Deep generative models for engineering design

Fan, J.

Citation

Fan, J. (2026, March 24). *Deep generative models for engineering design*. Retrieved from https://hdl.handle.net/1887/4298630

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4298630

Note: To cite this publication please use the final published version (if applicable).

Chapter 5

Learning Efficient Representations for 3D-Surfaces

Despite the advancements made in generating plausible engineering designs in the both forms of cross-section blueprints or meshes, utilizing natural CAD data remains our ultimate objective. However, as we initiated the study, we found that both the data representation and the generative capabilities of DGMs had not yet sufficiently advanced to address the generation of 3D B-Reps. With the growing interest and effort in this field, current advancements appear promising, as demonstrated in Section 1.4. Although these most recent works [66, 187] have already enabled the direct generation of B-Rep solids, they rely on UV-grids to represent the geometry of each surface, which is less optimal compared to the natural surface form, NURBS. Therefore, in this chapter, we propose NeuroNURBS, which addresses the research question 4: *How to enable DGMs to directly synthesize CAD native representation?* The content of this chapter has been published in paper [166].

5.1 Introduction

Boundary Representation (B-Rep) [177, 87] is commonly used to represent shapes and solids in computer-aided design (CAD) — widely applied in industrial design, simulation, and manufacturing. In a B-Rep, the solid boundaries are defined using a set of

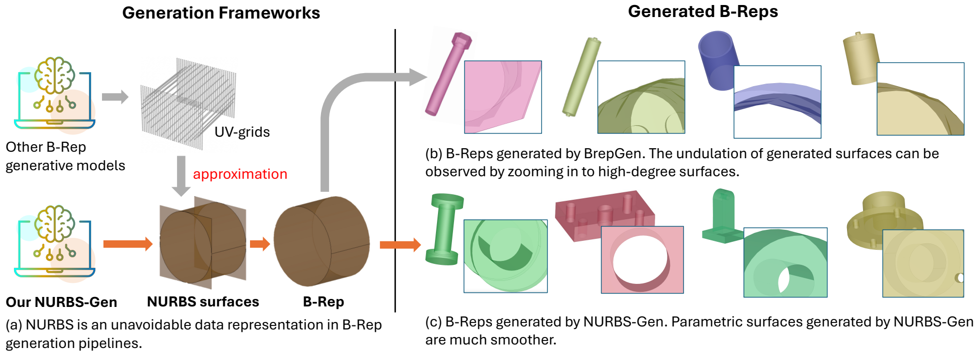


Figure 5.1: Why do we need UV-grids, if we could directly learn on NURBS. (a) Comparison of various generation frameworks. (b) Visualization of generated results. Our *NURBS-Gen* can directly generate valid NURBS parametrization and hence ensure the smoothness and regularity of the surfaces, in contrast to the BrepGen surfaces, which appear undulating.

surfaces [177, 39], which are, by default, parameterized by Non-Uniform Rational B-Splines (NURBS) [128]. Deep learning could offer solutions to several computational tasks important to the industry, e.g., B-Rep generation and solid segmentation. Aiming to solve these tasks, various works have been devoted to learning the geometry and topology of B-Rep data [67, 36, 66, 179, 187]. A major work, BrepGen [187], decomposes the B-Rep entities (faces, edges, vertices) into a tree data structure that encodes the topological information.

Despite the success of learning the topology of B-Rep, we find that these works [187, 67, 101] utilize a parametric approximation to surfaces, that is representing the surface with a certain number of 3D points uniformly distributed in the UV-domain, i.e., a UV-grid [67]. More details about UV-grids are introduced in Section 5.2. However, this approach has several drawbacks: (1) the approximation to the target surface is often imprecise unless a dense UV-grid is used to achieve an acceptable error range [67, 111, 109]. (2) The demand for high accuracy often incurs large data sizes, model sizes, and computational costs, which is sometimes unnecessary. For example, planar surfaces are represented by 32×32 3D points in [187]. (3) Generative models that use UV-grid-based surface approximation, e.g., BrepGen, can produce artificial undulating patterns on the surface. For instance, we showcase several solids generated by BrepGen in Figure 5.1 (a), where some sections of the curvy surface are not perfectly smooth.

Motivation Alternatively, it is more natural and advantageous to use the NURBS parametrization in the solids learning task: NURBS are more accurate [128, 109, 39], easier to manipulate [127, 63], and have less parameters [128, 44] compared to UV-grids. However, incorporating the parameters of a NURBS surface in deep neural networks is not trivial: (1) The parameters — control points, knot vectors, and weight matrix (see explanation in Section 5.2) — have varying sizes across surfaces in a B-Rep data set. It is challenging to unify them into a fixed-size input to a deep neural network. (2) These parameters are related (e.g., control points are only meaningful together with knot vectors and and the dimension of the weight matrix depends on the number of control points), and hence, they should be encoded and decoded jointly if we wish to learn a shared latent representation thereof.

