CAPRI-Net: Learning Compact CAD Shapes with Adaptive Primitive Assembly

Fenggen Yu[1] Zhiqin Chen[1] Manyi Li[1,3] Aditya Sanghi[2] Hooman Shayani[2]
Ali Mahdavi-Amiri[1] Hao Zhang[1]
[1]Simon Fraser University [2]Autodesk AI Lab [3]Shandong University

Abstract

We introduce CAPRI-Net, a self-supervised *neural network for learning* compact *and* interpretable *implicit representations of 3D computer-aided design (CAD) models, in the form of* adaptive primitive assemblies. *Given an input 3D shape, our network reconstructs it by an assembly of quadric surface primitives via constructive solid geometry (CSG) operations. Without any ground-truth shape assemblies, our self-supervised network is trained with a reconstruction loss, leading to faithful 3D reconstructions with sharp edges and plausible CSG trees. While the parametric nature of CAD models does make them more predictable locally, at the shape level, there is much structural and topological variation, which presents a significant generalizability challenge to state-of-the-art neural models for 3D shapes. Our network addresses this challenge by adaptive training with respect to each test shape, with which we* fine-tune *the network that was pre-trained on a model collection. We evaluate our learning framework on both ShapeNet and ABC, the largest and most diverse CAD dataset to date, in terms of reconstruction quality, sharp edges, compactness, and interpretability, to demonstrate superiority over current alternatives for neural CAD reconstruction.*

1. Introduction

Computer-Aided Design (CAD) models are ubiquitous in engineering and manufacturing to drive decision making and product evolution related to 3D shapes and geometry. With the rapid advances in AI-powered solutions across all relevant fields, several CAD datasets [29, 50, 57] have emerged to support research in geometric deep learning. A common characteristic of CAD models is that they are composed of well-defined parametric surfaces meeting along sharp edges. While the parametric nature of the CAD shapes do make them more predictable locally and at the primitive level, at the *shape* level, there is a great deal of *structural* and *topological variations*, which presents a significant *generalizability* challenge to current neural models for 3D shapes [8, 10, 21, 30, 35, 37, 40, 59, 63]. On the other hand, existing networks for primitive fitting typically focus on abstractions with simple, hence limited, primitives [39, 42, 54, 64], hindering reconstruction quality.

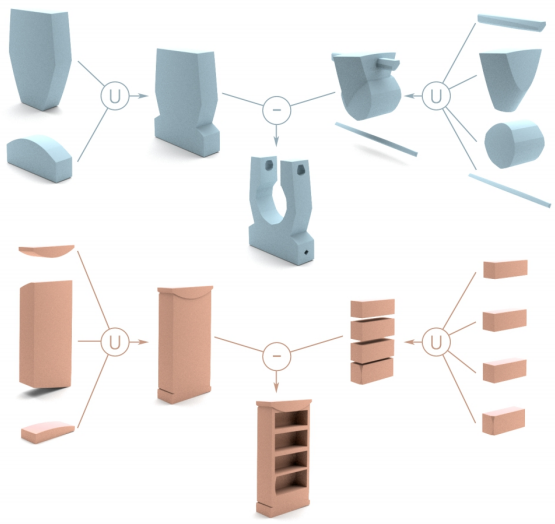

Figure 1. Our network learns *compact* and *interpretable* implicit representations of 3D CAD shapes in the form of primitive assemblies via CSG operations, without any assembly supervision.

In this paper, we develop a learning framework for 3D CAD shapes to address these very challenges. Our goal is to design a neural network that can learn a *compact* and *interpretable* representation for CAD models, leading to high-quality 3D reconstruction, while the network generalizes well over ABC [29], the largest and most *diverse* CAD dataset to date. This dataset is a collection of one million CAD models covering a wide range of structural, topological, and geometric variations, *without* any category labels, in contrast to other prominent repositories of man-made shapes such as ShapeNet [4] and ModelNet [60] which have only limited[1] number of object categories. Hence, targeting the ABC dataset poses a real generalizability challenge.

Our network takes an input 3D shape as a point cloud or

[1]While the full ShapeNet dataset has 270 object categories, to the best of our knowledge, most learning methods only work with up to 13.

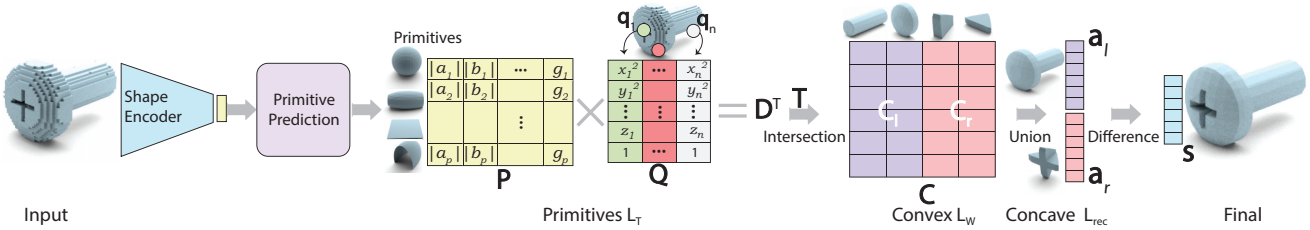

Figure 2. Overview of our network. Given an input 3D shape as a point cloud or voxels, we first map it into a latent code using an encoder. This latent code is used to predict p primitives with parameters included in $\mathbf{P}$. For any query point $\mathbf{q}_j$ packed in matrix $\mathbf{Q}$, we can obtain the matrix $\mathbf{D}$ indicating approximate signed distance from the query point to each primitive. A selection matrix $\mathbf{T}$ is used to select a small set of primitives from the primitive set to group convex shapes in matrix $\mathbf{C}$ which indicates inside/outside values for query points w.r.t convex shapes. Then, we perform min operation on each half of $\mathbf{C}$ (i.e. $\mathbf{C}_l$ and $\mathbf{C}_r$) to union convex shapes into two (possibly) concave shapes and get inside/outside indication vectors $\mathbf{a}_l$ and $\mathbf{a}_r$ for left and right concave shapes. Finally, we perform a difference operation as $\mathbf{a}_l - \mathbf{a}_r$ to obtain the final point-wise inside/outside indicator $\mathbf{s}$. $L_\mathbf{T}$, $L_\mathbf{W}$, and L_{rec} are the loss functions we define for our network.

voxel grids, and reconstructs it by a compact *assembly* of *quadric surface primitives* via constructive solid geometry (CSG) operations including intersection, union, and difference. Specifically, the learned quadrics are assembled by a series of binary selection matrices. These matrices intersect the quadrics to form convex parts, with union operations to follow to obtain possibly concave shapes, and finally a difference operation naturally models holes present in high-genus models, which are frequently encountered in CAD.

