left, we use a Hierarchical RNN to generate its parametric reconstruction as shown in the middle image. This could be used as initialization to obtain a better reconstruction using traditional optimization methods in the right image. (b) A real image is provided as input on the left, and the rotational symmetry of the object in the image is leveraged to reconstruct the surface on the right, after learning the spline curve that models the profile of the object in the image. Please note that even the subtle kink on the dish is captured in our reconstruction.

Abstract

Reconstruction of geometry based on different input modes, such as images or point clouds, has been instrumental in the development of computer aided design and computer graphics. Optimal implementations of these applications have traditionally involved the use of spline-based representations at their core. Most such methods attempt to solve optimization problems that minimize an output-target mismatch. However, these optimization techniques require an initialization that is close enough, as they are local methods by nature. We propose a deep learning architecture that adapts to perform spline fitting tasks accordingly, providing complementary results to the aforementioned traditional methods. We showcase the performance of our approach, by reconstructing spline curves and surfaces based on input images or point clouds.

CCS Concepts

- Computing methodologies → Parametric curve and surface models;

1. Introduction

Three-dimensional data, that is useful for everyday geometric representation and design, is essential in modern industry and research. Generating, approximating, processing and storing such data succinctly, and yet faithfully, are vital research problems with a long history of study. While it would be ideal to use three-dimensional data in its most lossless form, the theoretical limit on representability as well as the source of data collection make the mode of representation crucial. For different applications, varied modes of 3D data representations are used, such as grids, points clouds, meshes and splines.

In the broad spectrum of 3D data representations, the most abstract forms are those based on splines and parametric primitives. Their natural representability for arbitrarily smooth surfaces makes them the industry standard for computer-aided design. Therefore, while representations such as point clouds are more accessible, conversion to parametric representations is often necessary. Generating clean geometric data with parametric surfaces based on observations often requires approximation and fitting based on a distance measure. However, the requirement for a measurable distance between target and output and an initialization are common limitations for such a family of fitting algorithms.

On another front, creating geometry that could be induced and inferred from single images has historically been a very attractive

† jungao@cs.toronto.edu

research problem. A strikingly exciting direction is geometric inference from indirect inputs, especially single images. While such tasks, commonly known as Shape from X, have been studied for over four decades, the recent advances in deep learning have shed a new light on recovering these geometries. By synthetically generating observations based on ground-truth geometry, recovering geometry based on single images has been illustrated with the representation of voxel grids or point clouds over a wide variety of different shapes.

A natural question that then arises is if there is a way to infer the parametrizable surfaces, directly based on the input information. A deep learning framework is best suited to perform such inference for multiple reasons. Fundamentally, deep learning-based frameworks are adaptable to varied input formats, especially when the relationship between the data and geometry is highly nonlinear and non-convex, e.g., images. Besides, by learning over a myriad of examples, deep networks also do away with the need for manual initialization of the control points, thereby making the inference process less laborious. Finally, the deep learning frameworks also aid in performing inference when the number of control points and curves are variable in number, which often needs to be determined heuristically in the traditional setting.

In this paper, we attempt to reconstruct spline curves and surfaces using data-driven approaches. To tackle challenges with the 2D cases such as multiple splines with intersections, we use a hierarchical Recurrent Neural Network (RNN) trained with ground truth labels, to predict a variable number of spline curves, each with an undetermined number of control points. In the 3D case, we reconstruct surfaces of revolution and extrusion without self-intersection through an unsupervised learning approach, that circumvents the requirement for ground truth labels.

To summarize, our contributions are as follows:

- We define two single-layer RNNs, Curve RNN and Point RNN that can be used to perform curve predictions and control point predictions respectively.
- We provide a Hierarchical RNN architecture model that performs reconstruction of 2D spline curves of variable number, each of which contains a variable number of control points using nested Curve RNN and Point RNN units, and an algorithm to train it effectively.
- We provide an unsupervised parametric reconstruction model that performs reconstruction of 3D surfaces of extrusion or revolution.

2. Related Work

The problem of recovering faithful yet succinct representations of geometry, has been studied extensively over the past decades with varied forms of outputs and inputs. There are three categories of works that have been relevant to this study and fundamental to multiple applications. We first discuss previous work on spline fitting, in which a direct minimization of a distance between the target and the result is performed. Next, we discuss the creation of shapes from indirect information, i.e., Shape from X, with current advances in deep learning. Finally, we also review the work done in vectorization of rasterized images and discuss the main differences.

Spline Fitting In computer-aided design and computer graphics, registering or fitting curves or surfaces to targets, e.g., point clouds, are essential for a wide range of tasks such as industrial design following a physical sculpture. One of the most widely used method is the Iterative Closest Point (ICP) [BM^{*}92, CM91], which minimizes the distances between two clouds of points based on an initial configuration, iteratively evaluated. Following a similar goal of minimizing a directly measurable and differentiable distance, multiple variants of registration and fitting algorithms are proposed. By devising a metric that is adaptive to curvature, Wang et al proposed a faster and more robust algorithm [WPL06], which was further accelerated with quasi-Newton methods [ZBLW12]. This has further been extended to spline surfaces with constraints such as developability [TBWP16]. Despite being able to successfully and efficiently minimize the energies encoding distances, a proper initialization – a given number of points at selected positions – is always necessary for such approaches. In contrast, our method does not require such an initialization and could even be used to complement those previous approaches to improve their fits.

Shape from X There has been a long history of interest in discovering shapes from indirect inputs on images, e.g., from shading [Hor70, HB89] or texture [Ken79]. Most traditional methods follow a sequential procedure including, e.g., light source estimation, normal estimation, and depth estimation. More recently, multiple works generate shapes based on images in an end-to-end manner with the help of deep neural networks and synthetically generated data, for shapes represented as volumetric grid [CXG^{*}16] or point clouds [FSG16]. Despite the variety and complexity of recoverable shapes from single images, a large gap remains between the reconstruction and a clean geometry. While multi-scale approaches, e.g., based on octree [TDB17], attempt to enhance geometric details with higher resolution, we attempt to directly recover the parameters of geometric primitives, especially spline curves or surfaces directly, which provide a higher level geometric abstraction with an arbitrarily high resolution.

Image Tracing Another field of related work is image tracing or vectorization, in which a rasterized image is converted to a vectorized one. Commercially available tools, e.g., Illustrator, often provide a fine tessellation with an excessive amount of control points to ensure fidelity. While most vectorization techniques work on the boundaries of the shapes, recent works such as [FLB16] and [SSISI16] strive for a simplification of the output on curves in a globally consistent manner. Besides tracing based on direct differences of colors, Polygon-RNN [CKUF17] used an RNN to predict the polygon contour of an object on a semantic level, which could be used for instance segmentation or reducing labelling labor. In contrast to these methods that perform vectorization, we aim to abstract the simplest types of representation, based on general splines instead of polylines or interpolating cubic Bezier curves, and to create 3D surfaces based on images.

3. Overview

3.1. Motivation

Traditional methods attempt to extract curved lines or surfaces from images leveraging low-level local image features such as gradients.



Figure 2: Canny Edge Detector on real image. The image on the left shows a real image with a dish, and the image on the right is the output of the Canny Edge Detector of the image on the former.

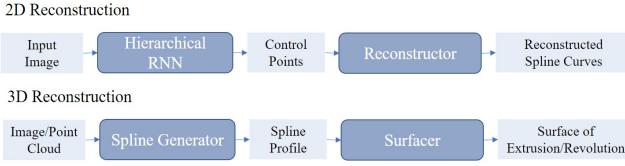


Figure 3: Overview. The image and point cloud representations of curves or 3D shapes are used to generate control points and splines, which can be used for 2D and 3D reconstruction.