Contribution We target learning *an effective representation of NURBS parameters*, for which we design a pipeline that preprocesses raw B-Rep solids into learnable NURBS parameters and propose the *NeuroNURBS model* to autoencode the heterogeneous sizes of NURBS parameters. We refer to the resulting latent representation as *NURBS features*. More precisely, *NeuroNURBS* is our proposed pipeline to convert *NURBS parameters* into *NURBS features* that can be used in downstream tasks, e.g., solid segmentation and generation. In Figure 5.1 (b), we illustrate some examples of surfaces generated with NURBS-Gen, which is obtained by replacing the preprocessing and autoencoder of surface entities with NeuroNURBS in the BrepGen [187] framework. As seen from Figure 5.1, the surfaces generated with NeuroNURBS and NURBS features are very smooth and regular. To evaluate NeuroNURBS, we conduct the following experiments:

1. We compare its performance with the UV-grid method for reconstructing surfaces on DeepCAD [182] in terms of accuracy, memory efficiency, and computation cost (Section 5.4.2). We see the memory required to store features of solids is reduced by **79.9%**, the surface autoencoder's size is reduced by **92.9%**, the GPU consumption for training the surface autoencoder is reduced by **86.7%**. We further test on the ABC [76] dataset, where our method NeuroNURBS shows a near-perfect surface reconstruction.

2. We test the NeuroNURBS on two downstream tasks: B-Rep generation on DeepCAD [182] and ABC [76] (Section 5.4.3) and solid segmentation on MFCAD [23] (Section 5.4.4). For the former, our NURBS-Gen improves the FID (Fréchet Inception Distance [58]) from 30.04 (achieved by BrepGen) to **27.24** on DeepCAD

dataset. For the latter, using NeuroNURBS achieves an accuracy of **99.65%**.

5.2 Preliminaries

UV-grids A 3D freeform surface $\boldsymbol{S}$ can be approximated by a parameterized function $\mathbf{F}(u, v)\colon \mathbb{R}^2 \to \mathbb{R}^3$, which is learned on a number of points evenly distributed in $[u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] \in \mathbb{R}^2$. $\mathbf{F}(u, v)$ is called UV-grid function (see the left side of Figure 5.2). To learn the grid function, one can sample $n \times m$ points (along the U and V dimensions, respectively) from the target surface. A large sample number can increase the approximation accuracy of the original surface but this comes at the cost of higher-dimensional representation, leading to increased training times and memory requirements.

Non-Uniform Rational B-Splines Non-Uniform Rational B-Splines (NURBS) are essentially B-splines applied in homogeneous coordinates, rather than 3D coordinates, i.e., $(x, y, z, w) \mapsto (x/w, y/w, z/w)$. Each NURBS of order n (polynomial of degree $n-1$) requires n control points and $2(n-1)$ knots. A NURBS curve $C(u)$ can be defined using coordinate u with the formula:

$$C(u) = \frac{\sum_{i=1}^{n} N_i^p(u) w_i p_i}{\sum_{i=1}^{n} N_i^p(u) w_i}\ , \tag{5.1}$$

where p_is are 3D control points and N_i^ps are basis functions of degree $p \leq n-1$, defined recursively on a knot vector $(u_1, \ldots, u_i, \ldots, u_{n+p+1})$:

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}(u) \tag{5.2}$$

$$N_i^0(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1}\ , \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

A NURBS surface $\boldsymbol{S}(u, v)\colon \mathbb{R}^2 \to \mathbb{R}^3$ is the tensor product of two NURBS curves, $C_U(u)$ (on n control points with order p) and $C_V(v)$ (on m control points with order q):

$$\boldsymbol{S}(u, v) \coloneqq C_U(u) C_V(v) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} N_i^p(u) N_j^q(v) w_{ij} p_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{m} N_i^p(u) N_j^q(v) w_{ij}},$$

which takes the following parameters: a grid of control points $p \in \mathbb{R}^{n \times m \times 3}$, weights $\boldsymbol{W} \in \mathbb{R}^{n \times m}$, the U-direction knot vector $\boldsymbol{U} = (u_1, \ldots, u_{n+p+1}) \in \mathbb{R}^{n+p+1}$, and the

V-direction knot vector $\boldsymbol{V} = (v_1, \ldots, v_{n+q+1}) \in \mathbb{R}^{m+q+1}$.

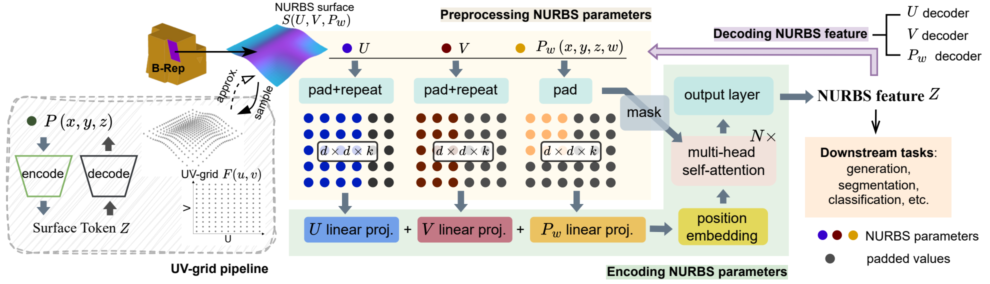


Figure 5.2: Diagram for NeuroNURBS. *Right:* two parts of NeuroNURBS, preprocessing and autoencoding NURBS parameters. *Left:* a simplified diagram for UV-grid, where the approximation from UV-grid back to NURBS surface is not deterministic.

5.3 Method

UV-grids enable the direct learning on B-Rep solids, but a UV-grid still serves as an approximation to the source surface, whereas using NURBS parametrization is a more natural and advantageous choice: NURBS are more accurate [128, 109, 39] and have less parameters [128, 44]. However, using modern neural networks to operate NURBS parameters is understudied and remains challenging (see Section 5.3). To address this, in this section, we describe NeuroNURBS which consists of a preprocessing pipeline for NURBS parameters and an autoencoder that is able to encode NURBS parameters into a low-dimensional feature space for downstream application.