The architecture of our network, shown in Figure 2, is inspired by BSP-Net [8]. At the high level, it is a coordinate-based network trained with an occupancy loss reflecting the reconstruction error; we also add a novel loss to accommodate a *new* difference operation. The reconstruction is performed in a latent space that is obtained by an encoder applied to the input shape. The other learnable parameters of the network include matrices that define the parameters of the quadric surfaces and the CSG operations, respectively, as well as MLP weights which map the latent code to the primitive parameter matrix. The resulting reconstruction is in the form of a CSG assembly, while the network training does not require any ground-truth shape assemblies — the network is *self-supervised* with the reconstruction loss.

Due to the significant variations among CAD shapes in ABC, we found that our network, when trained on a model collection only, does not generalize well. In fact, none of the existing reconstruction networks we tested, including IM-Net [10], OccNet [35], DISN [63], BSP-Net [8], generalized well on the ABC dataset. We tackle this issue by further *fine-tuning* the network, that is pre-trained on a training set, to a test CAD shape, so that the resulting network is *adaptive* to the test shape. Both pre-training and fine-tuning are performed using the same network architecture, shown in Figure 2, except that the encoder is not re-trained during fine-tuning for efficiency. We coin our learning framework CAPRI-Net, as it is trained to produce *Compact* and *Adaptive PRI*mitive assemblies for CAD shapes.

We evaluate CAPRI-Net on both ABC and ShapeNet, in terms of reconstruction quality, compactness as measured by primitive counts, and interpretability as examined by how natural the recovered primitive assemblies are. Qualitative and quantitative comparisons are made to BSP-Net [8], UCSG [28], and CSGStump [12], a concurrent work, all of which are representative of state-of-the-art learning methods suitable for CAD shapes. The results demonstrate superiority of CAPRI-Net on all fronts.

The primary application of CAPRI-Net is 3D shape reconstruction, where the network produces a novel implicit field that is unlike those obtained by classical computer graphics methods [23], e.g., a radial basis function [3], or those learned by recent neural implicit models, e.g., IM-Net [10], OccNet [35], and SIREN [48], etc. Our implicit representation is *structured*, as a compact assembly, and leads to quality reconstruction of 3D CAD models.

2. Related Work

We cover prior methods most closely related to our work, including classical and emerging techniques for geometric primitive fitting, as well as recent approaches for learning implicit and structured 3D representations.

Primitive detection and fitting. There has been extensive work in computer graphics and computer-aided geometric design on primitive fitting. Given a raw 3D object, different algorithms such as RANSAC [17] and Hough Transform [24] have been used to detect primitives [27]. RANSAC based methods have been used in [32, 45], to detect multiple primitives in dense point clouds. However, basic primitive fitting techniques such as RANSAC are limited to predefined primitive types and cannot predict the shapes that need to be subtracted in a complex CSG tree. Methods such as [15, 19] have then extracted a CSG tree from the raw input. Hough Transform has been used to detect planes [2] and cylinders [44] in point clouds. However, these meth-

ods usually do not generalize as they require different hyper parameters per shape. Recently, neural net based algorithms have been used to detect and fit primitives on point-clouds [31, 47]. SPFN [31] uses supervised learning to first detect primitive types and then estimate the parameter of the primitives. ParseNet [47] extends SPFN by in-cooperating spline patches and using differentiable metric-learning segmentation. Our method differs in requiring *no supervision*, being able to reconstruct a compact primitive assembly.

Neural implicit shape representations. Neural implicit surface representation [10, 35, 40] have gained immense popularity because of the ability to generate complex, high spatial resolution 3D shapes while using a small memory footprint during training. This representation has been used in 3D domain for part understanding [9, 14, 59], unsupervised single view reconstruction [26, 34, 38, 55] and scene or object completion [11, 43]. More recently, neural implicit representations have also been extended to 2D images and videos for application in single scene completion [36, 48, 52], image generative models [1, 16, 49], super resolution [7] and dynamic scenes [33, 41, 62]. In our work, we extend the use of neural implicit surface representation to CAD models and show how our method can overcome some of the challenges presented by CAD designs.

Structural representations. Structure-aware 3D representations are typically defined by atomic geometry entities and their relations [5], where the atomic entities can represent semantic parts [20, 30, 37, 51, 59, 61] or lower-level geometric primitives [8, 13, 42, 54]. These entities can be combined to form complete shapes as part collections [42, 54], tree structures [8, 13, 28, 46], scene graphs [18, 56], hierarchical graphs [30, 37], or structural programs [53].

In CAPRI-Net, we represent the atomic geometry entities using primitives defined by a constrained, implicit form of quadric equations. Works such as Superquadrics [42] have used unconstrained explicit forms of superquadratic equations but lead to primitives with low interpretability for CAD modeling. Furthermore, they cannot produce plane-based intersections to reconstruct sharp features.

Neural networks designed to learn CSG primitive assemblies include CSG-Net [46] and DeepCAD [58] which require supervision and others [8, 12, 28] that are unsupervised. Among the latter, BSP-Net [8] is most closely related to our work, but they differ in several significant ways: ① CAPRI-Net can generate several different interpretable primitives from a single quadric equation, while BSP-Net only uses planes. ② Unlike BSP-Net, our network includes a difference operation which is well-suited in CAD modeling. ③ In addition, we introduce a new loss term which encourages the use of the difference operation, resulting in more compact and more natural CSG assemblies.

