From Sketches to 3D Models

RnD Report

of

Akkapaka Saikiran

Indian Institute of Technology Bombay

Under the supervision of

Prof. Parag Chaudhuri

Indian Institute of Technology Bombay

Abstract

3D modeling in industry is a tedious process which requires skill and patience. Sketch-based modeling systems attempt to generate 3D models from user-drawn 2D or 3D sketches for time-sensitive applications like prototyping. In this report, we recount the literature surveyed, ideas brainstormed, and approaches tried in a four-month long research project whose goal was a novel approach to the above problem, the core idea being generating a set of smoothly connected Bézier patches from a sequence of 3D strokes.

1 Introduction

3D modeling is widely used in various industries like film, animation, gaming, interior design, and architecture. Commercially, softwares like Blender [1] and Maya [2] are employed for this task as they provide a lot of functionality and expressive power to the user. But these tools have a steep learning curve and are usually used by professions or enthusiasts.

For most people though, sketching is a natural way of representing objects pictorially. So a lot of research has been done on converting sketches to 3D content. There is a lot of variety among the different approaches devised to solve this problem. The input sketches can be 2D or 3D. Some approaches, to be able to generate complex models, require the user to sketch strokes from multiple views ([3]), while others try to create a model from a single-view sketch to minimize user input ([4]). Some approaches try to use very few sketches ([5]), some need many ([6]). Some approaches use annotated strokes in their algorithm ([7]). Some approaches perform retrieval from a large database of 3D models ([8]), some use ideas of inflation and extrusion ([9]), and some infer a feature set to construct the model ([10]).

Most methods generate a mesh representation of the model. We attempt to generate parametric surfaces; to be precise we want to generate a set of Bézier

Figure 1: Some sketches from Poly [11], an online repository of VR drawings

patches which have smooth joins amongst them. This join criteria can be expressed in terms of C1 or C2 continuity. We aim to take the input as sketches in Virtual Reality (VR), where sketching is akin to sculpting, as the user can physically move around and work on the sketch from any direction. Sketching in VR does require more skill than drawing on paper or in a 3D environment on a Desktop, but applications like Tilt Brush [12] and Gravity Sketch [13] have made advances in making this process intuitive and easy-to-learn.

The layout of this report is as follows. In section 2 we present a high-level overview of a couple of existing approaches of creating 3D models from sketches. In section 3 we present our approach and split it into two parts. We also review the problem of point cloud classification in this section. In section 4 we discuss our approaches to generate a dataset as our setup is data-driven and learning-based. In section 5 we discuss our thoughts on devising an algorithm to associate strokes to patches, while also reviewing an older approach on the same. Finally, in section 6, we make some concluding remarks and thoughts.

2 The Lay of the Land

A good place to start would be Teddy [9]. More than two decades ago, Igarashi et al. developed an easy-to-use sketching system, well-suited for rapid construction

Figure 2: Teddy in use on a display-integrated tablet

Figure 3: Some models created using Teddy

of 3D free-form objects. It is a draw-rotate-draw model, which is interactive and real time. It supports many operations - creation, extrusion, painting, cutting, etc. as shown in 4. However, it can only generate simple objects, whose polygonal mesh representations are homeomorphic to spheres, i.e. objects which can be smoothly transformed to a sphere without cutting or pasting. Thus, the generated objects can't have holes. Further, all models were smooth, so this method is unsuitable for CAD and related applications, which require precision. Nevertheless, this seminal work was the stepping stone for a lot of research.