For example, Canny Edge Detector [Can86] has been a popular edge detector in computer vision research for decades. Although low-level image features can work well for simple cases when the background is simple and clean, like handwriting on white clean papers, in images with clutter as in Figure 2, it is much more challenging to extract the profile of objects. It becomes vital then to develop a more robust method that can better exploit image content. As shown in recent computer vision papers for object recognition [KSH12], deep learning methods seem to be able to learn object categories agnostic to the nuisance factors such as lighting, background clutter, pose variation etc. We, therefore, resort to deep learning methods to detect and generate parametric curves for 2D/3D reconstruction. This serves as crucial motivation to solve this problem using learning based techniques.

3.2. Method

The spline-based reconstruction techniques presented in this paper are performed on multiple input modalities, specifically images and point clouds, and both 2D and 3D reconstruction are discussed in this paper (Figure 3). For 2D reconstruction from images, there are two stages, prediction of spline curves in the image to reconstruct, and prediction of the actual control points that reconstruct the afore-mentioned spline curve. In the paper, we address both these reconstructions as individual problems. We solve the prediction of identifying the spline curves, knowing each of the curves have a fixed number of control points, and the prediction of identifying the control points, given a single spline curve, but not knowing the number of control points that are used to generate this curve. The reconstruction techniques used for these individual problems are then utilized along with a hierarchical deep learning module

called the Hierarchical RNN, to solve the more general problem of predicting both the unknown quantities.

For 3D reconstruction, we perform surface reconstruction in the case of an extruded cross-section or rotational symmetry with two input modes, images or point clouds. These image or point cloud data are processed to learn features, from which spline curves that generate the shape are learned. From the spline curve that is learned, a surface of extrusion or revolution is generated by extruding the curve along the path of extrusion or revolving the curve about the axis of symmetry.

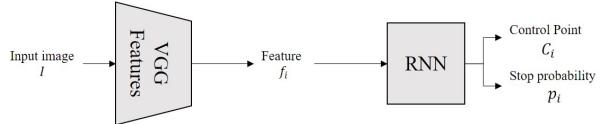


Figure 4: Network architecture for single spline curve fitting with a variable number of control points. VGGNet is used for feature extraction. Image features will be used as input to the RNN. At each iteration i , the RNN predicts the position of control point C_i and the stop probability p_i to decide if to continue the iterative process.

4. Supervised 2D Reconstruction

Reconstructing spline curves in images consists of two modes of variability: the number of spline curves and the number of control points. While solving for both these factors of variability can prove challenging, solving the sub-problems where one of these modes is fixed can provide ample intuition towards solving the harder problems with more variables. In this section, we provide basic models of spline curve fitting solving both these variability issues. First, we propose a model that infers a variable number of control points to fit a single spline curve. Then, we extend this to fitting multiple spline curves with a fixed number of control points. Finally, we propose the Hierarchical RNN to solve for multiple spline curves with a variable number of control points.

4.1. Single Spline Curve, Variable Number of Control Points

A vanilla model is used to tackle the problem of fitting a variable number of control points to a single spline curve. For the spline curve in consideration, the corresponding control points form an ordered sequence, with each element of the sequence being the position of one control point. The prediction of these variable number of control points could, therefore, be viewed as inferring a variable length sequence. A very similar learning technique has previously been used in Machine Translation and Image Captioning. [SVL14, BCB14, CVMG*14, VTBE15]. The use of RNNs for this generative process is a natural choice.

The input to the pipeline is an image that contains the spline curve. A deep convolutional network is used to extract a feature vector from this image. This feature vector is then forwarded to an RNN module, which predicts the control point sequence. The RNN module performs a dual prediction task. At each iteration, the RNN predicts the position of a new control point and the probability with which this control point is the endpoint. The probability is predicted

as a distribution over two states {CONTINUE = 0, STOP = 1}. Specifically, at time step t , the model predicts the t^{th} control point C_t and the probability p_t for the prediction to stop at this time step. In the ideal scenario, the value of p_t would be binary, with value 0 for all time steps, and 1 for the final time step, forcing the prediction to end.

The network architecture is shown in Figure 4. Here, we use the VGGNet, described in [SZ14], to extract image features, and then perform mean pooling to obtain a vector representation of the whole image. This feature vector is fed into a linear layer to get a more abstract feature vector with 512 dimensions and then supplied into the RNN. We use a one-layer Gated Recurrent Unit (GRU) [CVMBB14] as the basic block of the RNN, with the dimension of input and hidden layers set to 512. At each time step, the hidden vector h_t of RNN is fed forward into a two-layer fully-connected network to produce the control point's position C_t and the stop probability p_t . Specifically, the output of this two-layer fully-connected network has four units, the first two represent C_t and we append a softmax layer into the last two units, which provide a probability distribution over {CONTINUE = 0, STOP = 1}.

We use a Mean-Squared Error (MSE) term to optimize on the position of the control points, and a Cross-Entropy term to optimize for the predicted stop probability. The loss function used to train the RNN is

$$L_1 = \sum_{i=1}^N \left(\sum_{k=1}^{n_i} (C_k^i - \hat{C}_k^i)^2 \right) \quad (1)$$

$$+ \lambda (p_k^i \log(\hat{p}_k^i) + (1 - p_k^i) \log(1 - \hat{p}_k^i)) \quad (2)$$

where N is the size of the training dataset, n_i is the number of control points of i^{th} spline curve in the training dataset, \hat{C} is the predicted position of a control point, while C is the corresponding ground truth, \hat{p} is the predicted stop probability, p is the ground truth stop probability (1 when $k = n_i$, 0 otherwise) and λ is the optimization hyperparameter.

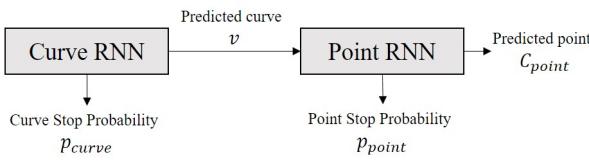


Figure 5: Hierarchical RNN architecture. The Curve RNN acts as an outer loop to determine when all curves in the image have been generated. For each iteration of the Curve RNN, the Point RNN runs the prediction based on the Single Curve, Variable Control Points process as described in Section 4.1 with input obtained from the Curve RNN to perform reconstruction of this curve. The Curve RNN predicts the stop probability to signal end of all curves, p_{curve} and the predicted curve v , while the Point RNN predicts the stop probability to signal end of predicted points, p_{point} and the predicted point at that particular iteration C_{point} .

4.2. Multiple Spline Curves, Fixed Number of Control Points

Fitting multiple spline curves with a fixed number of control points is solved with a minor modification to the model in Sec.4.1. Here, instead of predicting if a certain point is the final control point, the entire spline curve along with all control points are computed at each iteration of the RNN. The RNN also predicts the probability to determine if the most recent curve is the final curve.

However, there are two new challenges that arise in this solution. The RNN predicts an ordered sequence of curves, while the target of multiple spline curves is an unordered bag of spline curves. Therefore, a correspondence needs to be established between the target curves and the curves that are predicted by the network. This is achieved by modeling this problem as a matching problem in a bipartite graph, with the two sets being the set of target curves and the set of predicted curves. The weight of each edge between the curves in the two sets would be the distance between the two curves. The Hungarian Algorithm is implemented to obtain a matching of minimal cost. This is similar in spirit to the matching problem solved in [RT15].

The second challenge is to ensure that when a certain spline curve is being processed, influences from regions of other spline curves are minimized. Since there are multiple curves, occupying different regions of the image, it becomes necessary to nullify these influences. This is handled by adapting the attention mechanism [SSWF15, KIO*16] of the network. At each time step t , the image features are scaled by an attention map showing weights of different regions in the image before passing into the RNN. This ensures that the attention of the network is localized to the region in which the current spline dominates. The idea of using a localization network to perform this task of drawing attention to certain regions over others is used with considerable success in [XBK*15]. Our methods are demonstrated in more detail in Sections 4.2.1 and 4.2.2.

4.2.1. Loss Function

The training dataset is composed of labeled images. Image i as $I^i \in \mathcal{R}^{128 \times 128 \times 3}$, its annotation $S^i = \{S_1^i, S_2^i, \dots, S_{n_i}^i\}$ is a set of spline curves, n_i is the number of spline curves in this image, S_j^i is a sequence of control point positions that construct spline curve j inside the image I^i . In all our experiments, we use $n_i \in \{1, 2, 3\}$.