UCSG-Net [28] also learns CSG trees in an unsupervised manner, but uses only box and sphere primitives which can significantly limit the shape reconstruction quality. Also, unlike BSP-Net and CAPRI-Net, the order of the CSG operations from UCSG-Net are *dynamic* and not fixed. However, the order flexibility or generality comes at the cost of making the assembly learning task much more difficult. As we shall demonstrate, with a fixed order of CSG operations (see Figure 2), CAPRI-Net tends to produce more natural and more compact CSG trees than UCSG-Net.

In a concurrent work, CSG-Stump [12] also follows a fixed order of CSG operations. Key differences to our work include: ① CAPRI-Net relies on a simple quadric equation to represent all primitives, while CSG-Stump needs to preset the number for primitives for each primitive type. ② Like BSP-Net [8], CSG-Stump uses a non-differentiable inverse layer to model shape differences which limits the applicability of the difference operation to simple primitives. In contrast, our network has a dedicated differentiable difference layer and a new loss term to support the handling of complex convex shapes, leading to improved interpretability of the CSG trees of our method.

3. Methodology

Our network, coined CAPRI-Net, takes as input a 3D voxel shape and reconstructs the shape by predicting a set of primitives that can be combined to make intermediate convex and concave shapes via a CSG tree. In CAPRI-Net, primitives are represented implicitly by a quadric equation, which determines whether query point $\mathbf{q}_j = (x, y, z)$ is inside or outside that primitive. Our selected quadric equation has the capability of producing several useful primitives including spheres, planes, or cylinders.

Voxelized inputs are fed to a 3D convolutional network, the same as the encoder in IM-Net [10], to produce a latent code that is passed to our primitive prediction network (see Figure 2). The output of the primitive prediction network is matrix $\mathbf{P}$ that holds the parameters of the predicted primitives and is used to determine the *signed distance* of the query points from all predicted primitives. These signed distances are then passed to three *CSG layers* to output the occupancy value for each query point, indicating whether a point is inside or outside the shape.

3.1. Primitive Prediction

After obtaining a shape code with size 256 from an appropriate encoder, the code is passed to our primitive prediction network which is a multi-layer perceptron (MLP) to output primitives' parameters. To represent primitives, we use the following quadric equation:

$$ax^2 + by^2 + cz^2 + dx + ey + fz + g = 0. \qquad (1)$$

Our MLP outputs $\mathbf{P}_{p\times 7}$, where p is the number of primitives represented by seven parameters (a, b, c, d, e, f, g).

Specifically, $\mathbf{P}(i,:)$, which is the ith row of matrix $\mathbf{P}$, has the parameters of the ith primitive.

We sample n points near a given shape and learn the parameters of the p primitives to produce the same in/outside as the ground truth. The approximate signed distances of all points to all primitives are calculated as a matrix multiplication: $\mathbf{D}^T = \mathbf{PQ}$. Given a query point $\mathbf{q}_j = (x_j, y_j, z_j)$, we get its signed distances to all primitives as: $\mathbf{D}(j,:) = \mathbf{PQ}(:,j)$, where $\mathbf{Q}(:,j) = (x_j^2, y_j^2, z_j^2, x_j, y_j, z_j, 1)$ is the jth column of $\mathbf{Q}$. Therefore, for n query points in training, signed distance matrix $\mathbf{D}_{n\times p}$ stores the distance between point $\mathbf{q}_j$ and the ith primitive at $\mathbf{D}(j,i)$. Note that $\mathbf{D}$ only provides *approximate* signed distances, which are used to estimate in/outside occupancy for our reconstruction loss.

As Equation (1) is quite general and capable of producing rather complex shapes, we constrain the first three parameters a, b, c to be *non-negative* to ease the learning task while still covering frequently used convex quadric surfaces in CAD, such as planes, cylinders, ellipses, and paraboloids. But this would preclude primitives such as cones and tori.

3.2. CSG Operations

Constructive Solid Geometry (CSG) provides flexibility and accuracy in shape design and modeling. Therefore, being able to interpret the results in the form of a CSG tree has great benefits. In CAPRI-Net, we employ the three main CSG operations: intersection, union and difference in a fixed order to combine primitives and produce a shape. We now discuss how to incorporate these operations in our network to produce an interpretable and efficient CSG tree.

Intersection. By applying intersection operations, primitives that have been previously identified are combined to make a set of convex shapes. Having the signed distance matrix $\mathbf{D}_{n\times p}$ indicating point to primitive distances, primitives involved in forming convex shapes are selected by a selection matrix $\mathbf{T}_{p\times c}$. Relu activation function is first applied on $\mathbf{D}$ to map all the points inside the primitives to zero. Therefore, multiplying relu$(\mathbf{D})$ and $\mathbf{T}$ provides a matrix containing point to convex distances. In fact, this matrix multiplication serves as an implicit intersection operation among primitives. This way, only when $\mathbf{C}(j,i) = 0$, query point $\mathbf{q}_j$ is inside convex i:

$$\mathbf{C} = \text{relu}(\mathbf{D})\mathbf{T} \quad \begin{cases} 0 & \text{inside,} \\ > 0 & \text{outside.} \end{cases} \tag{2}$$

Union. After applying intersection operations on primitives and forming a set of convex shapes, the convex shapes are combined to obtain more complex and possibly concave shapes. In CAPRI-Net, we group convex shapes into two separate shapes whose in/outside indicators are stored in $\mathbf{a}_r$ and $\mathbf{a}_l$. This gives the network the option to learn two shapes that can later be fed into difference layer to produce desired concavities or holes. To do so, we first split $\mathbf{C}$ into two sub-matrices $\mathbf{C}_l$ and $\mathbf{C}_r$ both in size $n \times \frac{c}{2}$. $\mathbf{C}_l$ and $\mathbf{C}_r$ contain the in/outside indicators of all points to all convex shapes that are going to be combined into one shape on the left ($\mathbf{a}_l$) and another on the right ($\mathbf{a}_r$).