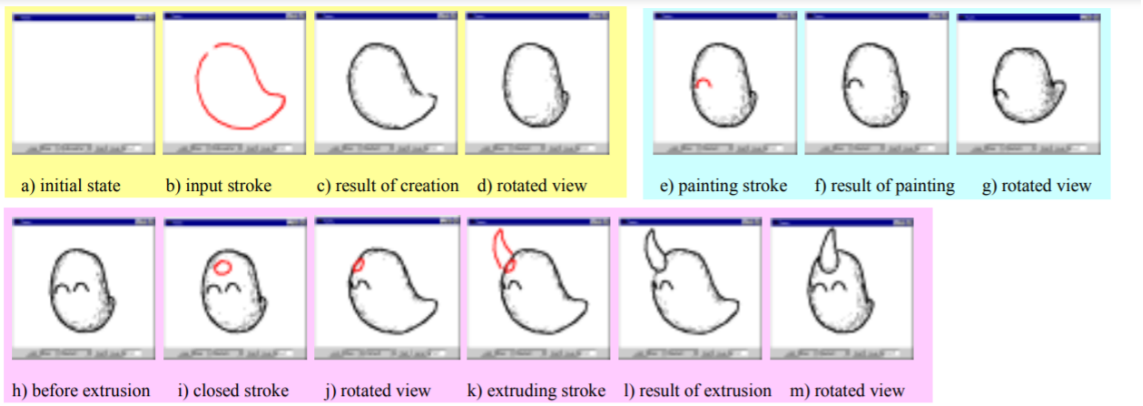

Figure 4: An overview of some modeling operations of Teddy

Han et al. [10] in 2017 propose a deep learning based sketching system for 3D face and caricature modeling. The application of this lies in low-cost interactive modeling, for eg avatars and animojis for social media or face-related design and art. The labor-efficient sketching interface allows a user to draw freehand sketches as if drawing on paper. The user interface provides a follow-up mode where users can refine or add lines to the sketch, and they can see the changes made on the

3D face inferred. A gesture-based refinement mode is also supported where shape refinement can be further performed directly on the 3D model.

Figure 5: Red: silhouette, blue: contour/feature lines, black:wrinkle lines

A natural way to draw 2D faces is to first draw the outermost silhouette first and then fill in with contours describing the eyes, nose, mouth, etc. More strokes can be drawn to give extra information about the shape of the face, but just the silhouette and the contour give a good depiction of the face even though they are a sparse set of curves. So the authors believe 3D face features and correlations can be learnt from data.The stokes are separated into three categories, silhouette lines, feature lines, and wrinkle lines, as shown in figure 5.

The system uses a deep regression model which has two branches. One branch rasterizes the sketch and feeds the input as a binary image to a CNN which computes a fixed length encoding of all strokes in the sketch. But because CNNs have a limited receptive field, a second branch samples a fixed number of points from the silhouette and the feature lines and created a shape-level vector via a bilinear model. The authors also build a repository of 15000 registered face models; formed by 150 identities, 25 expressions, and 4 levels of exaggeration. The final output of the regression network describes a new face model whose vertices are some combination of the faces in the repository.

Thus the system generates 3D faces. Figure 6 shows how the generated models look. Now we move on to our take on the problem.

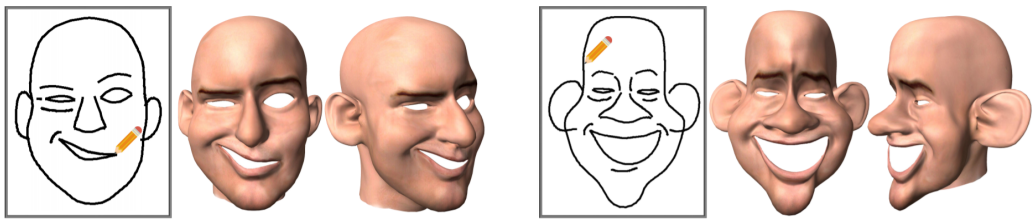

Figure 6: Models created by an amateur user in less than 10 minutes

3 Sketch-based modeling using parametric surfaces

Our approach to contribute a novel method for sketch-based modeling was to generate a set of smoothly connected Bézier patches which would try to approximate the user's intention. The input would be 3D strokes, where each stroke is a sequence of points, so can be thought of as an ordered point cloud. Our eventual

goal was to do this in Virtual Reality (VR), where the user would be wearing a head-mounted device and a hand-held controller which would be tracked to obtain the strokes. However, we started off with the Desktop setting, where the drawing would happen in a 3D environment like Blender's view port.

3.1 A Tale of Two Parts

This problem was broken up into two parts. The first part would try to fit a Bézier patch to some subset of the strokes seen so far. The second part would try to predict if the current stroke is part of the ongoing patch or not. This can be viewed as a sequence segmentation problem with the objective of predicting which patch each drawn stroke belongs to. The second part would tell the first part whether to continue extending the ongoing patch or to switch to a different (potentially new) patch. These two parts would then be joint-trained to obtain a system capable of fitting a set of Bézier patches to the drawn strokes. C1 or C2 continuity between the patches could be encoded somewhere as a constraint so as to generate visually appealing and realistically plausible models.

The first part already had some prior work done on it, so the focus of the project was on the second part. Since strokes are point clouds, let us take a small detour to look at segmentation and classification on point clouds, with a focus on a particular method involving a neural network model called DGCNN [14] .

