For Computer- Aided Design, a Deep Engineering Sketch Generative Network

Dhiraj V. Astonkar
Assistant Professor, Department of Mechanical Engineering
Dr. Sau. Kamaltai Gawai Institute of Engineering & Technology, Darapur, Tq. Daryapur, Dist. Amravati, Maharashtra, India
Email id: dhirajastonkar@gmail.com

Abstract. Deep generative models of 3D shapes have sparked a lot of attention in the scientific community. Nonetheless, they almost all produce discrete shape representations like voxels, point clouds, and polygon meshes. We provide the first 3D generative model for a fundamentally different shape representation: expressing a shape as a series of CAD procedures. CAD models, unlike meshes and point clouds, encode the user creation process of 3D shapes, which are widely utilised in industrial and engineering design. Existing 3D generative models, on the other hand, have considerable hurdles due to the sequential and irregular structure of CAD procedures. We propose a CAD generative network built on the Transformer, based on an analogy between CAD processes and spoken language. To encourage future research on this topic, we have made this dataset freely available. The 2D foundation of parametric Computer-Aided Design (CAD), the most common modelling paradigm for manufactured items, is engineering sketches. In this study, we look at the challenge of learning-based a deep engineering sketch creation as a first step toward parametric CAD model synthesis and composition.

1. Introduction

It is in our nature to create and invent, and to express our creations in three-dimensional forms. This is why drawing tools like the parallel bar, the French curve, and the divider were invented; and this is why, in today's digital era, computer aided design (CAD) software has been used for 3D shape creation in a wide range of industrial sectors, from automotive and aerospace to manufacturing and architectural design [1]. Is it possible for the machine to create 3D shapes as well? Many recent research efforts have been dedicated toward the development of 3D models, using the remarkable advancement in generative models of deep learning [2]. 3D point clouds, polygon meshes, and level set fields are examples of existing 3D generative models that merely construct computer discretization of 3D objects [3]. The ability to generate the drawing process, which is at the heart of 3D shape design, is still missing. We present a deep generative network that generates a sequence of operations that may be utilised to build a 3D shape in CAD software (such as SolidWorks and AutoCAD)

[5]. Such an operational sequence, commonly referred to as a CAD model, describes the "drawing" process of shape development. CAD models are used in practically all industrial 3D designs today. They are discretized into polygon meshes or point clouds only, if necessary, later in the manufacturing workflow [8]. This is the first approach toward a generative model of CAD drawings that we are aware of. The sequential and parametric nature of CAD design presents a hurdle [12]. A CAD model is made up of a succession of geometric operations (for example, curve sketch, extrusion, fillet, Boolean, and chamfer) that are all controlled by parameters. There are discrete alternatives for some of the parameters, while others offer continuous values. These abnormalities arise from the user's construction of 3D shapes, and they stand in stark contrast to the discrete 3D representations (voxels, point clouds, and meshes) employed in existing generative models [6]. As a result, previously produced 3D generative models are unsuitable for the creation of CAD models. Contributions in terms of technology. To address these issues, we're looking for a representation that reconciles CAD model anomalies [7]. We take the most often used CAD operations (or commands) and group them into a common structure that encapsulates their command types, parameters, and sequence of execution. Following that, we present an autoencoder based on the Transformer network [10], which draws parallels between CAD command sequences and natural languages. It decodes a latent vector into a CAD command sequence after embedding CAD models in a latent space. We also build a new dataset of CAD command sequences to train our autoencoder, which is orders of magnitude larger than the existing dataset of the same type [11]. This dataset has also been made publicly available in order to encourage future research on learning-based CAD designs.

The most common 3D modelling paradigm is parametric computer-aided design (CAD), which is used to design manufactured products ranging from vehicle parts to electronic devices to furniture. This approach is supported by all major solid modelling kernels and is common to all parametric CAD tools. A variety of CAD workflows can benefit from engineering sketch creation. For example, the capacity to autonomously reverse engineer a parametric CAD model from noisy 3D scan data has long been a goal [1]. Auto-completion of user input can also be done with engineering sketch generating. When creating complex engineering designs, the ability to infer repeating commands based on visual or geometric input could greatly reduce user effort. Another in-demand feature is the capacity to generate engineering sketches using approximate geometry, such as free-hand drawings. A generative model for engineering sketches, often known as beautifying, has the potential to improve user workflows over existing approaches [9]. Despite recent advancements in data-driven 2D vector graphic synthesis [3], there is little study on directly generating engineering sketches. This is a difficult topic since engineering designs contain a variety of 2D geometric primitives as well as topological information about how they are related.