To represent a surface, we propose to take its NURBS parametrization directly, which can be extracted from the B-Rep model, e.g., with `OpenCascade` functionalities. Compared to the UV-grid approach, the NURBS parametrization is (1) smooth and describes the surface precisely; (2) it can be memory efficient since it only requires storing a collection of 3D control points, a weight matrix, and two knot vectors. However, developing a deep learning model that takes as input the parameter of NURBS is challenging:

- NURBS parameters have different dimensions, e.g., a knot vector is a 1D object and control points are 3D objects and vary in size from one surface to the other.
- for a generative task, NURBS parameters have to be combined in the encoder and the decoder must map each latent point to a set of valid parameters.

We propose NeuroNURBS to resolve these challenges. It consists of two components: a preprocessing pipeline for the NURBS parameters (see below) and an autoencoder to learn an effective representation of NURBS parameters. We depict our method in Figure 5.2.

Table 5.1: Performance comparison of UV-grids and NURBS in surface representation tasks. All are evaluated on the DeepCAD dataset, except for the surface reconstruction, which is evaluated on the DeepCAD *(D)* and ABC *(A)*. We find that the reconstruction on NURBS data can achieve comparable accuracy as the one on UV-grids while offering significant advantages in representation efficiency.

	Input data size		Autoencoder size		Surface reconstruction		Construction speed
	Surface (MB/10k)	Solid (GB/10k)	VAE #Param.	GPU (GB)	CD ($\times 10^{-5}$) ↓	EMD ($\times 10^{-3}$) ↓	speed (NURBS/s)↑
UV-grids	245.8	37.7	84M	17.61	4.47 *(D)*, 3.20 *(A)*	1.90 *(D)*, 1.52 *(A)*	230
NURBS	**8.16** (**96.7%** ↓)	**7.6** (**79.9%**↓)	**6M** (**92.9%**↓)	**2.35** (**86.7%**↓)	4.23 *(D)*, 3.04 *(A)*	1.83 *(D)*, 1.67 *(A)*	**3230** (**92.9%**↑)

5.3.1 Preprocess NURBS Parameters

We consider the **control points** p, **weights** $\boldsymbol{W}$, and **knot vectors** $\boldsymbol{U}$ and $\boldsymbol{V}$ for the learning task, which we call the NURBS parameters. We exclude degrees p (degree in the U-direction) and q (in the V-direction) from parameters since they can be easily calculated: p is the length of knot vector $\boldsymbol{U}$ minus $n+1$, where n is the number of control points in the U direction. The degree q can be determined in the same way.

Normalization For control points $p_{ij} = (x_{ij}, y_{ij}, z_{ij})$, we determine the coordinate axis with the largest range among three coordinates:

$$d = \max\left(x_{\max} - x_{\min}, y_{\max} - y_{\min}, z_{\max} - z_{\min}\right), \tag{5.4}$$

and then normalize the coordinates as $\hat{x}_{ij} = (x_{ij} - x_{\min})/d$ (same for y-axis and z-axis). The normalization ensures $(\hat{x}_{ij}, \hat{y}_{ij}, \hat{z}_{ij}) \in [0,1]$. Note that weights w_{ij} also take values in the unit interval. Hence, we concatenate p and $\boldsymbol{W}$ together: $\boldsymbol{P_w} = [(\hat{x}_{ij}, \hat{y}_{ij}, \hat{z}_{ij}, w_{ij})]_{i\in[1..n], j\in[1..m]} \in \mathbb{R}^{n\times m\times 4}$.

Padding With a padding value of 0, we augment $\boldsymbol{P_w}$ to a tensor of size (d, d, 4), where d is the largest number of controls points across all surfaces in a data set. We also pad the knot vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ to length k (the length of the longest knot vector in a data set). To combine the knot vector and control points, we duplicate the knot

vector to a tensor of shape (d, d, k). Also, we save the padding mask of $\boldsymbol{P_w}$ for the autoencoder. Although the padding seems to impair the efficiency of the NURBS representation, the padding parts are omitted when training the transformer-based autoencoder using the padding mask.

5.3.2 Learning NURBS Features with Autoencoder

We study recent works in multi-modal and multi-task autoencoding [113, 43, 8, 121] and design a VAE model to learn a latent representation of the NURBS parameters, which we call *NURBS features*.

Our autoencoder is inspired by MultiMAE [8]. We use separate dense layers for each preprocessed NURBS parameter (control points with weights $\boldsymbol{P_w}$, U-knot vector $\boldsymbol{U}$ and V-knot vector $\boldsymbol{V}$). The sum of the dense outputs is then processed with a sine-cosine position embedding and is then input into a transformer together with the control point padding mask. For the selection of a different transformer backbone from MultiMAE [8], we do not use the Vision Transformer (ViT) [78] as our data has a low dimension and does not require patch embedding, but the introduction of a padding mask would be helpful. Thus, we select a transformer backbone from BERT [40]. Each transformer block is eight layers of four-head self-attention blocks. Followed up with a fully connected dense layer, we obtain $\mu \in \mathbb{R}^{d_z}$ and $\sigma \in \mathbb{R}^{d_z}$, which will then be reparameterized into the surface token $Z \in \mathbb{R}^{d_z}$, like every VAE model. To reconstruct the NURBS parameters from the NURBS feature Z, we employ multi-task decoding, namely a separate decoder for each parameter. See Figure 5.2 for the visualization of the model architecture.