3.2 Learning on Point Clouds

Point clouds are a flexible representation of geometric data which constitute the raw output of many 3D data acquisition devices. They are used by applications aplenty, for example indoor navigation, self-driving vehicles, robotics, and shape synthesis and modeling. However, point clouds lack topological information and do not encode any semantics about the object or parts of the object they represent. Assigning semantic labels to point clouds (or to individual points) is a high-level task, better done by data-driven learning-based methods rather than traditional methods which use handcrafted features.

Deep learning shot to fame with it's success on image processing tasks. Standard deep neural networks work on structured data like images or text. However, point clouds are inherently unstructured. They are distributed continuously in space and in general their semantics are rotation- and permutation-invariant. One approach to deal with the irregularity is to convert a point cloud into a volumetric representation, i.e a 3D grid. However, this introduces quantization artifacts and the richness in the data is usually lost. Volumetric representations also fail to capture fine-grained features.

One pioneering work on processing point clouds directly was PointNet [15]. It embraced the permutation-invariance by operating on all points individually be-

fore applying a symmetric function to accumulate features. But this treats points largely independently and fails to capture relationships between points. So Wang et al. [15] proposed an operation called the EdgeConv which generates edge features instead of point features by explicitly constructing a local graph. Thus the model learns to group points semantically by maintaining a dynamic graph.