The network predicts both a sequence of spline curves $\hat{S}^i = \{\hat{S}_1^i, \hat{S}_2^i, \dots, \hat{S}_{n_i}^i\}$, and stop probabilities $\hat{P}^i = \{\hat{p}_1^i, \hat{p}_2^i, \dots, \hat{p}_{n_i}^i\}$. We train the RNN by running n_i iterations for each training instance S^i . On inference, the number of iterations is determined by the predicted stop probability \hat{p}_j^i . The recurrence stops when $\hat{p}_j^i > 0.5$.

The loss on the probability sequence is defined as earlier.

$$L_p^i = \sum_{j=1}^{n_i} p_j^i \log \hat{p}_j^i + (1 - p_j^i) \log(1 - \hat{p}_j^i) \quad (3)$$

$$p_j^i = \begin{cases} 0 & j < n_i \\ 1 & j = n_i \end{cases} \quad (4)$$

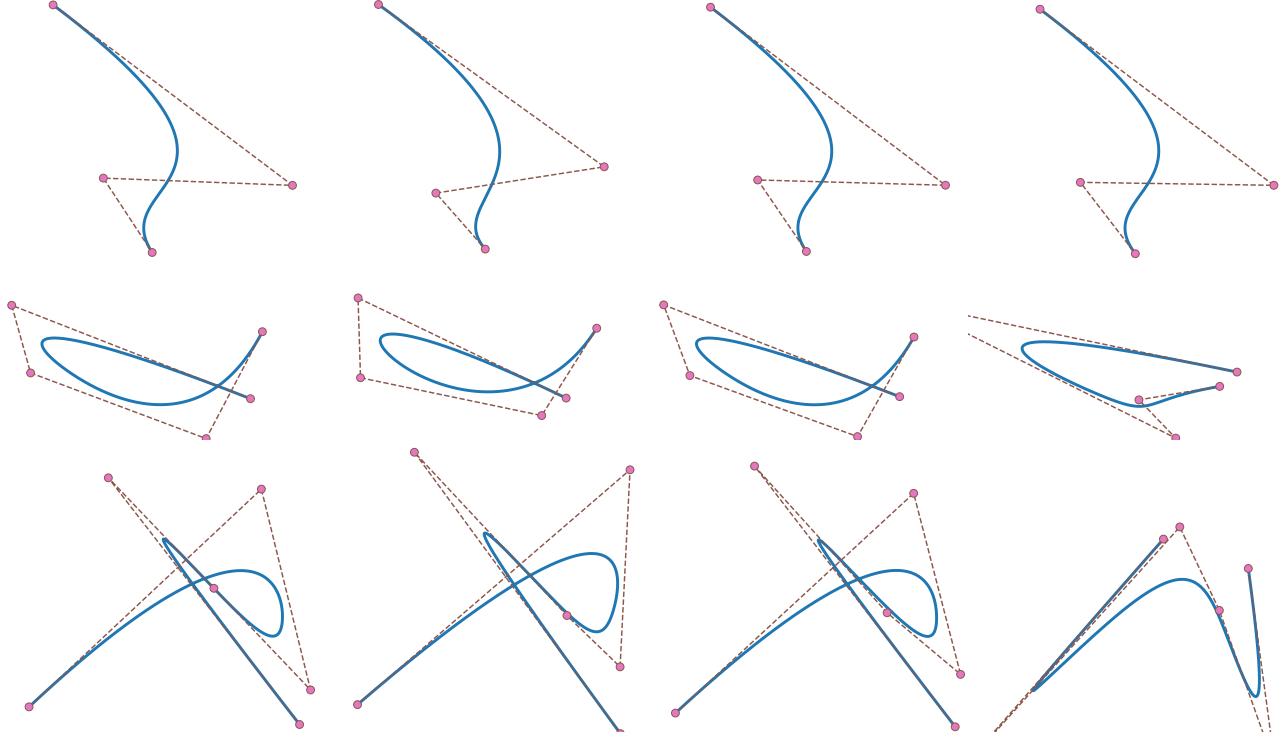


Figure 6: Randomly picked results of variable number of control points. The first column shows the target curve, the second column shows the prediction from the RNN, the third column shows the reconstruction obtained by using the RNN prediction as initialization for traditional methods, and the last column shows the reconstruction obtained by using a random initialization.

As described earlier, the loss term that measures the distance between the target and predicted spline curves also needs alignment. Since \hat{S}^i is an ordered sequence and the ground truth S^i is an unordered set, the order of processing the spline curves in an image is not easily determinable. Random allocation of processing rank to S^i will result in a problem of ambiguity and the network convergence is not guaranteed. The bipartite graph model is used here, as described earlier, to model the correspondence, while computing the reconstruction loss. This reconstruction loss is computed as follows:

$$L_c^i = \min_{\delta \in \mathbf{K}^i} f_{Match}(S^i, \hat{S}^i, \delta) \quad (5)$$

where

$$f_{Match}(S^i, \hat{S}^i, \delta) = \sum_{j=1}^{n_i} \left(\sum_{\hat{j}=1}^{n_i} f_{ED}(S_j^i, \hat{S}_{\hat{j}}^i) \delta_{j,\hat{j}} \right), \quad (6)$$

$$\mathbf{K}^i = \left\{ \delta \in \{0,1\}^{n_i \times n_i} : \sum_{j=1}^{n_i} \delta_{j,\hat{j}} = 1, \forall \hat{j} \in \{1, \dots, n_i\}, \right. \quad (7)$$

$$\left. \sum_{\hat{j}=1}^{n_i} \delta_{j,\hat{j}} = 1, \forall j \in \{1, \dots, n_i\} \right\}. \quad (8)$$

$f_{ED}(S_j^i, \hat{S}_{\hat{j}}^i) = \sum_{k=1}^m (C_{S_j^i}^k - C_{\hat{S}_{\hat{j}}^i}^k)^2$ is the Euclidean distance between the position of control points at S_j^i and $\hat{S}_{\hat{j}}^i$, and m is the number of control points. we use $m = 5$ in our experiments.

δ defines a matching between ground truth S^i and prediction result \hat{S}^i . Equation 7 guarantees a bijection between the ground truth and prediction results. The optimal matching δ in Equation 5 is efficiently computed by the Hungarian algorithm. Therefore, the overall loss function is defined as:

$$L_2 = \sum_{i=1}^N \left(\min_{\delta \in \mathbf{K}^i} f_{Match}(S^i, \hat{S}^i, \delta) \right) \quad (9)$$

$$+ \lambda \sum_{j=1}^{n_i} (p_j^i \log \hat{p}_j^i + (1 - p_j^i) \log (1 - \hat{p}_j^i)) \quad (10)$$

4.2.2. Attention Network

At each iteration of the RNN, a smaller network f_{att} is used to predict a soft attention mask. This mask provides localized weights to different regions of the image features.