During the training, we compute the loss: $L = \sum_{i=1}^{3} \|x_i - x_i'\|_2^2$, where $x_1 = P_w$, $x_2 = U$, $x_3 = V$ are the NURBS parameters, and $x_1' = P_w'$, $x_2' = U'$, $x_3' = V'$ are the reconstructed NURBS parameters. Practically, adding a KL-divergence term could help convergency. For the VAE training and inference, inputs are normalized to $[-1, 1]$ and outputs of decoding will be denormalized to $[0, 1]$. Notably, for generated NURBS parameters, we perform the following post-generation checks: we remove control points with weight less than or equal to 0, and clip knot vectors once the value in the sequence reaches 1.

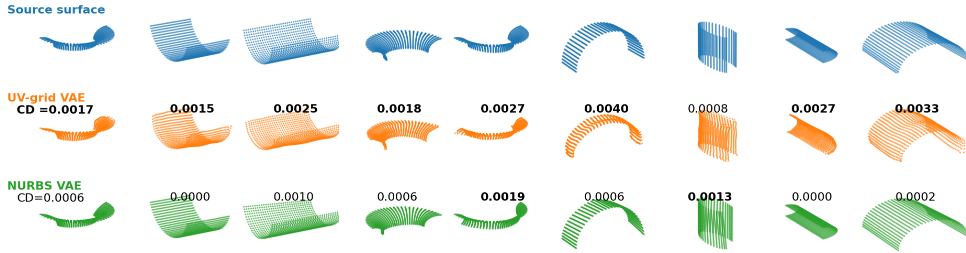


Figure 5.3: Qualitative evaluation of surface reconstruction on the ABC dataset. Considering that both pipelines work perfectly well on simple surfaces, we select surfaces of degree ≥ 5 for qualitative evaluation. We visualize results in evenly sampled points and marked measured CD (the higher the worse) on the sample. *First row*: source surfaces; *Second row*: surfaces reconstructed by UV-grid VAE of BrepGen; *Third row*: surfaces reconstructed by NURBS-based VAE of NeuroNURBS.

5.4 Experiments

We performed several experiments to investigate the effectiveness of our NeuroNURBS. NURBS parameters are more efficient than UV-grids for surface representation and can also be used as learning data. To demonstrate this, we compare its performance with UV-grids w.r.t. accuracy, memory efficiency, and computation cost (Section 5.4.2) on the DeepCAD dataset. After showing NURBS's utility in efficient representation, we introduce NURBS features encoded by NeuroNURBS into two downstream tasks: (1) The CAD (B-Rep) generation task on the DeepCAD dataset(Section 5.4.3), and (2) The solid segmentation task on the MFCAD data set (Section 5.4.4).

5.4.1 Datasets

DeepCAD We introduce the DeepCAD [182] and the ABC [76] datasets for the experiment of surface reconstruction and solid generation. The DeepCAD dataset originally contains CAD data in the form of a sequence of engineering operations that can be converted into B-Reps with their pre-defined construction algorithm. We follow the previous work [187], i.e., filtering out invalid B-Reps shapes (in case of multiple solids or being not watertight) and B-Reps with more than 30 surfaces. We use the train-val-test splitting of DeepCAD, resulting in 69 512 B-Reps for training and 7 083 B-Reps for testing; for ABC dataset, we randomly split the samples with the 90-5-5 ratio. Closed surfaces, e.g., cone surfaces, are cut into two separate NURBS surfaces as done in [66]. In representing surfaces as UV-grids, we set $n = m = 32$, the same as [187]. For training the autoencoder in NeuroNURBS, we list all

NURBS surfaces from the training solids and remove duplicated surfaces, obtaining 59 961 unique NURBS surfaces from the DeepCAD dataset (163 633 from ABC). The preprocessing of NURBS surfaces has been detailed in Section 5.3. Here, we set the NeuroNURBS hyperparameters $d = 10, k = 10, d_z = 48$. In Section 5.3.

For the solid segmentation experiment in Section 5.4.4, we introduce the MFCAD [23] dataset of 15 488 CAD solids in the STEP file format, which is the standard file format for B-Rep. MFCAD is designed for the machining feature segmentation task, where there are 16 various face labels and each face is labeled with a certain class. We split the MFCAD solids into train-val-test datasets with a ratio of 70-15-15, resulting in 10 842 solids for training. For all surfaces from training solids, we first remove duplicated surfaces and then split the surfaces into train-val-test datasets with a ratio of 90-5-5, where we collect 24 603 NURBS surfaces for training the NeuroNURBS. The surface embedding follows the pipeline described in Section 5.3, where we set $d = 2, k = 4, d_z = 48$.

5.4.2 Surface reconstruction

We present a comparative analysis between our proposed method and the UV-grid approach for surface reconstruction. For the UV-grid representation, we adopt the surface VAE and the weights from BrepGen [187] as a baseline. Our objective is to systematically evaluate the performance of our NeuroNURBS (the NURBS VAE, introduced in Section 5.3.2) relative to the UV-grid VAE from BrepGen. The comparison focuses on several key criteria: learning data size, model size, reconstruction error, distribution learning capability and NURBS construction speed. A summary of the results is provided in Table 5.1.

Input data size We randomly sample 10k surfaces from the source solids, collecting the corresponding NURBS parameters and sampling UV-grid on a 32×32 resolution for each surface. As shown in Table 5.1, representing surfaces with NURBS parameters reduces memory usage by 96.7% compared to UV-grid data. For solid representations, B-Rep entities are structured as trees and surfaces are encoded either as UV-grids or NURBS. In this setting, using NURBS for surface representation leads to a 79.9% reduction in memory cost.