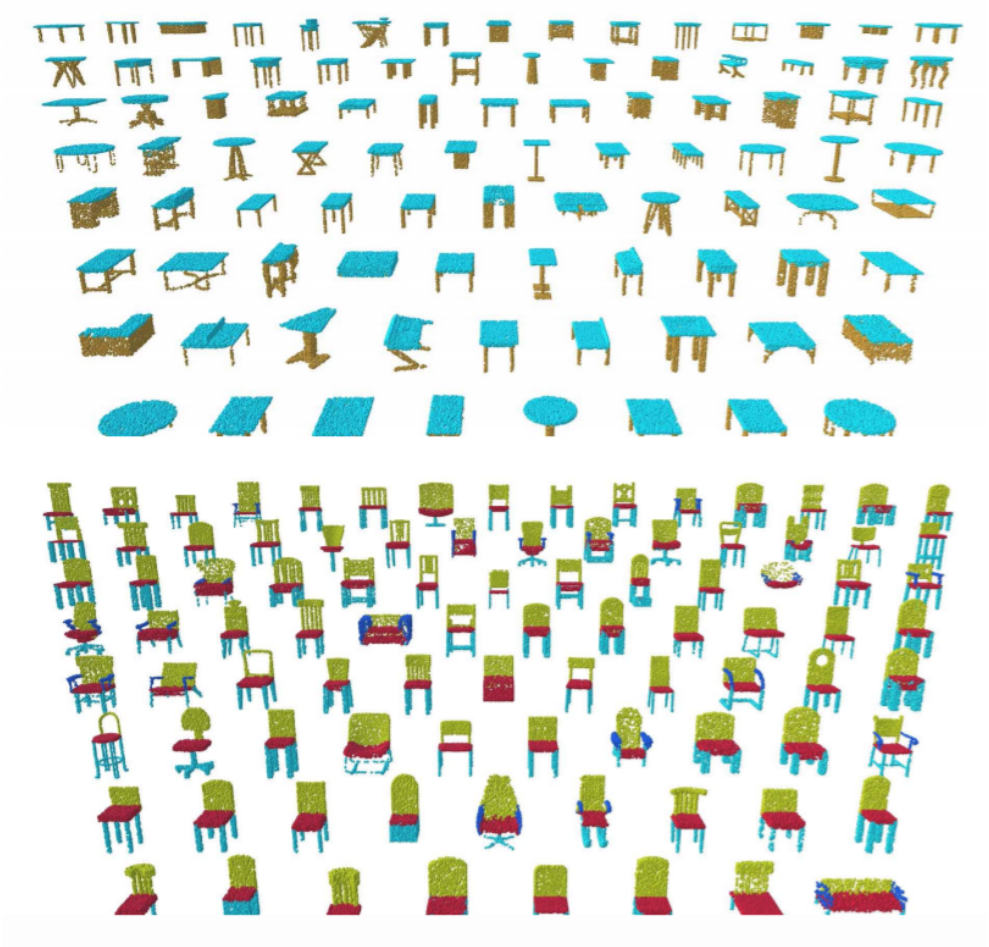

Figure 7: Some results of DGCNN [14] on the part segmentation problem

Formally, consider an intermediate layer of features $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where each $\mathbf{x}_i \in \mathbb{R}^F$ for some $F \in \mathbb{N}$. For the input layer, $F = 3$, because the data consists of 3D points. A k-nearest-neighbours (k-NN) graph $G = (V, E)$ with $V = \mathbf{X}$ is constructed such that $(\mathbf{x}_i, \mathbf{x}_j) \in E$ if $\mathbf{x}_j$ is among the k nearest neighbours of $\mathbf{x}_i$ (using the standard Euclidean distance in $\mathbb{R}^F$). Let $e_{ij} = h_\theta(\mathbf{x}_i, \mathbf{x}_j)$ be edge features, where $h_\theta : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ is an activation function with learnable weights θ. Aggregating all edge features gives us the output of an EdgeConv layer. So the output $\mathbf{x}'_i$ of the

EdgeConv operation is:

$$\mathbf{x'}_i = \square_{j:(i,j)\in E} h_\theta(\mathbf{x}_i, \mathbf{x}_j)$$

Taking $h_\theta(\mathbf{x}_i, \mathbf{x}_j) = h_\theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$ gets us an asymmetric edge function where the $\mathbf{x}_i$ component corresponds to global structure and the $\mathbf{x}_j - \mathbf{x}_i$ component is related to local information. Observe how if $h_\theta(\mathbf{x}_i, \mathbf{x}_j) = h_\theta(\mathbf{x}_i)$ we arrive at the model proposed by PointNet, since the edge feature of $(\mathbf{x}_i, \mathbf{x}_j)$ only depends on $\mathbf{x}_i$, so this is equivalent to treating the points independently.

A k-NN graph is thus constructed at each layer, forming a feedforward network. Note that every pass through the network modifies the graph as distances between the activations change, which is why this network is called a Dynamic Graph CNN. In figure 7 we can see some results of DGNN on the part segmentation problem.

4 Generating a dataset

The primary task then, to kickstart things, was to come up with a dataset for the second task. As explained in section 3.1, we had a model which could fit a Bézier patch to a set of strokes. The dataset generation for this model involved sampling strokes from surfaces. Using this sampling machinery, we could associate have strokes associated with patches once we could get hold of a collection of smoothly joined Bézier patches.

There already exist popular and established datasets of 3D shapes in the form of ModelNet [16] and the ABC Dataset [17], and these have mesh representations. We thus decided to try fitting parametric surfaces to meshes. There did not seem to be much recent literature on this, so perhaps this is an interesting and useful problem in its own right. However, we discussed one particular paper, whose approach we present next in brief.

4.1 Adaptive patch-based mesh fitting

There exist methods to fit parametric surfaces to meshes with low fitting error, but the continuity of the fitted patches is not often considered. Lin et al. [18] present an algorithm in which a triangular mesh model is fitted with bi-quintic Bézier patches with G1 continuity. The algorithm is adaptive, so the fitting error can be made arbitrarily low by subdivision and iteration.

The approach is as follows. The input mesh is first segmented into quadrilateral patches (or submeshes), where the boundary of each patch is a weakly four-sided closed curve. See figure 8 for a better understanding. Post this, normal curves are constructed for each of the four edges of a quandrilateral submesh by performing fitting on the normal vectors of the edge's mesh vertices. These normal curves are by design quadratic Bézier curves. Normalized chord length parametrization is used and a least squares objective is solved where fitting error is minimized.

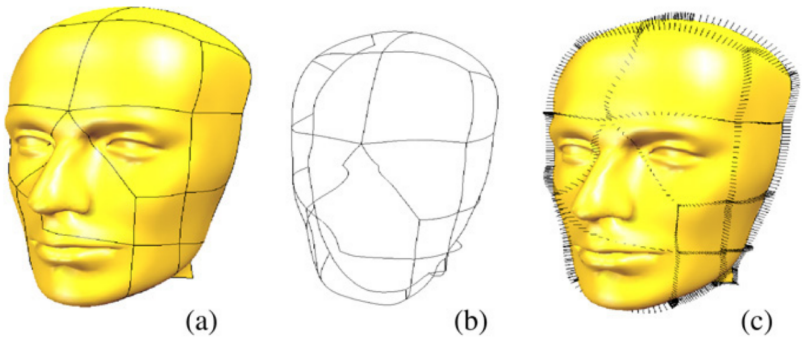

Figure 8: (a) A mesh segmented into quadrilateral patches, (b) The boundaries of the quads, (c) Normal curves on these boundaries

Next, a boundary-fitting curve, which is chosen to be a quintic Bézier curve, is constructed. This involves a constrained least squares problem where we require that this curve be perpendicular to the normal curve. The next step is patch fitting, where a bi-quintic Bézier patch is constructed which interpolates the boundary and normal curves on its four boundaries. Because patches fitted to adjacent quadrilateral submeshes both have the same normal vectors on the joining boundary, G1 continuity is ensured. The final step is to perform fitting adaptively till the fitting error is below a given threshold; i.e. if the fitting error is above the threshold, subdivide the quad into four quads and perform fitting.