The output of the feature extractor is a 3D tensor \mathcal{M} , of size (l, x, y) , where l is the number of channels in the last layer of the network and (x, y) is the downsampled size of input image. Each column, a vector that has l elements, relates to a certain region in

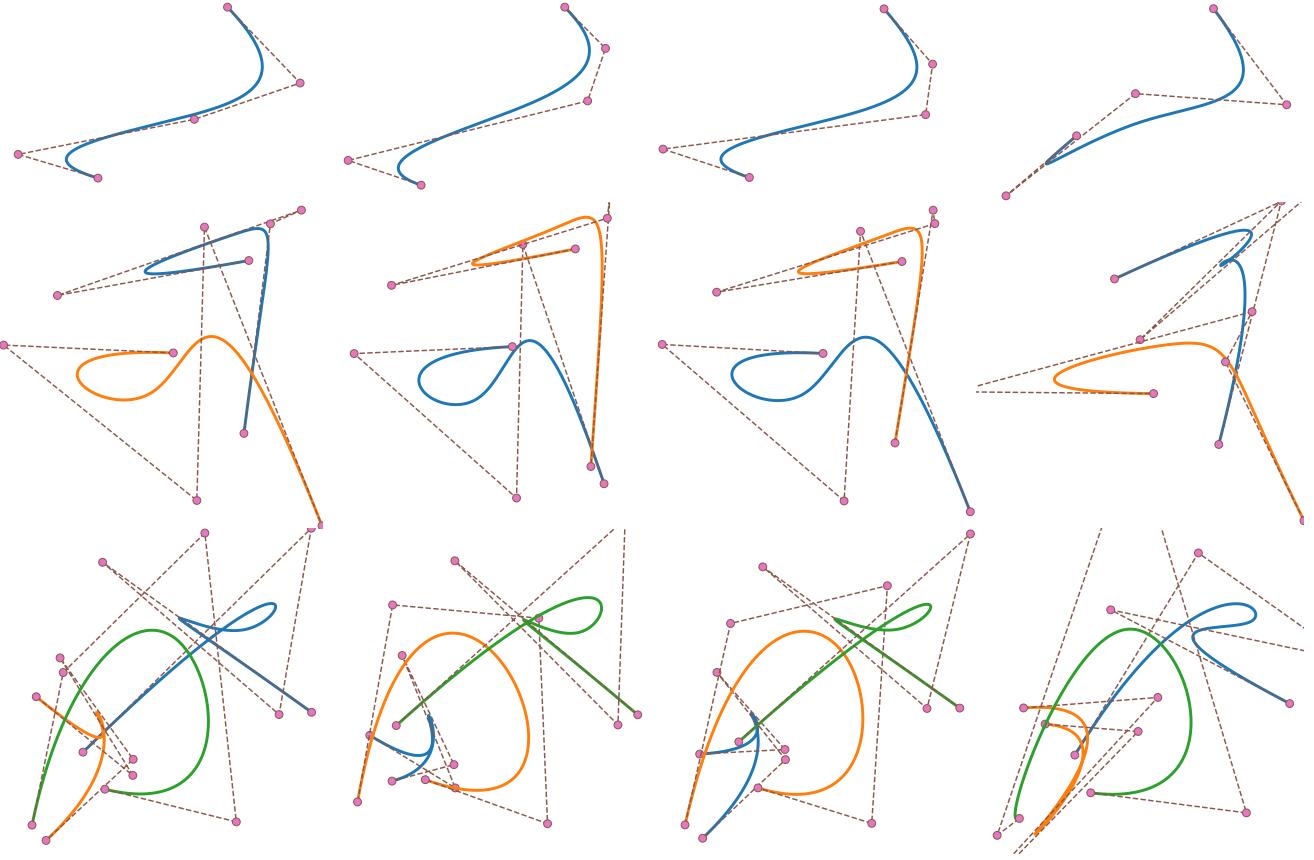


Figure 7: Randomly picked results of multiple number of spline curves with fixed number of control points. The first column shows the target curves, the second column shows the prediction from the RNN, the third column shows the reconstruction obtained by using the RNN prediction as initialization for traditional methods, and the last column shows the reconstruction obtained by using a random initialization.

the input image. Let $d = xy$ and the image feature be α .

$$\alpha = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_d\}, \mathbf{a}_i \in \mathbb{R}^l \quad (11)$$

At each time step t of the RNN, we denote the input vector as \mathbf{z}_t and hidden vector as \mathbf{h}_t

$$\mathbf{h}_t = RNN(\mathbf{z}_t, \mathbf{h}_{t-1}) \quad (12)$$

The attention network f_{att} is used to get the vector \mathbf{z}_t , the hidden vector \mathbf{h}_t is used to predict the control points and stop probability.

$$e_{ti} = f_{att}(\mathbf{a}_i, \mathbf{h}_{t-1}) \quad (13)$$

$$\mathbf{z}_t = \sum_{i=1}^d e_{ti} \mathbf{a}_i \quad (14)$$

f_{att} is a two-layer fully-connected network with ReLU activation. The inputs are \mathbf{a}_i , which corresponds to a certain part of the image, and \mathbf{h}_{t-1} , which contains the information of the curve that needs to be predicted at current step. The output is only a scalar e_{ti} which corresponds to the information about how important region i is.

4.3. Multiple Spline Curves, Variable Number of Control Point

Extending the previous model to the generation process of multiple spline curves with variable numbers of control points would consist of two nested loops. First, we loop over the curves and determine the number of curves, thereby generating one curve at each iteration of the generative process. The inner loop uses the information from the first loop that is needed to generate one curve, looping over the control points and determine the number of control points to generate the curve. This is implemented through a hierarchical RNN structure that is explained below.

4.3.1. Hierarchical RNN

We propose a Hierarchical RNN structure to model this generation procedure. This architecture has been previously used to describe visual content by attempting to understand and caption details in local regions of an image [KJKFF17]. We leverage the multiple recurrent units of a network to repeatedly extract local regions of an image containing information, and then to process this information. As before, an image feature extractor and an attention subnetwork are used to process the input image before feeding it to the RNN.

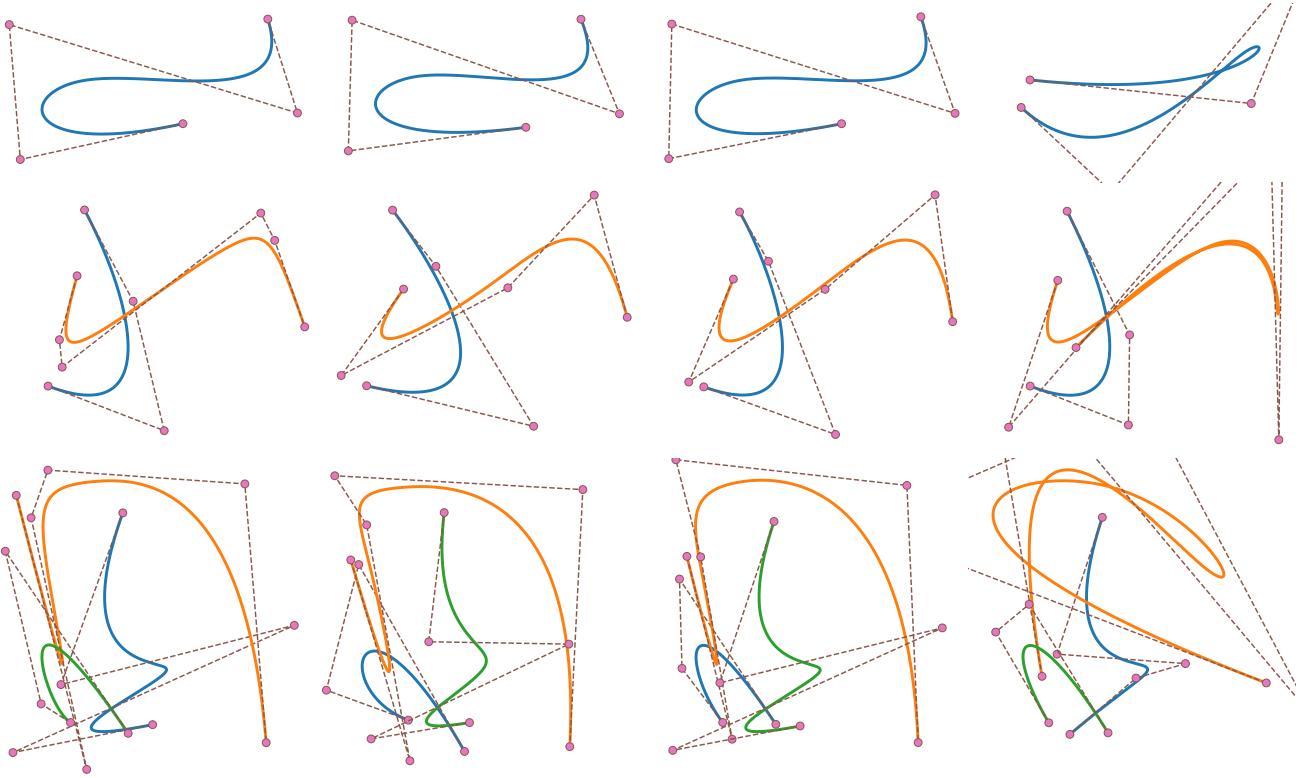


Figure 8: Randomly picked results of multiple spline curves with variable number of control points. The first column shows the target curves, the second column shows the prediction from the Hierarchical RNN, the third column shows the reconstruction obtained by using the Hierarchical RNN prediction as initialization for traditional methods, and the last column shows the reconstruction obtained by using a random initialization.