The following are the contributions made by this paper: • Without the use of a sketch constraint solver, we discuss two generative models that address the challenge of deep engineering sketch generation networks. • With our CurveGen model, we use a unique sketch representation that implicitly encodes the sketch primitive type based on hyperedge cardinality. • We present quantitative and qualitative engineering sketch generation results, as well as the findings of a perceptual research that compared our generative models to a state-of-the-art baseline.

2. Related work

Inference of a parametric shape. Deep learning advances have made it possible to create neural network models that analyse geometric data and infer parametric shapes. A 3D point cloud is decomposed into a series of parametric surface patches using ParSeNet. PIE-NET uses 3D point clouds to extract parametric boundary curves. UV-Net and BrepNet focus on encoding the boundary curves and surfaces of a parametric model. Used synthetic data to train a neural network to transform 2D user doodles into CAD processes. Recently used neural-guided search to deduce CAD modelling sequence from parametric solid shapes. 3D form generation models. In recent years, there has been a surge in research towards deep generative models for 3D shapes. Most known approaches, such as voxelized shapes, point clouds, polygon meshes, and implicit signed distance fields [12], generate 3D objects in discrete forms. The generated shapes may have noise, lack precise geometric characteristics, and aren't directly changeable by the user. As a result, more recent research has aimed to develop neural network models that construct 3D shape through a sequence of geometric operations. CSGNet infers a sequence of Constructive Solid Geometry (CSG) operations using voxelized shape input, and UCSG-Net pushes the inference forward without the help of ground truth CSG trees. Other than CSG procedures, some research offers domain specific languages (DSLs) to synthesise 3D shapes. Shape Assembly, a DSL that generates 3D structures by hierarchically and symmetrically constructing cuboid proxies, and this structure can be created using a variational autoencoder. In contrast to all of these previous studies, our autoencoder network produces CAD models that are specified as a series of CAD operations. In practically every field of industrial production, CAD models have become the standard shape representation. As a result, our network's output may be easily imported into any CAD software for user editing. Other shape formats, such as point clouds and polygon meshes, can be directly translated. This is the first generative model that produces CAD designs directly, as far as we know. Models based on transformers. The Transformer network, which was introduced as an attention-based building block for several natural language processing applications, is a technical relation of our work. The Trans former network's success has led to its application in image processing and other types of data. Transformer network is also used in concurrent works on limited CAD sketch generation. DeepSVG, a Transformer-based network for the creation of Scalable Vector Graphic (SVG) pictures, is also connected to our work. A set of parametric primitives is used to describe SVG images (such as lines and curves). Aside from being restricted to 2D, those primitives are organised in no particular sequence or reliance. CAD commands, on the other hand, are expressed in 3D, can be interdependent (e.g., via CSG boolean operations), and must be executed in a specific order. As a result, we're looking for a novel technique to encode CAD commands and their order in a Transformer-based autoencoder.

Graphics in Vector Format In commercial software, vector graphics are widely utilised to allow for the resolution-independent design of fonts, logos, animations, and images. Generative adversarial network to create smooth curves and applies it to 2D airfoil profiles. Vector graphic generating system that uses differentiable rendering and is trained only on raster pictures. The usage of Bézier curves has been common in previous research. Engineering sketches prefer line, arc, and circle primitives over NURBS surfaces Layout and technical drawings technical drawings are two-dimensional projections of three-dimensional computer-aided design (CAD) models, with key information denoted by dimensions, notes, or section views. The dataset is primarily intended for challenges involving spatial thinking. Their approach enables designers to create freehand sketches that resemble engineering sketches used in the creation of a 3D model. They use a Trans former-based network to anticipate the parameters of lines and Bézier curves, then refine them via optimization. Our technology, on the other hand, concentrates on the creation of new sketch geometry that may be used with 3D CAD modelling procedures.

3. Generative Models

We build and analyse two different neural networks for engineering sketch generation: Curve Gen and TurtleGen.

3.1.CurveGen

CurveGen is an engineering sketch generating application of the PolyGen architecture. CurveGen automatically generates the sketch hypergraph representation. We separate the creation of G into two parts based on the chain rule, as we did with the original PolyGen implementation: 1) create the sketch vertices V, and 2) create the sketch hy peredges E based on the vertices.