Our plan was to first try to implement this algorithm on some standard dataset and see if we could improve the fitting to G2 continuity using bicubic Bézier surfaces.

4.2 The ABC dataset

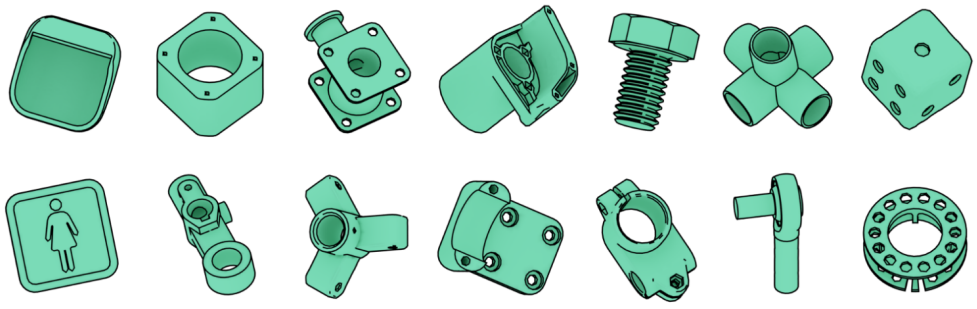

Figure 9: Some random objects from the ABC dataset

The ABC dataset [17] (A Big CAD model dataset) is a collection of one million

Computer-Aided Design (CAD) models. Each model is represented as a collection of parametric curves and surfaces. By sampling from these surfaces, the dataset authors also provide a mesh representation of each model. Each curve is one of the following types: line, circle, ellipse, and BSpline; and each surface is one of the following types: Plane, Cylinder, Cone, Sphere, Torus, Revolution, Extrusion, BSpline.

Each region (a region is the submesh corresponding to one parametric surface) is exactly representable by a (possibly trimmed) NURBS patch. So we hoped that by leveraging this geometric criteria we could achieve a low-error Bézier patch fitting using the adaptive fitting algorithm. Further, because we exactly know the parametric form of the regions, it may be possible to deterministically segment each region into quads in the start, thus easing the fitting process. Lastly, one hidden problem with the adaptive fitting algorithm is that solving the least squares objective corresponding to the bi-quintic Bézier patch requires a mesh parametrization, i.e. assigning (u,v) values to each mesh vertex. This problem is solved by the ABC dataset as it provides parameter values at each mesh vertex.

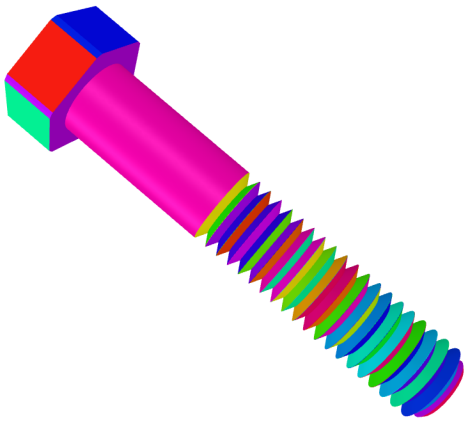

Figure 10: A visualization of the patch decomposition provided in the ABC dataset

However, a setback here was that many regions were incomplete; for instance a region whose type was cone could be truncated, and one whose type was plane could be any arbitrary shape as long as all points lied on a single plane. This is natural as cutting, extrusion, and boolean operations are commonly used in CAD. But this meant that the deterministic segmentation idea was a non-trivial problem in itself.

4.3 Random perturbation for dataset generation

At this juncture, we decided to pause and pursue a different direction. We went back to the drawing board and decided to try and generate our own dataset by

systematically and randomly spawning control nets (equivalently control grids) subject to the continuity constraints. This approach does not need any fitting algorithms, but generating vast data which contains enough variations to be generalizable and robust is no mean feat. This is a work in progress, but in figure 11 we show some of the generated surfaces, where each surface refers to a bunch of bicubic Bézier patches stitched together.