This model constitutes two RNNs combined hierarchically, one for looping over the curves (Curve RNN) and the other for looping over the control points (Point RNN). At each time step of the Curve RNN, it predicts the stop probability of the curve generation and also generates a vector representation of the current spline curve and forward it into the Point RNN. The Point RNN uses this vector representation of a spline curve and then decodes it into the position of control points. Our model is shown in Figure 5.

4.3.2. Curve RNN

The Curve RNN is a single-layer GRU with hidden size $H = 512$, the initial hidden vector \mathbf{h}_0 is predicted by a two-layer fully-connected network with the average image feature as input. At each time step, the Curve RNN receives the image feature vector \mathbf{z}_t , after passing through the attention subnetwork, as input. The hidden vector \mathbf{h}_t is then fed into a two-layer fully-connected network to obtain curve stop probability p_t and a vector representation \mathbf{v}_t using which the model predicts the curve at the current step, which is also the input to the Point RNN. This is different from what is done in Sections 4.1 and 4.2, where the hidden vector is directly fed into the two-layer fully-connected network to get positions of control points (or control point sequence).

4.3.3. Point RNN

The Point RNN is also a single-layer GRU with hidden size $H = 512$, which, given a vector representation \mathbf{v}_t from Curve RNN, is used to generate the sequence of positions of control points. We follow the network configuration in Section 4.1. At each time step, the Point RNN predicts one control point combined with a stop probability that represents whether this control point is the end point.

4.3.4. Training and Inference

The two RNNs are trained accordingly to predict the spline curves, and their stop probabilities as suggested in previous sections. The pseudocode to train this Hierarchical RNN model is provided below in Algorithm 1.

The loss function has three terms: the mean-squared error of predicted positions, the cross entropy loss of curve stop probabilities

Algorithm 1 Pseudocode for Training Hierarchical RNN

```

1: for epoch = 0 → T do                                ▷ Loop over epoch
2:   for i = 0 → N do                               ▷ Loop over all training data
3:      $\alpha = CNN(\mathbf{I}^i)$                            ▷ Get the image feature
4:      $\mathbf{h}_{curve} = Hidden\_pred_{curve}(\alpha)$  ▷ Initial hidden vector
5:     for j = 0 → L do                               ▷ Predict the attention mask
6:        $e_j = f_{att}(\mathbf{a}_j, \mathbf{h}_{curve})$ 
7:        $\mathbf{z} = \sum_{j=1}^D e_j * \mathbf{a}_j$            ▷ Get the input vector for RNN
8:     for curve = 0 →  $n_i$  do      ▷ Loop over spline curves
9:        $\mathbf{h}_{curve} = Curve\_RNN(\mathbf{z}, \mathbf{h}_{curve})$ 
10:       $\mathbf{v} = Curve\_to\_Point(\mathbf{h}_{curve})$ 
11:       $p_{curve} = Stop\_Prob(\mathbf{h}_{curve})$     ▷ stop probability
12:      for j = 0 → D do                      ▷ Predict the attention mask
13:         $e_j = f_{att}(\mathbf{a}_j, \mathbf{h}_{curve})$ 
14:         $\mathbf{z} = \sum_{j=1}^D e_j * \mathbf{a}_j$ 
15:         $\mathbf{h}_{point} = Hidden\_pred_{point}(\mathbf{v})$     ▷ Initial hidden
16:        for point = 0 →  $n_{point}$  do      ▷ Loop over Point
17:           $\mathbf{h}_{point} = Point\_RNN(\mathbf{v}, \mathbf{h}_{point})$ 
18:           $C_{point}, p_{point} = Point\_Pred(\mathbf{h}_{point})$ 

```

and of point stop probabilities:

$$L_3 = \sum_{i=1}^N \left(\min_{\delta \in \mathbf{K}^i} f_{Match}(S^i, \hat{S}^i, \delta) \right) \quad (15)$$

$$+ \lambda_1 \sum_{j=1}^{n_i} (p_j^i \log \hat{p}_j^i + (1 - p_j^i) \log (1 - \hat{p}_j^i)) \quad (16)$$

$$+ \lambda_2 \sum_{j=1}^{n_i} \sum_{k=1}^{n_{i,j}} (p_{j,k}^i \log \hat{p}_{j,k}^i + (1 - p_{j,k}^i) \log (1 - \hat{p}_{j,k}^i)) \quad (17)$$

λ_1 and λ_2 are optimization hyperparameters. n_i is the number of spline curves in the i^{th} image, $n_{i,j}$ is the number of control points in the j^{th} spline curve of image i . p is the target probability and \hat{p} is the predicted probability.

5. Unsupervised Parametric 3D Reconstruction

While the methods discussed in Section 4 could be employed to perform control point prediction when corresponding ground truth exists, solving for the parameters in the absence of ground truth is significantly harder.

Two natural obstacles are to be overcome in this setting. The first is to devise a loss function different from Mean-Squared Error that can be used to optimize the neural network. This is because one needs a ground truth to make a point-wise comparison and compute the error. This ground truth is missing in this case, and a technique to compare against a target point cloud, must be devised. We assume we have the target point cloud, which is natural and necessary in traditional methods. The second obstacle is to make the network aware of its purpose, that is to predict spline curves (or surfaces), as opposed to some other arbitrary primitives. These obstacles are entangled two-way, with the loss function design needing to take into consideration properties of spline curves (or surfaces), which might provide feedback to the neural network to determine the parameters.

Both traditional optimization methods (ICP [BM^{*}92, CM91]) and ideas in [TSSP17], which utilizes neural network for fluid simulation, provide as inspirations to solving this problem. Suppose we are provided an oracle which could predict the parameters perfectly, then the point cloud representation of the predicted curves (or surfaces) could be generated. The Chamfer distance measurement between the point clouds of predicted curves (or surfaces) and the target point clouds could be used as the loss function to train the network.

Though we use the same loss function as in the case of traditional methods, we use a learning technique to predict the control points, as opposed to optimizing them directly, since this helps us leverage the entire dataset, as opposed to considering just one sample while optimizing. The prediction learned by the network could also be used as the initialization to the traditional optimization methods, circumventing the manual initialization while reducing the number of optimization iterations. A learning based approach also equips us to deal with multiple input formats, while traditional methods only use point cloud input.

5.1. Reconstruction of Images or Point Clouds

In this section, we provide a technique to reconstruct surfaces of extrusion or revolution from images or point clouds. For generating a surface of extrusion, we extrude a spline curve at a random height. As for a surface of revolution, this is generated by revolving the spline curve around the axis by 360 degrees. We only consider spline curves without self-intersections and with monotonically decreasing y -coordinate in the control point sequence. This assumption is made since generally self-intersected surfaces are not ubiquitous. Given the input data, features are extracted from it. If this is a 2D image, the VGG network [SZ14] is used to extract the image features as in Section 4.1. If the input is a point cloud, the PointNet [QSMG16] is used to extract features. The extracted features \mathbf{v} are forwarded into a two-layer fully-connected network, which predicts the position of control points \mathbf{C} (if surfaces of extrusion are considered, \mathbf{C} also contains the extruded height). Since \mathbf{C} contains the control points, the predicted curve can be reconstructed from a linear combination of \mathbf{C} , which is dependent on the parameterization of the spline curve \mathbf{t} . This is represented as follows.

$$\mathbf{C} = NN(\mathbf{v}) \quad (18)$$

$$P_{pred} = \mathbf{f}(\mathbf{t})(\mathbf{C}) \quad (19)$$

where NN denotes the two-layer fully-connected network, and \mathbf{f} is the basis function dependent on the spline generator \mathbf{t} .