To compute $\mathbf{a}_l$ and $\mathbf{a}_r$, two different functions and notations are used according to our training stage. $\mathbf{a}^+$ and $\mathbf{a}^*$ respectively refer to early and later stages. Details of our multi-stage training are presented in Section 3.3.

We obtain $\mathbf{a}_l^*$ and $\mathbf{a}_r^*$ by applying a min operation on each row of $\mathbf{C}_l$ and $\mathbf{C}_r$. This way, if point $\mathbf{q}_j$ is inside any of the convex shapes in $\mathbf{C}_l$, it will be considered as inside in $\mathbf{a}_l^*$. Assume that $\mathbf{C}_l(j,:)$ is the jth row of $\mathbf{C}_l$ representing in/outside indicators of point $\mathbf{q}_j$ with respect to convex shapes, then we define:

$$\mathbf{a}_l^*(j) = \min_{i<\frac{c}{2}}(\mathbf{C}_l(j,i)) \quad \begin{cases} 0 & \text{inside,} \\ > 0 & \text{outside,} \end{cases} \tag{3}$$

where $\mathbf{a}_r^*$ is defined similarly with $i \geq \frac{c}{2}$, and $\mathbf{a}_l^*$ and $\mathbf{a}_r^*$ are $n \times 1$ vectors indicating whether a point is in/outside of the left/right concave shape.

Using the min operation, gradients can be only back-propagated to the convex shape with the minimum value. Therefore, other convex shapes cannot be adjusted and tuned during training. To facilitate learning, we distribute gradients to all convex shapes by employing a (weighted) sum in the early stage of our multi-stage training:

$$\mathbf{a}_l^+(j) = \mathscr{C}(\sum_{i<\frac{c}{2}} \mathbf{W}_j \mathscr{C}(1 - \mathbf{C}_l(j,i))) \quad \begin{cases} 1 & \approx \text{inside,} \\ < 1 & \approx \text{outside,} \end{cases} \tag{4}$$

where $\mathbf{W}_j$ is a weight vector and $\mathscr{C}$ clips the values into $[0,1]$, and $\mathbf{a}_r^+$ is defined similarly with $i \geq \frac{c}{2}$.

This function gives the chance to all elements of $\mathbf{C}$ to be adjusted during training. However, in this multiplication, small values referring to outside points can add up to a value larger than 1 which classifies an outside point as inside. Therefore, the in/outside in Equation (4) are only approximate estimates. To avoid vanishing gradients, we initially set $\mathbf{W}_j$ to very small values (i.e., 10^{-5}) and gradually increase $\mathbf{W}_j$ to be 1 by Equation (9). This setup helps the network converge to proper values for $\mathbf{a}^+$ at early stages that are finalized in $\mathbf{a}^*$ at later training stages.

Difference. Here, we introduce our novel difference operation which helps the network produce more complex and sharper objects. After we obtain $\mathbf{a}_l^*$ and $\mathbf{a}_r^*$, we perform a difference operation as below:

$$\mathbf{s}^*(j) = \max(\mathbf{a}_l^*(j), \alpha - \mathbf{a}_r^*(j)) \quad \begin{cases} 0 & \text{inside,} \\ > 0 & \text{outside,} \end{cases} \tag{5}$$

$$\mathbf{s}^+(j) = \min(\mathbf{a}_l^+(j), 1 - \mathbf{a}_r^+(j)) \begin{cases} 1 & \approx \text{inside}, \\ < 1 & \approx \text{outside}, \end{cases} \quad (6)$$

where $\mathbf{s}$ is used to reconstruct a shape since $\mathbf{s}(j)$ indicates whether a query point $\mathbf{q}_i$ is in/outside. Note that α needs to be small and positive, since $\mathbf{s}^*$ is close to 0 when points are inside the shape; see parameters settings in Section 4.1.

3.3. Multi-Stage Training and Loss Functions

Directly learning a *binary* selection matrix $\mathbf{T}$ is difficult. In addition, as already discussed, $\mathbf{a}^+$ can facilitate better gradient backward propagation than $\mathbf{a}^*$. Therefore, we perform a multi-stage training scheme exploiting different training strategies to gradually achieve better results.

We start training by operations and functions with better gradients, such as those in $\mathbf{a}^+$. Then we switch to more accurate and interpretable operations and functions, such as those in $\mathbf{a}^*$. In summary, at stage 0, selection matrix $\mathbf{T}$ is not binary and the max operation is not used in $\mathbf{a}^+$. At stage 1, $\mathbf{T}$ is still not binary, but we use the max operation to obtain $\mathbf{a}^*$. At stage 2, $\mathbf{T}$ becomes binary to allow deterministic selection of the right primitives. In the following, we discuss each training stage in detail.

Stage 0. At this stage, we apply the following loss:

$$L^+ = L_{rec}^+ + L_{\mathbf{T}} + L_{\mathbf{W}}, \quad (7)$$

where $L_{\mathbf{T}}$ and $L_{\mathbf{W}}$ are defined similarly as in BSP-Net [8] by forcing each entry of $\mathbf{T}$ to be between 0 and 1 and each entry of $\mathbf{W}$ to be approximately 1:

$$L_{\mathbf{T}} = \sum_{t \in T} \max(-t, 0) + \max(t - 1, 0), \quad (8)$$

$$L_{\mathbf{W}} = \sum_i |\mathbf{W}_i - 1|. \quad (9)$$

L_{rec}^+ in CAPRI-Net differs from the usual reconstruction losses such as L_1 or L_2 that are normally used in deep learning networks based on implicit representations. Note that $\mathbf{s}$ in Equation (5) is considered as the final output of our network to indicate whether a query point is inside or outside the shape and $\mathbf{g}$ holds ground truth values of these values: 1 for inside and 0 for outside. Instead of defining the loss directly on $\mathbf{s}$ and $\mathbf{g}$, we use two weighted L_2 losses for $\mathbf{a}_l^+$ and $\mathbf{a}_r^+$ separately to make them complement each other and avoid vanishing gradients in the max operation of Equation (5). Note that $\mathbf{s}$ is only used during meshing and it helps remove redundant primitives; see details in the supplementary material. Our L_{rec}^+ is defined as follows:

$$L_{rec}^+ = \frac{1}{n} \sum_{j=1}^{n} [\mathbf{M}_l(j) * (\mathbf{g}(j) - \mathbf{a}_l^+(j))^2 + \mathbf{M}_r(j) * ((1 - \mathbf{g}(j)) - \mathbf{a}_r^+(j))^2], \quad (10)$$

where n is the number of query points. We define $\mathbf{M}_l(j) = \max(\mathbf{g}(j), \mathbb{1}(\mathbf{a}_r^+(j) < \beta))$ and $\mathbf{M}_r(j) = \max(\mathbf{g}(j), \mathbb{1}(\mathbf{a}_l^+(j) > \beta))$, where β is set close to the inside value 1 in Equation (4). The function $\mathbb{1}$ transfers Boolean values to float, while $\mathbf{M}_l$ and $\mathbf{M}_r$ adjust the losses on each side with respect to the value of the other side.

Stage 1. In this stage, we use $\mathbf{a}^*$, instead of $\mathbf{a}^+$, to encourage the network to produce accurate in/outside values for the left and right shapes. The loss function for Stage 1 is

$$L^* = L_{rec}^* + L_T, \quad (11)$$

where L_{rec}^* is a reconstruction loss for $\mathbf{a}_l$ and $\mathbf{a}_r$ as below:

$$L_{rec}^* = L_l^* + L_r^*, \quad (12)$$

$$L_l^* = \frac{1}{n} \sum_{j=1}^{n} \mathbf{M}_l(j) * [(1 - \mathbf{g}(j)) * (1 - \mathbf{a}_l^*(j)) + w_l * \mathbf{g}(j) * \mathbf{a}_l^*(j)], \text{and} \quad (13)$$

$$L_r^* = \frac{1}{n} \sum_{j=1}^{n} \mathbf{M}_r(j) * [\mathbf{g}(j) * (1 - \mathbf{a}_r^*(j)) + w_r * (1 - \mathbf{g}(j)) * \mathbf{a}_r^*(j)], \quad (14)$$

where $\mathbf{M}_l(j) = \max(\mathbf{g}(j), \mathbb{1}(\mathbf{a}_r^*(j) > \gamma))$, $\mathbf{M}_r(j) = \max(\mathbf{g}(j), \mathbb{1}(\mathbf{a}_l^*(j) < \gamma))$, $\mathbf{g}$ serves as a mask, and γ is set close to the inside value 0 in Equation (3).

Equation (13) acts as an L_1 loss where $(1 - \mathbf{g}) * (1 - \mathbf{a}_l)$ encourages the outside points to be one and $\mathbf{g} * \mathbf{a}_l$ encourages the inside points to remain inside with a value zero. w_l and w_r are weights to control shape decomposition structure by encouraging the left shape to cover the volume occupied by the ground truth shape and the right shape to cover a meaningful residual volume. This way, an effective subtraction is obtained with capability of producing sharp and detailed shapes with concavities and holes. We set $w_l = 10$ and $w_r = 2.5$ in all the experiments and show the effects of these weights in our ablation study.

Stage 2. In the first two stages, we use L_T to make each entry of selection matrix $\mathbf{T}$ to be a float value between 0 and 1 facilitating the learning, but this $\mathbf{T}$ is not CSG interpretable. Therefore, we quantize $\mathbf{T}$ into $\mathbf{T}_{hard}$ with float values into binary values (i.e., $t_{hard} = (t > \eta)?1 : 0$), and use intersection operation with $\mathbf{T}$ replaced by $\mathbf{T}_{hard}$ in Equation (2) for each convex shape. Since values are small in $\mathbf{T}$, we set $\eta = 0.01$ in all experiments. Having $\mathbf{T}_{hard}$, we use the same loss function in Equation (12) as stage 1 for training.

3.4. CAD Mesh Construction via CSG

During inference, after obtaining $\mathbf{P}$ and $\mathbf{T}$, we assemble the learned primitives into convex shapes and execute

the learned CSG operations to output a CAD mesh, just as in BSP-Net [8]. The output meshes contain sharp edges and regular surfaces as expected from performing the CSG; more details can be found in the supplementary material.

4. Result and Evaluation

In this section, we present qualitative and quantitative results of our experiments to demonstrate the effectiveness of CAPRI-Net. We test our network and state-of-the-art methods for neural CAD modeling on both the CAD dataset ABC [29] and ShapeNet [4]. We compare the performance of the various networks for shape reconstruction from voxels and point clouds. We also conduct ablation studies to assess the components of our network architecture.

4.1. Data Preparation and Training Details

We *randomly* chose, from the ABC dataset, 5,000 single-piece shapes whose normalized bounding box has sides all larger than 0.1. We use the same method as in IM-Net [10] to discretize these shapes into 256^3 voxels to sample 24,576 points close to the surface with their corresponding occupancy values for pre-training . In addition, we sample 64^3 voxels for the input shape that is passed to the shape encoder. For ShapeNet, we employ the dataset provided by IM-Net [10], which contains 64^3 voxelized and flood-filled 3D models from ShapeNet Core (V1) with sample points close to surface for pre-training. Since achieving satisfactory performance by other methods on ABC also requires fine-tuning and this is time consuming (e.g., 15 minutes per shape for BSP-Net), we have randomly selected a moderately sized subset of shapes as test set to evaluate the fine-tuning: 1,000 shapes from ABC, and 100 from each of the 13 categories from ShapeNet — 1,300 shapes in total.