The key idea was to start off with a flat and evenly spaced out control grid $\mathbf{P} = \{\mathbf{p}_{i,j}\}_{0 \le i,j \le m}$; with $m = 3$ if we are dealing with bicubic patches. We would then stochastically perform perturbation operations on the control points. The next control grid $\mathbf{Q} = \{\mathbf{q}_{i,j}\}_{0 \le i,j \le m}$ would have the property that $\mathbf{q}_{0,j} = \mathbf{p}_{m,j}$ for every $j \in \{0, 1, \ldots, m\}$. This would ensure C0 continuity. Further, $\mathbf{p}_{m-1,j}$, $\mathbf{p}_{m,j}$ ($= \mathbf{q}_{0,j}$), and $\mathbf{q}_{1,j}$ would be collinear. This would ensure G1 continuity. We can have C1 continuity by demanding $\mathbf{p}_{m,j}$ to be the midpoint of the line joining $\mathbf{p}_{m-1,j}$ and $\mathbf{q}_{1,j}$), but this is not always possible due to joins with multiple patches. This gives us surfaces. Next we sample strokes from each surface, where strokes possibly span multiple patches. This process gives us our dataset. In figure 11, We showcase a few surfaces generated by ut method.

Now we move on to thinking about part two of the problem.

5 The stroke-patch association problem

Let us revisit the problem statement. Our goal is to devise a method or an algorithm to predict whether a given stroke belongs to a given patch or not. We again review some literature on stroke classification.

Zhu et al. in [19] consider the problem of assigning labels to strokes. Each sketch belongs to one of a set of pre-determined categories (for example lamp, chair, etc) and each category has a pre-decided set of labels. A major obstacle is the dearth of training data for deep learning models. Each category in Huang dataset [20] has 30 sketch samples, where each sketch has not more than 30 strokes. Transfer learning on CNNs helps alleviate this problem. A VGGNet16 [19] pre-trained on the image data is fine-tuned on the sketch data. Since VGGNet16 works on 3-channel images, a novel input representation is called for. The proposal here is to rasterize the sketch and the stroke together, placing the former in the first channel and the latter in the second. The third channel is fed with zeros. The classifier module is replaced to suit this classification problem, and a sequence of grouped convolutional layers is added just before the classifier. This is shown in figure 12. A few sample results can be seen in figure 13.

We started thinking of lifting this idea of transfer learning to our 3D setting by fine tuning a network like DGCNN. The inputs would now be a patch and a stroke. A stroke is nothing but an ordered point cloud, and a patch, which is stored as a parametric surface, can also be converted to a point cloud by sampling. A novel