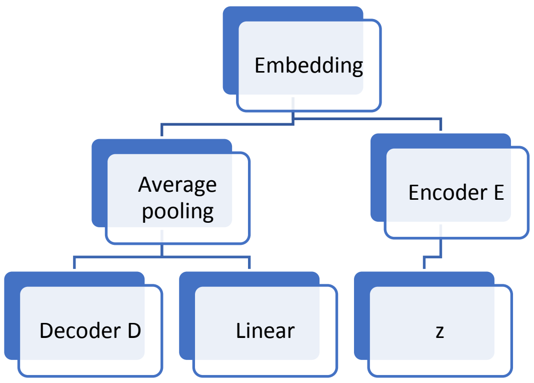


Figure 1. The structure of our network. The input CAD model, which is represented as a command sequence, is projected to an embedding space before being fed to the encoder E, resulting in a latent vector z. The decoder D receives as input learned constant embeddings as well as the latent vector z. The projected command sequence is then output.

p(.) stands for probability distributions. Figure 1 depicts the two-step generation process, which begins with the vertex model (left) and ends with the curve model (right). With 2D vertex coordinates, we use the PolyGen vertex model directly. Negative log likelihood loss is used to train both the vertex and curve models against the ground truth data.

3.2.TurtleGen

A sequence generator is a neural network that generates a programme in the Turtle representation. We randomise the turtle sequence creation for each given sketch in the hypergraph format by selecting the loop order, loop beginning vertex, and loop drawing direction at random. Long sequences of more than 100 turtle instructions are discarded. The network comprises seven input and output linear branches, with the corresponding command other corresponding coordinates. Zeros are used in commands that are less than three. At the previous sequence phase, the output branches are coupled to the Transformer encoding. The sketch hypergraph representation is then recovered, as well as the geometry and topology, for each sampled sequence.

4. Network-friendly Representation

Our CAD model M specification is written in natural language. Individual CAD commands are expressed sequentially to build phrases in the language. The sketch profile is the subject of a sentence, and the extrusion is the predicate. This parallel suggests that we could use network structures that have been successful in natural language processing, such as the Transformer network, to achieve our goal. CAD commands, on the other hand, differ from normal language in various ways. The number of parameters for each command varies. The parameters in some commands (e.g., extrusion) are a combination of continuous and discrete values, and the parameter values span distinct ranges. These characteristics make the command sequences unsuitable for use in neural networks. The dimensions of command sequences are regularised to overcome this problem. First, the parameters of each command are piled into a 161 vector, the elements of which correspond to the collective parameters. Each command's unused parameters are simply set to 1. The total number of commands Nc in each CAD model M is then fixed. This is accomplished by padding the command sequence of the CAD model with the empty command hEOSi until the length of the sequence approaches Nc. In fact, we use Nc = 60 because that is the longest command sequence length we found in our training dataset. We also quantize the continuous parameters to unify continuous and discrete parameters. Every CAD model is normalised within a cube, and every sketch profile is normalised within its bounding box, with a scale factor s (in the extrusion command) to restore the normalised profile to its original size. The normalisation limits the ranges of continuous parameters, allowing us to quantize them into 256 levels and express them with 8-bit integers. As a result, only discrete sets of values exist for all command parameters. The parameter quantization is not just a continuation of the standard technique for training Transformer-based networks. It is critical, especially for CAD models, to improve the generating quality. Certain geometric relationships, such as parallel and perpendicular sketch lines, must be respected in CAD drawings. However, if a generative model generates continuous parameters directly, the values obtained by parameter regression are prone to errors, which might cause these stringent relationships to be broken. Parameter quantization, on the other hand, allows the network to "classify" parameters into specified levels, allowing it to better respect learnt geometric relationships. We'll show ablation studies that empirically support our CAD command representation choices.

5. Autoencoder for CAD Models

We'll now talk about an autoencoder network that uses our CAD command representation. The decoder element of the network will naturally act as a CAD generative model once it has been trained. The Transformer network is the basis for our autoencoder, which was inspired by its performance in processing sequential data. Our autoencoder takes as input a CAD command sequence $M = [C_1, \cdot\cdot\cdot, C_{N_C}]$, where N_C is a fixed number. First, each command C_i is projected separately onto a continuous embedding space of dimension dE = 243.

5.1.**Encoder**.