We use the Chamfer distance to measure the distance between the predicted point cloud and target point cloud.

$$d_{CD}(P_1, P_2) = \sum_{x \in P_1} \min_{y \in P_2} \|x - y\|_2^2 + \sum_{y \in P_2} \min_{x \in P_1} \|x - y\|_2^2 \quad (20)$$

Since d_{CD} is differentiable, it is possible to train the network end-to-end. The function $\mathbf{f}(\mathbf{t})(\cdot)$ represents control point weights to generate the surface. This is implemented as a linear layer with

fixed weights in the network. This architecture is generalizable, since predicting other kinds of surfaces (like surfaces of sweeping or NURBS), would require only a change of this individual layer, with the rest of the model remaining the same.

6. Experiments

Training deep neural networks usually requires a copius amount of data. However, since there isn't enough real data with ground truth spline curve labeling, we use randomly synthesised data to perform our training and testing experiments. We synthesize a dataset of size 500,000, with a 70-30 train-to-test split. For each instance of the dataset, multiple spline curves and control points for each of the spline curves are generated. The training dataset consists of self-intersecting or looping spline curves, but do not contain any closed spline curves. This could be done by adding a circular loss term to the loss function in Equation 1. But for the purpose of this paper, we shall not venture into closed curves. According to the problem we attempt to solve, we generate the dataset and perform the training. We also run the trained model on real images to check the generalizability of our model.

For 2D reconstruction, we set the image size to 128×128 and randomly generate the number of spline curves and the number and the position of control points for each spline curve in 2D image plane. The number of control points are varied between 4 to 6 for the problem with spline curve reconstruction with variable control points. The number of spline curve are varied between 1 to 3 for the problem of reconstructing multiple spline curve with the number of control points fixed at 5. These two variations are combined to train and test for multiple spline curve reconstruction with variable number of control points. For 3D reconstruction, the number of control points are fixed to 5, and then the spline curve is revolved or extruded to gain a 3D surfaces. If the input is an image, we render the 3D surface with fixed camera angle and lighting condition. If the input is point cloud, we randomly sample from the 3D surface.

We use the VGGNet as image feature extractor for all the three models in Sec. 4. We initialize our network using pretrained weights from ImageNet [DDS^{*}09]. We then train the network with Adam optimizer through back-propagation. We set both the learning rate and weight decay to 10^{-4} . Determining the number of points (or curves) is easier than predicting the position of the control points. Therefore, the hyperparameter λ is set to 0.1 to lay more emphasis on position regression. Our models are implemented in PyTorch and trained for 100×10000 iterations with batch size set to 32. It took approximately one to two days on GeForce GTX 1080 GPU to converge. Testing on one image needs only 12ms.

Measuring the performance of our model is not a straightforward task. Due the ambiguity that could potentially be caused by two shape-similar curves having totally different control point sequences, or even different number of control points, one single measurement would not be able to cover all the characteristics of spline curves. We use three measurements here that complement each other: Mean-Squared Error between the position of predicted control points and target control points, Classification accuracy of the number of control points and the number of spline curves and Chamfer distance between point clouds of the predicted spline curves and the target curves.

6.1. Baseline Method

Traditional methods minimize a distance metric measured directly between a pair of geometric entities: the reconstructed surface and a target often provided as a point cloud. In each iteration, the target points are projected onto the predicted surface (and vice versa) to obtain point-to-foot-point matches. As this distance can be expressed through the parameters of the prediction, minimizing the distance updates the parameters. Throughout the optimization, at each iteration, the distance and the foot-points are reevaluated. Faster algorithms based on normal-to-normal distances and point-to-tangent-plane distances have been used to accelerated the process, with an essentially similar idea of minimizing a local distance metric [BM^{*}92, CM91, WPL06, TBWP16]. For all our experiments, we minimize the point-to-point distance as the traditional method to compare against.

6.2. Results

We present results pertaining to both 2D spline curve reconstruction and 3D surface of extrusion or revolution reconstruction. As mentioned in Section 4 and 5, the method that is employed to perform the 2D spline reconstruction is supervised, while the one used to reconstruct the surface of extrusions or revolutions is unsupervised.

6.2.1. Supervised 2D Reconstruction

We have proposed three supervised models in Sections 4.1, 4.2 and 4.3 aiming to reconstruct single spline curve with variable control points, variable number of spline curves with a fixed number of control points and variable number of spline curves with variable control points respectively. In this section, we showcase the reconstruction performance of these individual models, as well as in comparison to traditional energy-based models.

Single Spline Curve, Variable Number of Control Points.

We attempt to show that traditional energy-based models that optimize to reconstruct a spline curve with fixed number of control points as input, show significantly improved performance when initialized with control points that are more systematically learned by our method, as opposed to a random initialization that is classically performed. To this effect, we randomly initialize the position of control points and use energy-based methods to perform the optimization. Since the traditional method uses a fixed number of input control points, we perform this optimization for three different cases of initial control points (4, 5, 6) and select the best optimized case as the final result. We observe that the performance of the learned initialization easily outperforms the random initialization. This is expected, since the systematic learning of the control points leverages the information contained in the training datasets about the relative position of the control points to the curves, and this can provide us with a very good starting point to perform the optimization from. This is observed in Figure 6. It is observed that the prediction by our RNN is close enough to the original curve, and therefore serves as an excellent initialization for control point optimization. After optimization based on this initialization, the obtained reconstruction almost exactly mirrors the input curves.

Variable Spline Curves, Fixed Number of Control Points.

The method discussed in Section 4.2 is used to perform predictions of number of spline curves and the control points for each of the spline curves. In Figure 7, we again perform the comparisons between the target and the predictions of the RNN, the optimization-based reconstruction dependent on initializations, both random and based on RNN output. We again observe that the prediction of the RNN, when used as an initialization, comfortably outperforms the random initialization.

Variable Spline Curves, Variable Number of Control Points

The method discussed in Section 4.3 is used to perform predictions of a variable number of spline curves, each containing a variable number of control points. We perform the comparisons between the target and the random and RNN-based initialization and reconstruction schemes again in Figure 8. It is also to be noted that in Figure 8, as in the case of the orange curve in the second row, in spite of prediction of number of control points being wrong and predicted positions also being wrong, the predicted and target curves look similar, which provide us with the intuition that the RNN architecture learns the curves though it does not necessarily learn it in the specific manner we intend it to.

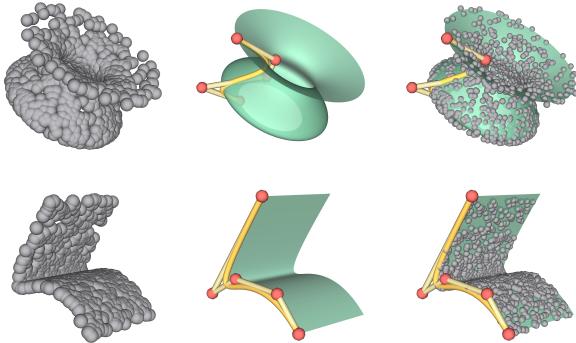


Figure 9: Randomly picked results of point cloud reconstruction. The left column consists of input point clouds, the middle column consists of generator splines revolved (top) or extruded along a path (bottom) to generate surface reconstructions. We plot the input point clouds and predicted surfaces in the right column.

The optimization results after the RNN-based initialization are an order of magnitude better than the optimization results after a random initialization. This can be quantitatively validated over the entire test set, that has been synthetically generated with random positions of control points and variability in the number of spline curves and control points. This evaluation is indicated in Table 1 as well, where V refers to the case of using variable number of control points on single spline curve, M refers to the case of using multiple spline curves with fixed number of control points and MV refers to the case of using multiple spline curves each containing a variable number of control points.