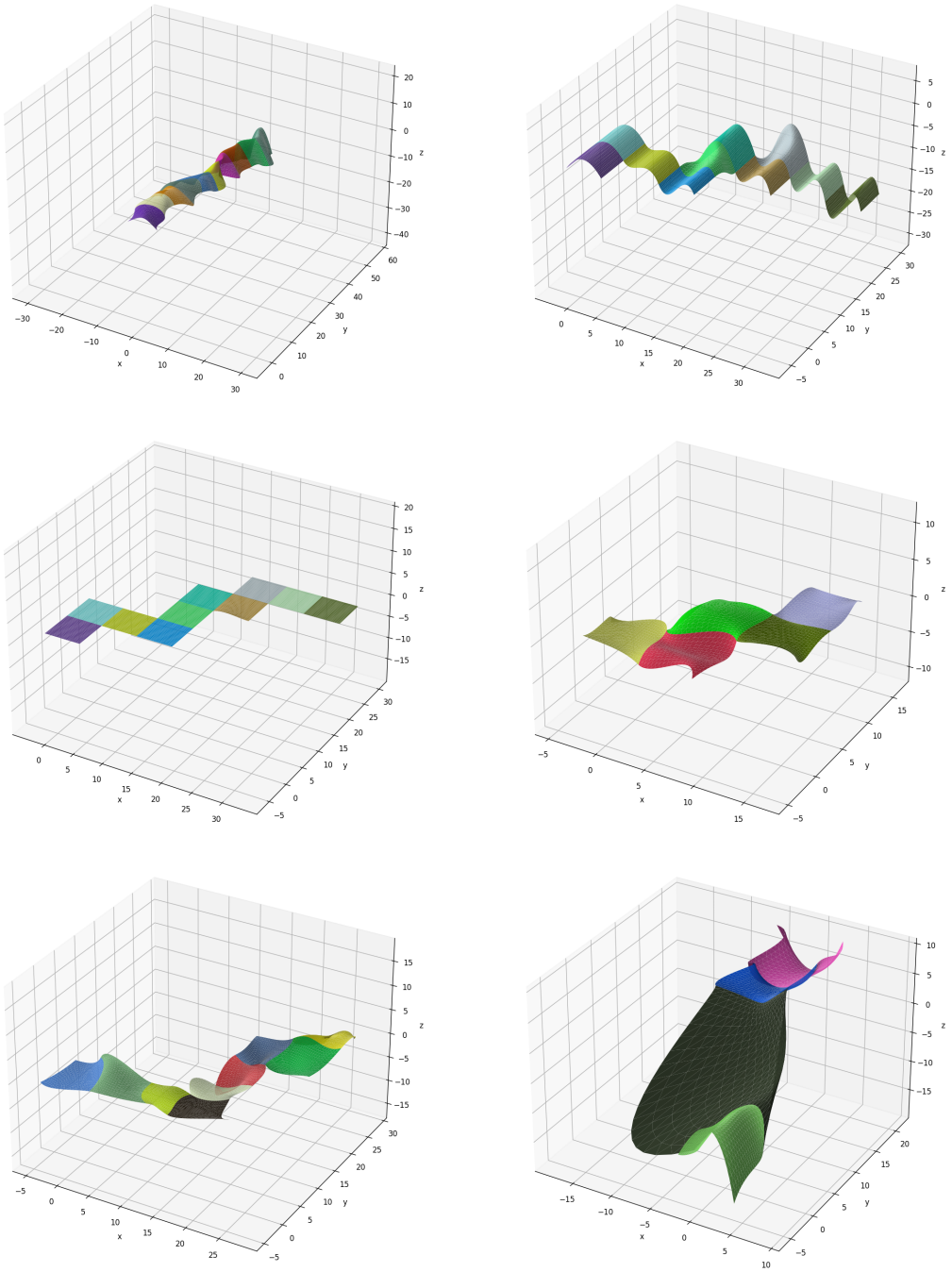

Figure 11: A few of the surfaces generated using random perturbation

input representation would be needed here too to feed the patch to the network.

However, on further thought, it seemed as if the problem was not as much of

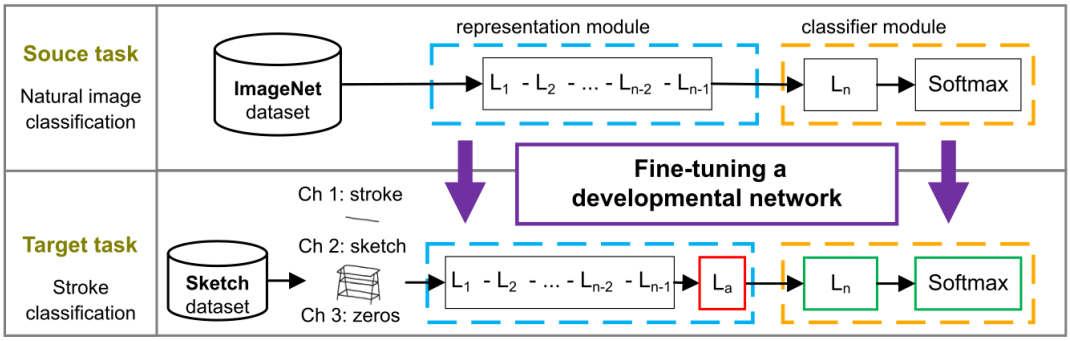

Figure 12: Transfer learning for stroke classification

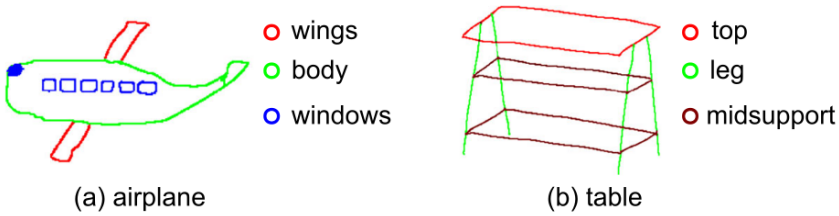

Figure 13: Classifying sketch strokes; note that the labels depend on the category

a high-level task as classification of strokes in a sketch. In 3D, distance cues can provide an easy way to associate strokes to patches. So we propose an algorithm to perform stroke-patch association.