Each of the four layers of Transformer blocks in our encoder E has eight attention heads and a feed-forward dimension of 500. The encoder accepts the embedding sequence as input and produces vectors with the same dE = 256 dimension. Finally, the output vectors are averaged to get a single dE dimensional latent vector z.

5.2.Decoder.

Our decoder D, which is also made of Transformer blocks, has the same hyper-parameter values as the encoder. It uses learnt constant embeddings as input while also taking into account the latent vector z—a similar input structure was utilised. The output from the final Transformer block is input into a linear layer, which predicts a CAD command sequence $M = [C_1, \cdot\cdot\cdot, C_{N_C}]$, which includes both the command type t_i and the command parameters p_i for each command. We apply the feed-forward method instead of the autoregressive strategy typically used in natural language processing.

6. Results

To provide quantitative and qualitative findings on the job of engineering sketch development, we compare the CurveGen and TurtleGen generating models to the SketchGraphs generative model.

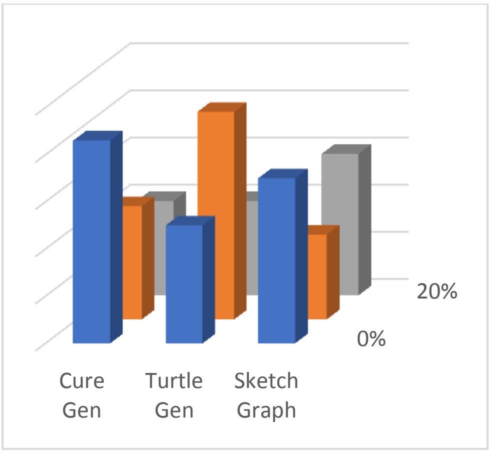
6.1.Quantitative Results

We conducted a perceptual evaluation utilising human volunteers to see how deep engineering sketches generated by each model compare to human-designed sketches. Each participant is shown with one human-designed and one computer-generated deep engineering sketch and asked, "Which sketch is more realistic?" in our two-alternative forced choice study.

Table 1. The findings of quantitative sketch generation. The negative log-likelihood over the test set is calculated as bits per sketch. Over 1000 created sketches, Unique, Valid, and Novel are calculated.

Model	Bits per sketch	Unique %	Valid %	Novel %
Curve Gen	31.09	98	82	92
Turtle Gen	55.06	85	44	84
Sketch Graphs	98.87	77	67	73
Sketch Graphs (Duplicate)	95.06	61	72	50

Brief instructions are included, as well as an illustration of a deep engineering sketch in context. The above table 1 shows the outcomes of quantitative sketch generation. The negative log-likelihood determined across the test set is referred to as Bits per Sketch. Over 1000 created sketches, the terms "unique," "valid," and "novel" are calculated. We record the responses of three human volunteers for each pair of sketches and utilise the majority answer.



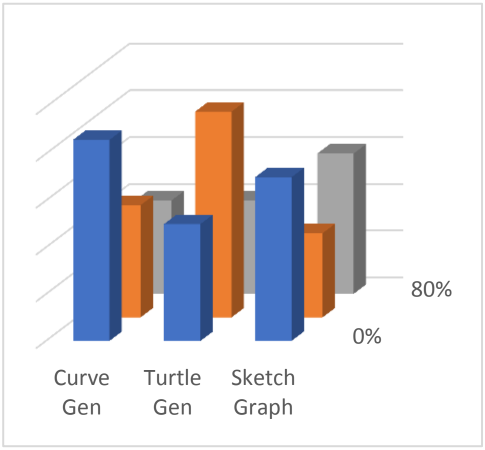


Figure 2: The results of our perceptual study with human subjects to determine which engineering sketch is the most realistic. Left: The percentage of generated sketches that are considered more realistic than sketches created by humans. Right: The inter-rater agreement percentage.

The research was carried out with the help of Amazon Mechanical Turk workers. The percentage of generated sketches classified as more realistic than human-designed sketches is shown in Figure 2 on the left; greater numbers are better. A 50% rating suggests that the generated sketches are indistinguishable from human-designed designs. The inter-rater agreement between each of the three human subjects is shown in Figure 4 to the right as a percentage. A lower score suggests that the produced and human-designed sketches are more confused. CurveGen output is the most realistic of the generated engineering designs, according to the study's findings.

6.2.Qualitative Results

Curve Gen, in instance, is able to make sketches with closed loops, symmetrical features, perpendicular lines, and parallel lines on a constant basis, we present further qualitative results. CAD Models from Sketch to Solid The current study is motivated by the desire to enable the synthesis and composition of solid CAD models.