In our experiments, we set the number of primitives as $p = 1,024$ and the number of convex shapes as $c = 64$, relatively large values to account for more complex shapes. The size of our latent code for all input types is 256 and a two-layer MLP is also used to predict the parameters of the primitives from the shape latent code. We set $\beta = 0.5$, for Stage 0, and $\gamma = 0.01$, for Stage 1, since the occupancy values get closer to 0 when points are inside the assembled concave shape at Stage 1. We set $\alpha = 0.2$ during meshing.

We first pre-train our network on a training set so as to obtain a general and initial estimate of the primitive parameters and selection matrices. During fine-tuning, conducted per test shape, we only optimize the latent code, primitive prediction network and selection matrix for each shape individually. We pre-train our network in stage 0 on our training set with 1,000 epochs, which takes ≈ 9 hours using an NVIDIA RTX 1080Ti GPU. We fine-tune our network in all stages with 12,000 iterations for each stage, taking about 3 minutes per shape overall to complete.

Methods	BSP-Net	UCSG	STUMP	Ours
CD ↓	0.491	0.300	1.180	**0.136**
NC ↑	0.868	0.877	0.829	**0.914**
ECD ↓	10.098	5.022	11.848	**2.208**
LFD ↓	1,342.7	1,494.8	2,945.2	**800.2**
#Primitives (#P) ↓	114.44	-	-	**46.93**
#Convexes (#C) ↓	11.60	12.72	90.88	**6.03**

Table 1. Comparing 3D shape reconstruction from voxels on ABC. As UCSG and CSG-Stump do not produce surface primitives, only convex solids (e.g., boxes and spheres), we count the convexes.

Methods	BSP-Net	UCSG	STUMP	Ours
CD ↓	0.220	1.317	2.288	**0.175**
NC ↑	0.869	0.815	0.792	**0.872**
ECD ↓	2.111	5.233	10.457	**2.101**
LFD ↓	2,254.4	3,582.5	4,983.2	**1,824.1**
#Primitives (#P) ↓	214.70	-	-	**61.56**
#Convexes (#C) ↓	18.86	12.40	180.54	**8.71**

Table 2. Comparing 3D reconstruction from voxels on ShapeNet.

4.2. Reconstruction from Voxels

Given a low resolution input in voxel format (i.e. 64^3), the task is to reconstruct a CAD mesh. First, we pre-train our entire network with our training set. We then fine-tune the shape's latent code, primitive prediction network, and the selection matrix for each shape individually.

At the fine-tuning stage, we first sample input voxels whose centers are close to the shape's surface (i.e. with distance up to $1/64$). We specifically keep voxels whose occupancy values differ from their neighbors. We then randomly sample other voxels' centers and obtain 32,768 points. Sampled points are then scaled into the range $[-0.5, 0.5]$, these points along with their occupancy values are used to supervise the fine-tuning step.

We compare CAPRI-Net with BSP-Net [8], UCSG [28], and CSG-Stump [12] (labelled as STUMP), which output structured parametric primitives. For a fair comparison, we fine-tune all of these networks with the same number of iterations as well. For each shape, BSP-Net needed about 15 minutes, UCSG about 40 minutes, and CSG-Stump about 60 minutes to converge while our CAPRI-Net took only about 3 minutes. Note that CSG-Stump employed different network settings for shapes from ABC (with shape differences) and ShapeNet (without shape difference) in their experiments; we followed the same settings in our comparisons. Additional results can be found in the supplementary material, where we also show results without fine-tuning.

Evaluation metrics. Our quantitative metrics for shape reconstruction are symmetric Chamfer Distance (CD), Normal Consistency (NC, higher is better), Edge Chamfer Dis-

Methods	w.o QS	w.o Diff	w.o Weight	Ours
CD ↓	0.22	0.20	0.18	**0.14**
NC ↑	0.90	0.88	0.90	**0.91**
ECD ↓	2.918	2.582	2.720	**2.208**
LFD ↓	1,083.4	985.7	886.0	**800.2**
#P ↓	57.58	62.62	58.95	**46.93**
#C ↓	**5.86**	9.62	6.64	6.03

Table 3. Ablation study on ABC; see texts for descriptions.

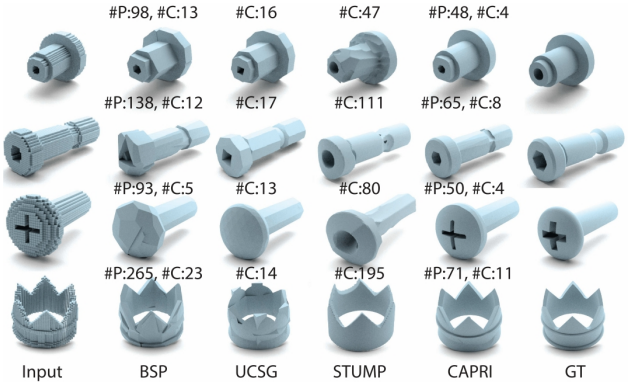

Figure 3. Visual comparisons between reconstruction results from 64^3 voxel inputs on ABC. We also show number of surface primitives (#P) and number of convexes (#C) reconstructed.

tance [8] (ECD), and Light Field Distance [6] (LFD). For ECD, we set the threshold for normal cross products to 0.1 for extracting points close to edges. CD and ECD values provided are computed on $8k$ surface sampled points and multiplied by 1,000. For LFD, we render each shape at ten different views and measure Light Field Distances.