Computational cost and reconstruction error The VAE model size and computational requirements differ significantly between representations. In Table 5.1, we observe the surface VAE from BrepGen has 84M parameters, which takes a UV-grid

of dimension 32×32. In contrast, our method requires only 6 million parameters. When training both VAEs with a batch size of 512, the UV-grid VAE requires 17.61 GB of GPU memory, while our NURBS-based method uses only 2.35 GB—an 86.7% reduction.

For reconstruction accuracy, we report the average Chamfer Distance (CD) and Earth Mover's Distance (EMD) between input and reconstructed surfaces, evaluated on 10k pairs with 32×32 uniformly sampled points per surface. In the early stages of this work, we evaluated the reconstruction using MMD, but it turned out to be unsuitable for reconstruction accuracy because it does not directly measure the distance between the source and the generated surface. The results of CD and EMD, summarized in Table 5.1, quantify the reconstruction error for both methods. For UV-grid representations, source and reconstructed UV-grids are directly used for evaluation. We note that the performance of NURBS VAE and UV-grid VAE is comparable in the case of sufficient training, while NURBS has a clear advantage in terms of representation efficiency. In Figure 5.3, we demonstrate the reconstruction of complex surfaces labeled with chamfer distances. Although the undulation problem is easily observed on surfaces reconstructed by UV-grid VAE, their measured CDs are concerningly better than the ones of NURBS VAE.

NURBS construction In B-Rep generation tasks, UV-grid representations require a surface fitting step (e.g., `GeomAPI_PointsToBSplineSurface` in `PythonOCC`) to convert grids to NURBS surfaces. In contrast, our NeuroNURBS approach only requires denormalizing the NURBS parameters (see Section 5.3.1) and passing them to a CAD tool (e.g., `Geom_BSplineSurface` in `PythonOCC`), which is computationally inexpensive. As shown in Table 5.1, our method increases NURBS construction speed by 92.9%.

Most importantly, as a main flaw of using the UV-grid representation and the biggest advantage of using NURBS directly, we investigate the degree evolution during surface reconstruction. Table 5.2 presents the degree distributions for the test set, the surface VAE, and NeuroNURBS. Our method precisely matches the degree distribution of the test data, as it directly learns a latent representation of the NURBS parameters. In contrast, UV-grid methods tend to produce surfaces with higher polynomial degrees, leading to increased data size of resulted NURBS and the introduction of artificial undulations, as illustrated in Figure 5.1. For a fair comparison, we have removed the lower degree constraint in the `PythonOCC` approximation function. In the Supplementary Material, we also add the surface degree distribution of real-world

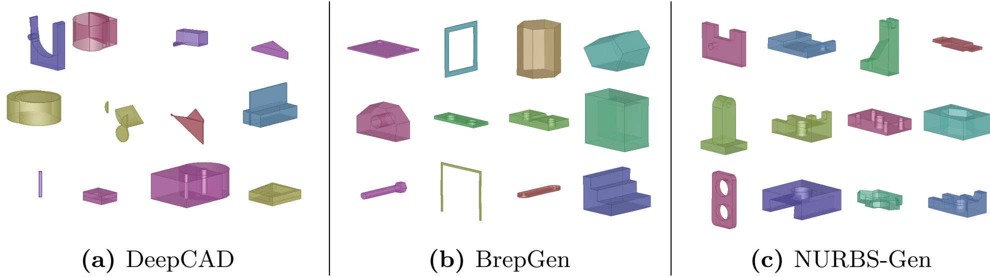
Table 5.2: The empirical distribution of the degree of polynomials along U(V) direction are measured in two tasks: surface reconstruction and generation. To reconstruct surfaces, BrepGen heavily leverages the UV-grid VAE and the UV-to-NURBS approximation function, which causes serious distortions in the surface properties (e.g., planes become uneven), as we highlight in red in the table. Note that, for this test, we remove the bottom limitation of degree in the `PythonOCC` approximation function. On the other hand, having the same degree distribution in both test and reconstructed surfaces means **a perfect reconstruction** of our NeuroNURBS. In addition, we added the surface degree distribution of real-world solids (car rims), where 30.41% of the high-degree surfaces (≥ 5) have a degree of 5.

NURBS source	Distribution of surface degrees			
	$d=2$	3	4	≥ 5
Surface reconstruction				
Test data (surfaces)	88.59%	2.03%	0.00%	9.38%
UV-grid VAE [187]	0.00%	94.01%	5.92%	0.07%
NeuroNURBS	**88.59%**	**2.03%**	**0.00%**	**9.38%**
Solid generation (DeepCAD dataset)				
Test data (solids)	93.45%	2.32%	0.00%	4.50%
BrepGen [187]	0.00%	0.00%	99.99%	0.01%
NURBS-Gen	88.69%	0.88%	0.00%	10.43%
Solid generation (ABC dataset)				
Test data (solids)	82.26%	8.26%	0.04%	9.44%
BrepGen [187]	0.00%	0.00%	99.76%	0.23%
NURBS-Gen	80.26%	2.96%	0.26%	16.5%
Real-world solids	35.96%	28.51%	0.81%	30.41% (d=5) + 3.65%

solids (car rim), where which sharpens the drawbacks of using a UV-grid.