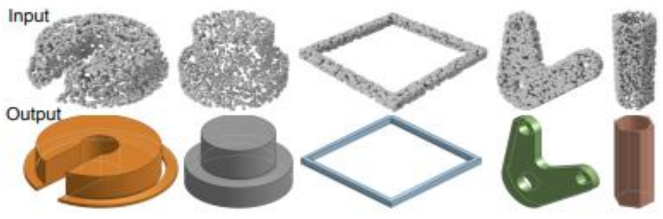


Figure 3. The restoration of a CAD model from point clouds. input point clouds (top). CAD models that have been recreated (bottom)

The above figure 3 shows how engineering sketches created by CurveGen with closed loop profiles can be procedurally hoisted into 3D using the extrude modelling process. Constraints Diagram Our generative network models' geometric output can be post-processed to apply sketch constraints and create a constraint graph.

7. Conclusion and Discussion

There are various limitations in our approach to the CAD generative model. We've looked at the three most common forms of curve instructions so far (line, arc, and circle), but other curve commands can easily be added. A cubic Bézier curve, for example, can be defined by three control points plus the starting point from the previous curve's ending position. These variables can also be organised in the same way. Other operations, such spinning a sketch, can be represented in the same way as the extrusion instruction. Certain CAD operations, such as fillet, work on areas of the form boundary, necessitating a reference to the model's B-rep rather than just other instructions. It will be left to future study to include those directives into the generative model. Not every CAD command sequence will result in a topologically correct shape. Our generative network can't ensure that the output CAD sequences are topologically sound. The produced CAD command sequence seldom fails in practise. As the command sequence grows longer, the chances of failure increase. Also analyse certain failure scenarios, providing some material for future research. In conclusion, DeepCAD, a deep generative model for CAD designs, has been demonstrated. Almost all prior 3D generative models generated distinct 3D shapes like voxels, point clouds, and meshes. This is the first generative model for CAD drawings that we are aware of. We also present a large dataset of CAD models, each of which is represented as a CAD command sequence.

References

[1] Francesco Buonamici, Monica Carfagni, Rocco Furferi, Lapo Governi, Alessandro Lapini, and Yary Volpe. Reverse engineering modeling methods and tools: a survey. Computer-Aided Design and Applications, 15(3):443–464, 2018.

[2] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. In Advances in Neural Information Processing Systems (NeurIPS), 2020.

[3] Wei Chen, Kevin Chiu, and Mark D Fuge. Airfoil Design Parameterization and Optimization Using Bézier Generative Adversarial Networks. AIAA Journal, 58(11):4723–4735, 2020.

[4] Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. Deep vectorization of technical drawings. In European Conference on Computer Vision, pages 582–598. Springer, 2020.

[5] Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl. In Advances in Neural Information Processing Systems (NeurIPS), pages 9169– 9178, 2019.

[6] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B Tenenbaum. Learning to infer graphics programs from hand-drawn images. arXiv preprint arXiv:1707.09627, 2017.

[7] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake sleep bayesian program learning. arXiv:2006.08381, 2020.

[8] Jun Gao, Chengcheng Tang, Vignesh Ganapathi Subramanian, Jiahui Huang, Hao Su, and Leonidas J Guibas. Deepspline: Data-driven reconstruction of parametric curves and surfaces. arXiv:1901.03781, 2019.

[9] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. International Conference on Learning Representations (ICLR), 2019.

[10] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floor plan generation from layout graphs. ACM Transactions on Graphics (TOG), 39(4):118–1, 2020.

[11] Kacper Kania, Maciej Zi˛eba, and Tomasz Kajdanowicz. Ucsg-net–unsupervised discovering of constructive solid geometry tree. arXiv:2006.09102, 2020.

[12] Hsin-Ying Lee, Weilong Yang, Lu Jiang, Madison Le, Irfan Essa, Haifeng Gong, and Ming-Hsuan Yang. Neural design network: Graphic layout generation with constraints. European Conference on Computer Vision (ECCV), 2020.

[13] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. In International Conference on Learning Representations (ICLR), 2019.

[14] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2652–2660, 2019.

[15] Tzu-Mao Li, Michal Lukác, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. ACM Transactions on Graphics (TOG), 39(6):1–15, 2020.

[16] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 7930–7939, 2019.