A space-partitioning grid structure like an octree [21] would be maintained which keeps track of which patches occupy what grid points. The patches would be augmented with a buffer zone enveloping them. By querying the grid structure, we can efficiently do voting on patches, i.e. figure out which patches' regions the strokes lies in and how much. This output could be handled by the module which deals with the first part of the problem and it could decide which action to take; whether to extend the ongoing patch, or to start a new patch joined to the previous patch, or to go and update an older patch. This is still an idea in development and we have not been able to run any experiments to see how it performs.

6 Conclusions and Reflections

So there is much the future holds in store. The dataset generation is the part which has received most attention and work so far, but it needs a few refinements and additions. We aim to build a Desktop application which will take user-drawn sketches from the screen and generate, in real-time, a suitable 3D model. Building

the front-end for the application is developmental task yet to be started. The main generative network is functional but needs some improvement. The segmentation module (stroke-patch associator) has been ideated but not experimented with. On a tangential note, fitting parametric surfaces to meshes is an interesting problem we might consider working upon. Indeed, there is much to look forward to.

This four month period has been very instructive and eventful. It was often challenging, managing the core coursework parallelly, and sometimes there were weeks on end when we would be stuck at the same place (literally as well!). But it would then be satisfying when understanding dawned. We also had to abandon some trails a few times. This was painful, but probably part and parcel of research. The meetings were fun and often imparted motivation and insight. All in all, it was quite an experience and I am glad to have gone through it.

Acknowledgements

I am extremely grateful to Prof Parag Chaudhuri and Sukanya Bhattacharjee, a PhD student here in the CSE department. The “we” in this report really refers to all three of us, and this has been a wholly collaborative effort in these trying times. I am also grateful to the CSE department for providing an opportunity to students in the form of a credited course to undertake a research project.

References

[1] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, 2018. URL: `http://www.blender.org`.

[2] Autodesk, INC. Maya, 2019. URL: `https://autodesk.com/maya`.

[3] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, page 151–160, New York, NY, USA, 2008. Association for Computing Machinery. `doi:10.1145/1449715.1449740`.

[4] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: Designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26(3):41–es, July 2007. `doi:10.1145/1276377.1276429`.

[5] Alexis Andre and Suguru Saito. Single-view sketch based modeling. page 133–140, New York, NY, USA, 2011. Association for Computing Machinery. `doi:10.1145/2021164.2021189`.

[6] Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. Graph.*, 33(4), July 2014. doi:10.1145/2601097.2601128.

[7] Luke Olsen, Faramarz Samavati, and Joaquim Jorge. Naturasketch: Modeling from images and natural sketches. *IEEE Computer Graphics and Applications*, 31(6):24–34, 2011. doi:10.1109/MCG.2011.84.

[8] Yongxin Yang and Timothy M. Hospedales. Deep neural networks for sketch recognition. *CoRR*, abs/1501.07873, 2015. URL: http://arxiv.org/abs/1501.07873, arXiv:1501.07873.

[9] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 409–416, USA, 1999. ACM Press/Addison-Wesley Publishing Co. doi:10.1145/311535.311602.

[10] Xiaoguang Han, Chang Gao, and Yizhou Yu. Deepsketch2face: A deep learning based sketching system for 3d face and caricature modeling. 36(4), July 2017. doi:10.1145/3072959.3073629.

[11] Poly, 2021. URL: https://poly.google.com/.

[12] Google Inc. Tilt brush, 2021. URL: https://www.tiltbrush.com/.

[13] Gravity sketch, 2021. URL: https://www.gravitysketch.com/.

[14] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2019. arXiv:1801.07829.

[15] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017. arXiv:1612.00593.

[16] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2015. arXiv:1406.5670.

[17] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[18] Hongwei Lin, Wei Chen, and Hujun Bao. Adaptive patch-based mesh fitting for reverse engineering. *Comput. Aided Des.*, 39(12):1134–1142, December 2007. doi:10.1016/j.cad.2007.10.002.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. arXiv:1409.1556.

[20] Zhe Huang, Hongbo Fu, and Rynson W. H. Lau. Data-driven segmentation and labeling of freehand sketches. *ACM Trans. Graph.*, 33(6), November 2014. doi:10.1145/2661229.2661280.

[21] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982. URL: https://www.sciencedirect.com/science/article/pii/0146664X82901046, doi:https://doi.org/10.1016/0146-664X(82)90104-6.