5.4.3 CAD (B-Rep) Generation

BrepGen [187] generates B-Rep solids by progressively creating the B-Rep entities (i.e., faces, edges, and vertices) following a hierarchical tree structure from top to bottom with four diffusion models, which are Transformer-based DDPM [61]. For generating faces and edges, each node in the tree structure contains a node feature representing the global bounding box of each face/edge and a latent code for local geometry, e.g., the shape of the surface. BrepGen represents all B-Rep surfaces as UV-grids, and fits UV-grids to NURBS surfaces after generation. Here, we substitute the surface VAE model, implemented by BrepGen for autoencoding UV-grids, with our NeuroNURBS (Section 5.3.2). Technically, we set the latent dimension of our NURBS VAE to $d_z = 48$, which is exactly the same as the latent dimension of surface VAE in BrepGen. Then, we fine-tune the four pre-trained diffusion models in BrepGen



(a) DeepCAD **(b)** BrepGen **(c)** NURBS-Gen

Figure 5.4: Qualitative evaluation. It is observe that DeepCAD often generates invalid (i.e., multiple and collapsed) solids; while BrepGen and our NURBS-Gen are able to synthesize clean and upstanding B-Rep solids. In Figure 5.1, we visualize generated samples from BrepGen and NURBS-Gen with zooming-in for a deep-diving qualitative evaluation.

(i.e., diffusion models for face bounding box, face geometry, edge bounding box, edge geometry, and vertex coordinates) on the DeepCAD data set. We call the resulting model NURBS-Gen. As for the details, face denoisers are fine-tuned with a batch size of 256 for 1 000 epochs, the edge denoisers with a batch size of 64 for 500 epochs, and the training is conducted using one NVIDIA A10G GPU. As for the inference, the sampling process of NURBS-Gen follows the pipeline of BrepGen and the surface decoding is replaced by the decoder in NeuroNURBS.

Results We generate 3 000 B-Rep solids with DeepCAD model [182] (only for DeepCAD data, because the ABC data contains no construction operations for training DeepCAD), BrepGen [187], and our NURBS-Gen and compare them to 1 000 test solids. For the DeepCAD model and BrepGen, we utilize their model checkpoints that are pre-trained on the target dataset. Between the generated and test solids, we compute Minimum Matching Distance (MMD), Coverage (COV), and Jensen-Shannon Divergence (JSD) metrics. To calculate these metrics, we convert each solid into a point cloud with 2 000 points, following the evaluation method in [182, 187]. We show the results in Table 5.3.

We argue that the above evaluation method, which takes as input point clouds, might be biased toward methods like BrepGen, which utilizes UV grids. Hence, we additionally measure the Fréchet Inception Distance (FID) [58] between the 3D solids rendered in 2D with a certain viewpoint. We repeat the FID calculation for 20 different viewpoints and report an average FID value, as proposed in [184, 196].

To check the validity of the generated solids, we first convert each generated B-Rep solid to a STEP file until 3 000 STEP files are obtained. Then, we use the

Table 5.3: In the CAD (B-Rep) generation task, we list the performance of NURBS-Gen and other related models in terms of the COV, MMD ($\times 10^{-2}$), JSD ($\times 10^{-2}$), and Valid ratio.

Model	MMD ↓	JSD ↓	COV (%) ↑	FID ↓	Valid (%) ↑
DeepCAD dataset					
DeepCAD	1.41	3.92	**77.9**	**14.36**	34.79
BrepGen	**1.02**	1.16	74.7	30.04	60.95
NURBS-Gen	1.10	**0.98**	73.9	27.24	**64.58**
ABC dataset					
BrepGen	1.68	3.1	46.4	24.28	48.1
NURBS-Gen	**1.51**	**3.0**	**52.3**	**21.12**	**50.7**

`BRepCheck_Analyzer()` and `IsValid()` functions from `pythonOCC` to test the watertightness of each solid. From this process, we calculate a valid ratio among all generated solids (including the ones not saved as STEP files) for each method.

Based on the results of the quantitative evaluation in Table 5.3, NURBS-Gen consistently outperforms BrepGen with a slight edge. Note that the main advantages brought by our approach are representation efficiency, better surface quality and a CAD-native approach. Also, we visualize some generated solids from DeepCAD, BrepGen, and NURBS-Gen in Figure 5.1 and Figure 5.4. In Figure 5.1, we observe that BrepGen solids are undulating, i.e., they exhibit the wobbly geometry described in [187]. In contrast, our NURBS-Gen can generate smooth and regular surfaces and accurately join the surfaces.

5.4.4 Segmentation

In the task of machining feature segmentation [23, 36], current state-of-the-art model [67] tends to leverage increasing information of geometry or topology (e.g., face normal vector, face geometry, trimming mask, face adjacent or edge features) from the source B-Rep solid to perform accurate machining feature recognition.

We design a NURBS-based segmentation model, NURBS-GAT, that leverages a Graph Attention Network (GAT) [175] to operate the graph data derived from B-Rep. Replacing the graph convolutional network (GCN) in CADNet with a GAT is due to constant memory issues when increasing the size of the node features. For the graph data, a stack of NURBS feature encoded with NeuroNURBS, face normal vector $\boldsymbol{n}$ and coefficient d calculated from a planar equation serves as the $(48+3+1)$-dimensional node feature V and the face adjacency serves as edge connection, as shown

Table 5.4: Ablation study on machining feature segmentation task. Although UV-GAT shows a comparable results to our NURBS-GAT, but their model size is significantly larger.