We also show other quantitative measures in Table 2. Here, we compute the Mean-Squared Error in curve prediction and accuracy in computing number of points and number of curves, as well as the Chamfer distance between the reconstructed prediction and the

target curves. It is to be noted that in the case of variable curves with fixed control points, the point accuracy is not an applicable measure since it is known beforehand, and the same holds for curve accuracy in the case of variable control points for single curve. It is observed that in spite of the fact that the performance of curve accuracy and point accuracy both drop in the case of multiple spline curves with variable control points, the drop is very minor and the performance is still excellent by all comparable measures as shown in Table 1.

	NN	NN Init	Random Init
V	1.38×10^{-3}	2.30×10^{-5}	3.33×10^{-4}
M	1.93×10^{-3}	7.74×10^{-5}	8.25×10^{-4}
MV	2.22×10^{-3}	9.53×10^{-5}	7.39×10^{-4}

Table 1: Chamfer distance between the predicted curve and target curve. “NN” denotes the average distance of target from Neural Network prediction, “NN Init” denotes the average distance of target from the optimization performed with the Neural Network prediction as initialization, and “Random Init” denotes the same distance with optimization performed with random initialization. The optimization here is performed using traditional methods as discussed in Section 6.2. The distances are all normalized to lie in the unit interval. The rows refer to the three modes of variability on which we operate our networks as described in Section 6.2.

	MSE	Point Acc	Curve Acc	Chamfer Distance
V	0.01302	94.58	N/A	1.38×10^{-3}
M	0.02699	N/A	99.85	1.93×10^{-3}
MV	0.03738	82.75	99.49	2.22×10^{-3}

Table 2: Error and Accuracy measures. Quantitative evaluation of Mean Squared Error, point prediction accuracy, curve prediction accuracy and the Chamfer distance between the predicted and target curves are provided. The accuracy terms are provided in percentages. The distances are all normalized to lie in the unit interval. The rows refer to the three modes of variability on which we operate our networks as described in Section 6.2.

6.2.2. Point Cloud Reconstruction

The reconstruction of surfaces of extrusion or revolution from inputs in the form of point clouds is performed as described in Section 5.1. A sampling of these reconstructions is shown in Figure 9, where spline curve prediction to resemble the point cloud when revolved or extruded is performed in an unsupervised manner.

6.2.3. Real/Synthetic Image Reconstruction

MNIST Spline Reconstruction. We use the trained networks to reconstruct images in the MNIST dataset [LBBH98]. The networks have been trained on synthetically generated data, and testing it to perform real data reconstruction, as is the case of the MNIST dataset, is overreaching of the capabilities of the network. Nevertheless, it is to be noted that we perform significantly well on this

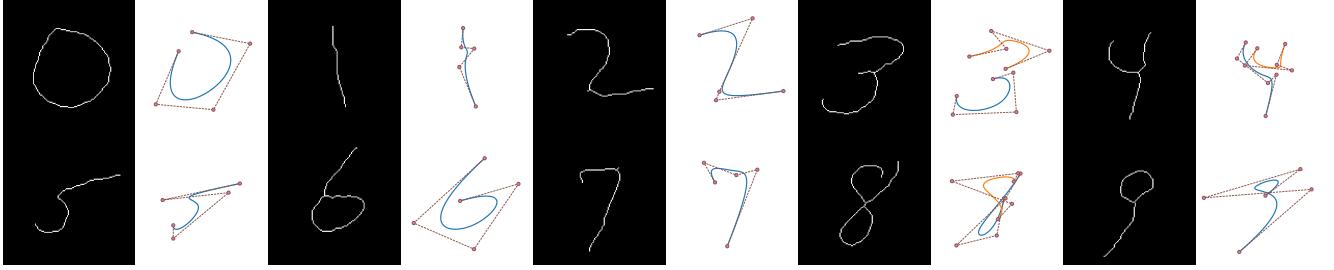


Figure 10: MNIST reconstruction. Reconstruction of MNIST numerals 0-9 is shown. The left sub-image of each of the images is the original MNIST numeral, and the right sub-image is the reconstructed image from our method. The mode that is used is the one with variable number of curves and fixed number of control points for each curve.

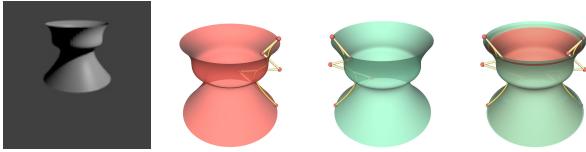


Figure 11: Results of surfaces of revolution on images. Left to Right: Input synthetic image, ground truth surface, predicted surface, ground truth and predicted surface plotted together.



Figure 12: Results of surfaces of revolution on real image. The image on the left is the real image, and one on the right is the reconstructed surface..

dataset, especially when the input images are more curved in nature, with less sharp edges. A sample of these reconstructed results are shown in Figure 10.

We first enlarge the MNIST image to size 128×128 , and then thin the digits to lines, which are the input to our network. It is observed that the mode of multiple spline curves with fixed number of control points performs best for this task. It is also to be noted that no post-processing is performed on the result.

Surface Reconstruction. We perform testing on real images to generate surfaces of revolution. As a preprocessing step, we convert the photo to a gray-scale image, crop and pad it properly to fit the input size of the neural network. An instance of this reconstruction, performed as described in Section 5.1 on a synthetic input image, is shown in Figure 11. This can also be performed on real images as illustrated in Figure 12.

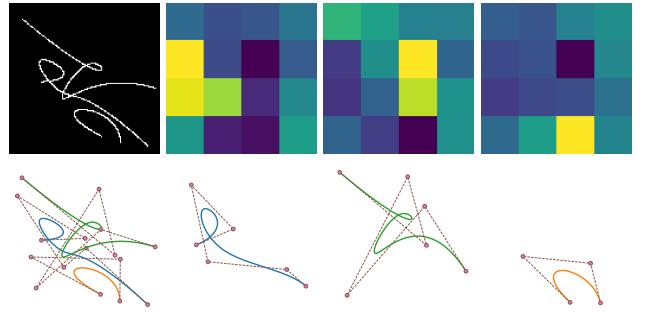


Figure 13: Attention results for Hierarchical RNN. The first row shows the input image, and the maps provided by the attention networks for the prediction of each of the individual spline curves. The second row shows the target curves and the predictions of spline curves one at a time, corresponding to the attention map provided in the first row.

6.2.4. Visualization and Analysis

The attention map in the Hierarchical RNN is plotted at different curve-prediction iterations in Figure 13. This gives us a method to visualize how the networks learns to pick the next curve to predict. In each iteration, the network tends to focus on the regions of the image that correspond to the curve that is being predicted currently. The attention region in these maps is usually close to the center of the curve. Since the operation of convolution is repeated applied, as the network gets deeper, the center part of an object or curve tends to contain more information than the marginal parts, and thus this part deserves more attention.

6.2.5. Failure Cases

We showcase a number of failure cases for our method in Figure 14. It is observed that when multiple spline curves are heavily entangled, the model seems to fail. These entanglements are difficult to separate through manual human supervision, and so it can be expected with fairly high chance that this would fail, and this is what is observed in the figure.