Evaluation and comparison. We provide visual comparisons on representative examples from the ABC dataset in Figure 3 and ShapeNet in Figure 4. Our method consistently reconstructs more accurately the geometric and topological details such as holes and sharp features, owing to the richer sets of supported surface primitives and the difference operation which was not present in BSP-Net. UCSG performs well when the input can be well modeled by boxes and spheres, but not others such as the lamp in Figure 4. CSG-Stump tends to use considerably more difference operations to model ABC shapes than CAPRI-Net. This could also cause the shape surfaces to be carved by many redundant simple primitives; see the top shape in Figure 3. In contrast, CAPRI-Net generally predicts fewer convexes and redundant difference operations during inference. Also, since CSG-Stump does not support difference operations between complex shapes, it can fail to reconstruct small holes or intricate concavities; see the third row of Figure 3. Overall, with predefined primitives, neither UCSG nor CSG-Stump is capable of reproducing delicate details or achieving the

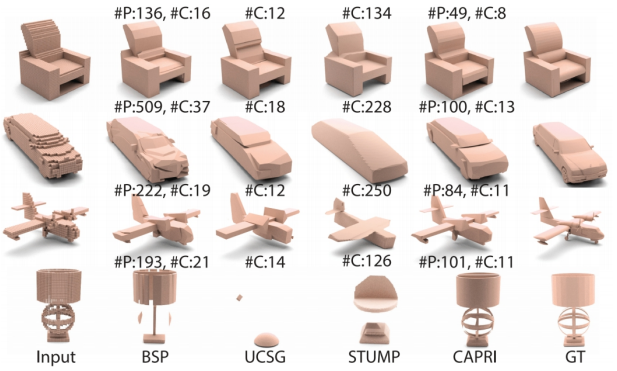

Figure 4. Some reconstruction results from voxels on ShapeNet.

kind of reconstruction accuracies as CAPRI-Net.

Quantitative comparison results are shown in Tables 1 and 2. CAPRI-Net achieves the best reconstruction quality on all metrics reported. We measure compactness of the reconstruction by counting average surface primitives and convexes the methods produce. Again, CAPRI-Net outperforms all other compared alternatives on these statistics.

CSG trees. Our network can learn to produce a plausible CSG tree from a given latent code without direct supervision as is shown in Figure 1.We provide additional CSG trees and comparisons in the supplementary material.

Ablation. We examine the effects of three important design choices we made in CAPRI-Net: quadric surface (QS) representation, difference layer (Diff), and weighted reconstruction loss (Weight). We deactivate each of these components and make three ablation studies called: w.o QS, w.o Differ, and w.o Weight; see Table 3 and visual comparison in supplementary material. It is apparent that quadric surface representation makes our method suitable for ABC dataset by using fewer appropriate primitives (e.g., cylinders) in the reconstruction. Difference operation can also offer compactness and fewer primitives in the final reconstruction. Finally, the weighted reconstruction loss helps CAPRI-Net reproduce fine details such as small holes.

4.3. Reconstruction from Point Clouds

In our last experiment, we test reconstruction of CAD meshes from point clouds, each containing $8,192$ points with normal vectors. During pre-training, we voxelize the input point clouds to 64^3 and train a 3D convolution network as the encoder to generate the shape latent code.

During the fine-tuning stage, network training adapts to the original input point clouds, not the voxels. Specifically, inspired from [25], for each point with its normal vector, we sample 8 points along its normal with Gaussian distribution $(\mu = 0, \sigma = 1/64)$. If this point is against point normal direction, then occupancy value is 1, otherwise it is 0. This way, we can sample $65,536$ points to fine-tune the network for each shape. Similar to the fine-tuning step of voxelized

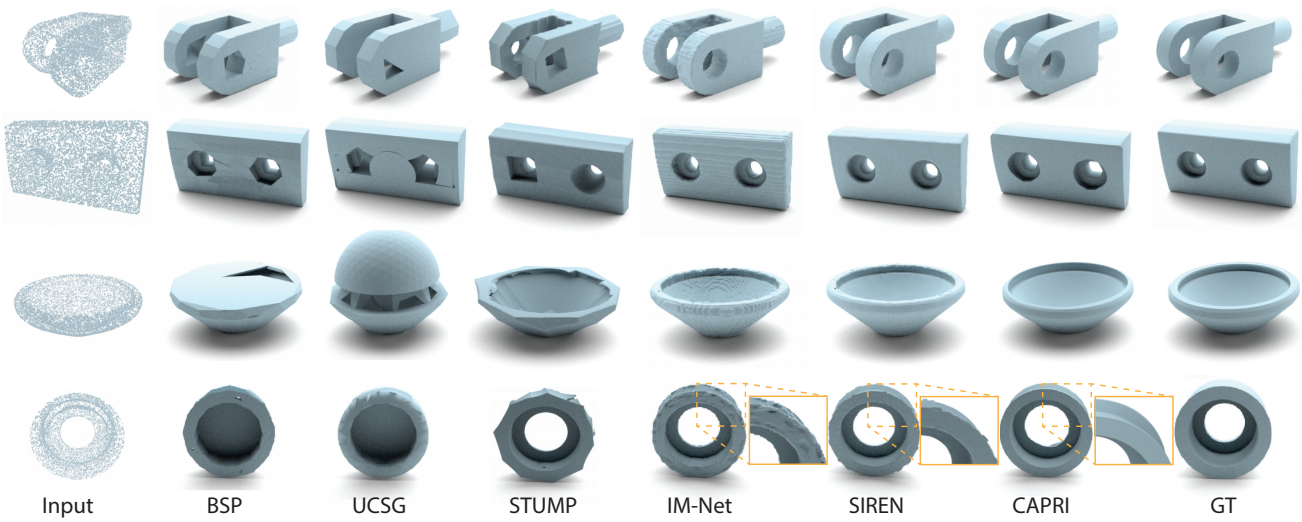

Figure 5. Visual comparisons between reconstruction results from point clouds (8,192 points) on ABC. Pay attention to the insets which show noticeable surface artifacts from IM-Net and SIREN results, both at 128^3 resolution.

Methods	CD ↓	NC ↑	ECD ↓	LFD ↓	#P ↓	#C ↓
UCSG	1.33	0.85	5.76	2,428.3	-	12.16
BSP-Net	0.47	0.89	9.65	920.8	153.80	14.45
STUMP	6.58	0.85	8.89	4,649.8	-	62.345
Ours	**0.14**	**0.92**	**1.57**	**581.8**	**64.06**	**6.82**

Table 4. Comparing 3D point cloud reconstruction on ABC.

inputs, we only fine-tune our latent codes, primitive prediction network and selection matrix.