Ablated model	Acc. (%)	#Param.
CADNet (with GAT)	92.18	0.02M
UV-GAT	99.63	34.2M
NURBS-GAT (only NURBS feature)	97.03	1.28M
NURBS-GAT	**99.65**	1.29M

in Figure 5.5. Now, we use the ablation study to show the importance of each input feature by conducting the ablation study on NURBS-GAT.

NURBS-GAT (only NURBS features) We remove the face normal vector $\boldsymbol{n}$ and coefficient d from the node features of NURBS-GAT, resulting in node feature $V \in \mathbb{R}^{48}$.

CADNet (with GAT) We remove the NURBS feature from the node of NURBS-GAT, resulting in node feature $V \in \mathbb{R}^{4}$.

UV-GAT We replace the NeuroNURBS embedding with UV-grids embedding in node features of NURBS-GAT, resulting in node feature $V \in \mathbb{R}^{52}$. The UV-grids embedding is obtained by training a surface VAE model, same as in Section 5.4.2, on MFCAD surface training data.

The evaluation in ablation study is conducted by reporting the per-face accuracy in Table 5.4, following the usual process in the machining feature segmentation task [36, 67]. Observed from the results, utilizing the NeuroNURBS embedding is able to improve the per-face accuracy from 92.18% to 99.65%, which shows a comparable result to UV-grid embedding. In Figure 5.6, we visualize the segmentation task and its results: since the per-face accuracy has already achieved 99.65%, the predicted labels are the same as the ground truth in the qualitative evaluation.

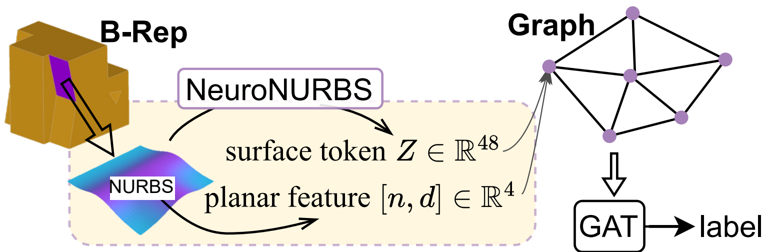


Figure 5.5: NURBS-GAT diagram.

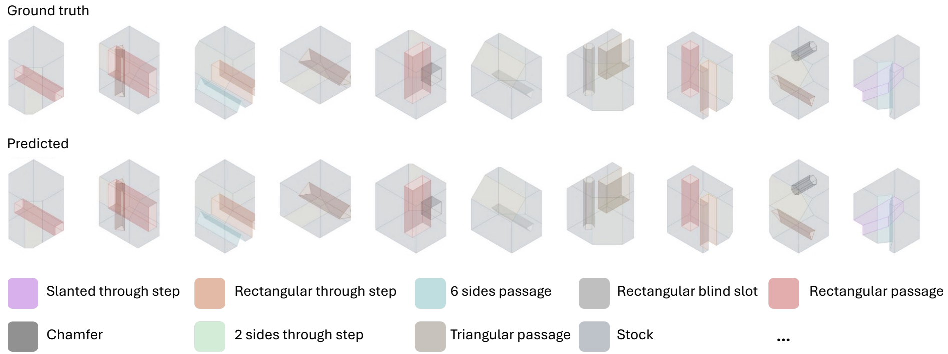


Figure 5.6: Qualitative evaluation of NURBS-GAT in machining feature segmentation task.

5.5 Implementation Details

Get NURBS parameters. In a B-Rep solid, the surfaces usually exist in the form of trimmed NURBS surfaces. To convert them into NURBS parameters, i.e., control points, weights, and knot vectors, we design the following pipeline with functions from `OpenCascade` and `pythonOCC` [124]:

1. Convert the input with form of `TopoDS_Face` into a untrimmed NURBS surface with function `BRepBuilderAPI_NurbsConvert()`, followed up with `Surface()` from `BRep_Tool`.

2. Use `SurfaceToBSplineSurface()` from `geomconvert` to convert the NURBS surface into the from of `Geom_BSplineSurface`.

3. Finally, list the NURBS parameters from the `Geom_BSplineSurface` surface with `Pole()`, `Weight(u,v)`, `UKnotSequence()` and `VKnotSequence()`.

Note that the `BRepBuilderAPI_NurbsConvert` function can be applied to planes, resulting in 4 corner points and U/V knot vectors $[0, 0, 1, 1]$.

NURBS construction from NURBS parameters After removing the padded values from NURBS parameters in Section 5.3.1, the parameters need to be converted into certain formats, so that they can be used for defining the `Geom_BSplineSurface`: Every control point is converted in the form of `gp_Pnt` and the tensor of control points is presented in the form of `TColgp_Array2OfPnt`, serving as `Poles`; Weights will be seperated and stored in the form of `TColStd_Array2OfReal`, serving as `Weights`; We

round the values in the reconstructed U(V)-knot vector to 1 decimal and store its sorted set (a ordered list of unique values) in the form of `TColStd_Array1OfReal()`, which serves as `U(V)Knots`; and the number of times each variable repeated is recorded in `TColStd_Array1OfInteger`, serving as `U(V)Mults`. The constructed NURBS is then `Geom_BSplineSurface`(Poles, Weights, UKnots, Vknots, UMults, VMults, UDegree, VDegree).Surface().

One may concern that our postprocessing brings damage to the surface geometry. As we already pointed out in Table 5.2, using UV-grids brings real damage to the surface geometry, whereas using natural parameters helps. Besides, our evaluation results will point it out if the postprocessing brings damage to the geometry. Processing the NURBS parameters is to make it valid for the PythonOCC API and easier to remove padding. This does not cause significant geometric damage, which would otherwise destroy the shape and the evaluation results. In early training we need this for model validation, otherwise the generated NURBS parameters cannot form a NURBS surface with PythonOCC, after the model being well-trained, the postprocessing won't bring much difference to the predict values.