We also showcase some failure cases in the reconstruction of

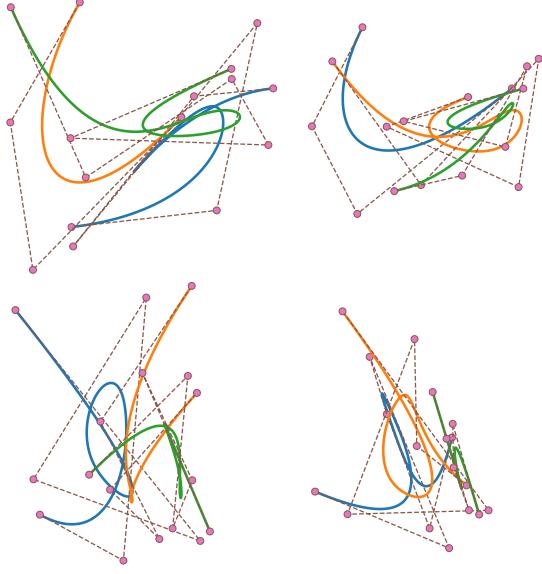


Figure 14: Reconstruction failure. Failure cases of multiple spline curves with variable number of control points

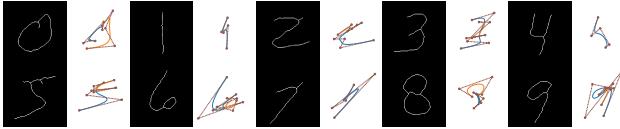


Figure 15: Failure cases at MNIST reconstruction. Reconstruction of MNIST numerals 0-9 is shown. The left sub-image of each of the images is the original MNIST numeral, and the right sub-image is the reconstructed image from our method. The mode that is used is the one with multiple number of curves and variable number of control points for each curve.

MNIST numerals in Figure 15. This can be attributed to the lack of training on real images. There are multiple issues with using the MNIST images as test images. One issue that we run into often is that of closed curves. The training data does not contain closed spline curves, but in MNIST closed curves are ubiquitous, such as numerals 0,6,8,9 in the Figure 15. This tends to be alien to the network when it is seen in the MNIST images. Another issue is that images such as numerals, which are sharper in nature, than plain spline curves, and thus the curves that are predicted, while they attempt to approximate the input image, they are not expressive enough to be able to make a close enough approximation. Real images also contain some noise, such as unnecessary pixels, which might lead the network to predict complicated curves to account for the noise (numerals 1, 2, 5, 7 in the Figure 15). But we believe these issues could be mitigated with some domain adaptation techniques.

There are a few limitations to our methods. For 2D reconstruction, our methods only consider the family of spline curves where the knot position in the splines is fixed. If both the control points

and the knots need to vary, our methods need modifications. For 3D reconstruction, our methods only consider surfaces with no self-intersections and decreasing y-coordinates to avoid the local minima problem. Finally, due to lack of training data, we train our model on synthetic data and test it on real data. While we do this, we do not apply domain adaptation techniques to help the network generalize well to real data, but this is a problem which could be attempted as a future work.

7. Conclusion and Future Work

In this paper, we have illustrated approaches to reconstruct spline curves and surfaces using data driven approaches. Being both different and complementary to the traditional methods of spline fitting, our methods, adaptive to different forms of inputs, do not need initialization and can handle variable number of control points. There are many exciting directions that remain to be explored. A viable future direction would be to investigate methods to detect, decompose, and recover multiple parametric surfaces from single images and consistently assemble them across multiple images. Another area of interest would be to study how other types of information, such as semantics and physics, can be utilized to design and reconstruct clean, complex, and functional geometry and structures.

References

- [BCB14] BAHDANAU D., CHO K., BENGIO Y.: Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014). 3
- [BM*92] BESL P. J., MCKAY N. D., ET AL.: A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence* 14, 2 (1992), 239–256. 2, 8, 9
- [Can86] CANNY J.: A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 8, 6 (June 1986), 679–698. 3
- [CKUF17] CASTREJÓN L., KUNDU K., URTASUN R., FIDLER S.: Annotating object instances with a polygon-rnn. In *CVPR* (2017). 2
- [CM91] CHEN Y., MEDIONI G.: Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on* (1991), IEEE, pp. 2724–2729. 2, 8, 9
- [CVMBB14] CHO K., VAN MERRIËNBOER B., BAHDANAU D., BENGIO Y.: On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014). 4
- [CVMG*14] CHO K., VAN MERRIËNBOER B., GULCEHRE C., BAHDANAU D., BOUGARES F., SCHWENK H., BENGIO Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014). 3
- [CXG*16] CHOY C. B., XU D., GWAK J., CHEN K., SAVARESE S.: 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision* (2016), Springer, pp. 628–644. 2
- [DDS*09] DENG J., DONG W., SOCHER R., LI L.-J., LI K., FEI-FEI L.: Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 248–255. 9
- [FLB16] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 120. 2

- [FSG16] FAN H., SU H., GUIBAS L.: A point set generation network for 3d object reconstruction from a single image. *arXiv preprint arXiv:1612.00603* (2016). [2](#)
- [HB89] HORN B. K., BROOKS M. J.: *Shape from shading*. MIT press, 1989. [2](#)
- [Hor70] HORN B. K.: Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. [2](#)
- [Ken79] KENDER J. R.: Shape from texture: An aggregation transform that maps a class of textures into surface orientation. In *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1* (1979), Morgan Kaufmann Publishers Inc., pp. 475–480. [2](#)
- [KIO*16] KUMAR A., IRSOY O., ONDRUSKA P., IYYER M., BRADBURY J., GULRAJANI I., ZHONG V., PAULUS R., SOCHER R.: Ask me anything: dynamic memory networks for natural language processing. *international conference on machine learning* (2016), 1378–1387. [4](#)
- [JKKFF17] KRAUSE J., JOHNSON J., KRISHNA R., FEI-FEI L.: A hierarchical approach for generating descriptive image paragraphs. In *Computer Vision and Pattern Recognition (CVPR)* (2017). [6](#)
- [KSH12] KRIZHEVSKY A., SUTSKEVER I., HINTON G. E.: Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, Pereira F., Burges C. J. C., Bottou L., Weinberger K. Q., (Eds.). Curran Associates, Inc., 2012, pp. 1097–1105. [3](#)
- [LBBH98] LECUN Y., BOTTOU L., BENGIO Y., HAFFNER P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324. [10](#)
- [QSMG16] QI C. R., SU H., MO K., GUIBAS L. J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593* (2016). [8](#)
- [RT15] ROMERA-PAREDES B., TORR P. H. S.: Recurrent instance segmentation. *CoRR abs/1511.08250* (2015). [4](#)
- [SSISI16] SIMO-SERRA E., IIZUKA S., SASAKI K., ISHIKAWA H.: Learning to simplify: fully convolutional networks for rough sketch cleanup. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 121. [2](#)
- [SSWF15] SUKHBAATAR S., SZLAM A., WESTON J., FERGUS R. D.: End-to-end memory networks. *neural information processing systems* (2015), 2440–2448. [4](#)
- [SVL14] SUTSKEVER I., VINYALS O., LE Q. V.: Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (2014), pp. 3104–3112. [3](#)
- [SZ14] SIMONYAN K., ZISSERMAN A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014). [4, 8](#)
- [TBWP16] TANG C., BO P., WALLNER J., POTTMANN H.: Interactive design of developable surfaces. *ACM Transactions on Graphics (TOG)* 35, 2 (2016), 12. [2, 9](#)
- [TDB17] TATAROVENKO M., DOSOVITSKIY A., BROX T.: Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *IEEE International Conference on Computer Vision (ICCV)* (2017). [2](#)
- [TSSP17] TOMPSON J., SCHLACHTER K., SPRECHMANN P., PERLIN K.: Accelerating Eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning* (International Convention Centre, Sydney, Australia, 06–11 Aug 2017), Precup D., Teh Y. W., (Eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 3424–3433. [8](#)
- [VTBE15] VINYALS O., TOSHEV A., BENGIO S., ERHAN D.: Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 3156–3164. [3](#)
- [WPL06] WANG W., POTTMANN H., LIU Y.: Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Transactions on Graphics (ToG)* 25, 2 (2006), 214–238. [2, 9](#)
- [XBK*15] XU K., BA J., KIROS R., CHO K., COURVILLE A., SALAKHUDINOV R., ZEMEL R., BENGIO Y.: Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)* (2015), Blei D., Bach F., (Eds.), JMLR Workshop and Conference Proceedings, pp. 2048–2057. [4](#)
- [ZBLW12] ZHENG W., BO P., LIU Y., WANG W.: Fast b-spline curve fitting by l-bfgs. *Computer Aided Geometric Design* 29, 7 (2012), 448–462. [2](#)