NURBS approximation from UV-grids In Section 5.4.2, we conduct an experiment to study the distribution of U(V) degree in NURBS construction with NURBS parameters and UV-grids. The results displayed in Table 5.2. As mentioned in Section 5.4.2, to obtain UV-grids the function `GeomAPI_PointsToBSplineSurface` is used. In the BrepGen pipeline, they constrain the range of resulting U(V) degree to be $[3, 8]$. In this experiment, we use $[0, 8]$, i.e., removing the minimum limitation, for a fair comparison.

5.6 Conclusion

With data representation becoming one of the major challenges for solid learning, our work directly operates the native representation of B-Rep surfaces, i.e., NURBS parameters, by proposing NeuroNURBS. Our NeuroNURBS can embed NURBS parameters into a low-dimension NURBS feature. Using NeuroNURBS has demonstrated its ability of significantly reducing storage and GPU consumption, while delivering a comparable performance to state-of-the-art method on both solid generation and segmentation. We also observe that NeuroNURBS can remarkably help with the undulation problem in generated surfaces, which could be caused by using an approximate representation (such as UV-grids) to represent source surfaces.

As a first attempt to integrate NURBS parameters into solid learning tasks, there is still much room for improvement in neural networks, e.g., directly learning on NURBS parameters without using an autoencoder. For the solid generation task, the edge geometry in NURBS-Gen is still represented by sampling points in the parametric domain, and it is also technically possible to replace it with NURBS parameters.

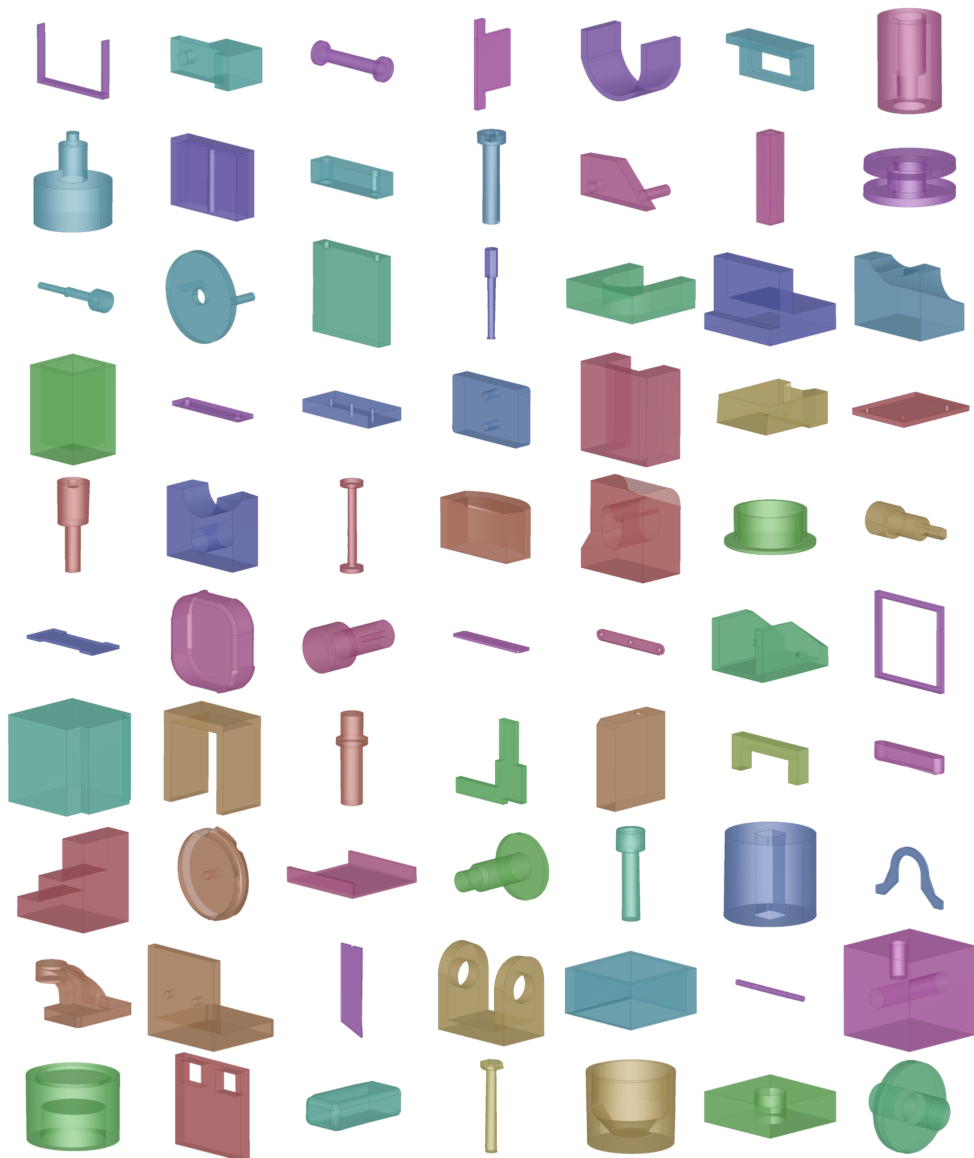


Figure 5.7: Qualitative evaluation of NURBS-Gen on the ABC dataset.