Quantitative comparisons in Table 4 show that our network outperforms BSP-Net, UCSG, and CSG-Stump across the board. We provide additional ShapeNet comparison results in the supplementary material. With respect to state-of-the-art, *unstructured*, non-parametric-surface learning methods such as IM-Net [10] and SIREN [48], CAPRI-Net produces comparable metric performances, but is slightly worse, since there is an apparent trade-off between reconstruction quality and the desire to obtain a compact primitive assembly; see supplementary material for details. In terms of visual quality however, as shown in Figure 5, geometric artifacts such as small bumps and pits are often present on results from IM-Net and SIREN, while the mesh surfaces produced by CAPRI-Net, are more regularized.

5. Discussion, limitation, and future work

In recent years, there has been a large volume of works which target ShapeNet [4] for the development of neural 3D shape representations. We are not aware of similar efforts devoted to CAD models which possess *richer* geometric and topological variations, but lack strong structural predictability tied to a limited number of object categories. Our network, CAPRI-Net, fills this gap as it targets the ABC dataset [29] whose CAD models exhibit these very characteristics. As our results demonstrate, our reconstruction network outperforms state-of-the-art alternatives on both ABC *and* ShapeNet. Yet, CAPRI-Net only represents an early attempt at learning primitive assemblies for CAD models and beyond, since it is still limited on several fronts.

First, our network follows a *fixed* assembly order with intersection followed by union and then a single difference operation. As such, not all assemblies, e.g., a nested difference, can be represented. Second, despite the compactness exhibited by the recovered assemblies, CAPRI-Net does not have a network loss to enforce minimal CSG trees. Consistent with the minimum description length principle [22], devising such a loss could benefit many tasks beyond our problem domain. Third, limited quadric primitive types also affect reconstruction quality, so additional fine-tuning is needed to improve generalization over data with significant structure variations. However, our current fine-tuning does not yet work well on single-view images as input for the ABC CAD models. Last but not the least, our current approach does not take full advantage of the *local* regularity of CAD models due to their parametric nature.

In addition to addressing the above limitations, we would also like to extend CAPRI-Net into a fully generative model for CAD design. Conditioning the generator on design or hand-drawn sketches is also a promising direction considering its application potential. Finally, learning *functionality* of CAD models is also an intriguing topic to explore.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This work was supported in part by NSERC (No. 611370) and gift funds from Autodesk.

References

[1] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. *arXiv preprint arXiv:2011.13775*, 2020. 3

[2] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2011. 2

[3] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH*, pages 67–76, 2001. 2

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1, 6, 8

[5] Siddhartha Chaudhuri, Daniel Ritchie, Jiajun Wu, Kai Xu, and Hao Zhang. Learning generative models of 3d structures. *Computer Graphics Forum (Eurographics STAR)*, 2020. 3

[6] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, 2003. 7

[7] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020. 3

[8] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes vis binary space partitioning. In *CVPR*, 2020. 1, 2, 3, 5, 6, 7

[9] Zhiqin Chen, Kangxue Yin, Matthew Fisher, Siddhartha Chaudhuri, and Hao Zhang. BAE-NET: Branched autoencoder for shape co-segmentation. *ICCV*, 2019. 3

[10] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *CVPR*, 2019. 1, 2, 3, 6, 8

[11] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *CVPR*, 2020. 3

[12] Jianmin Zheng Daxuan Ren, Jianfei Cai, Haiyong Jiang Jiatong Li, Zhongang Cai, Junzhe Zhang, Liang Pan, Mingyuan Zhang, Haiyu Zhao, and Shuai Yi. CSG-Stump: A learning friendly csg-like representation for interpretable shape parsing. *ICCV*, 2021. 2, 3, 6

[13] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *CVPR*, 2020. 3

[14] Boyang Deng, JP Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Nasa: neural articulated shape approximation. *arXiv preprint arXiv:1912.03207*, 2019. 3

[15] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)*, 2018. 2

[16] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions. *arXiv preprint arXiv:2102.04776*, 2021. 3

[17] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981. 2

[18] Matthew Fisher, Manolis Savva, and Pat Hanrahan. Characterizing structural relationships in scenes using graph kernels. In *ACM SIGGRAPH 2011 papers*, 2011. 3

[19] Markus Friedrich, Pierre-Alain Fayolle, Thomas Gabor, and Claudia Linnhoff-Popien. Optimizing evolutionary csg tree extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019. 2

[20] Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. Sdm-net: Deep generative network for structured deformable mesh. *ACM Transactions on Graphics (TOG)*, 2019. 3

[21] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. In *CVPR*, 2018. 1

[22] Peter D. Grünwald. *The Minimum Description Length Principle*. 2007. 8

[23] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH*, pages 71–78, 1992. 2

[24] Paul VC Hough. Machine analysis of bubble chamber pictures. In *Proc. of the International Conference on High Energy Accelerators and Instrumentation, Sept. 1959*, 1959. 2

[25] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020. 7

[26] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *CVPR*, 2020. 3

[27] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. A survey of simple geometric primitives detection methods for captured 3d data. *Computer Graphics Forum*, 2019. 2

[28] Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. UCSG-Net–unsupervised discovering of constructive solid geometry tree. *arXiv preprint arXiv:2006.09102*, 2020. 2, 3, 6

[29] 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 *CVPR*, June 2019. 1, 6, 8

[30] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Trans. on Graphics (TOG)*, 2017. 1, 3

[31] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *CVPR*, 2019. 3

[32] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. Globfit: Consistently fitting primitives by discovering global relations. In *SIGGRAPH*, 2011. 2

[33] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. *arXiv preprint arXiv:2011.13084*, 2020. 3

[34] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision. *arXiv preprint arXiv:1911.00767*, 2019. 3

[35] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019. 1, 2, 3

[36] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*. Springer, 2020. 3

[37] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *SIGGRAPH Asia*, 2019. 1, 3

[38] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *CVPR*, 2020. 3

[39] Chengjie Niu, Jun Li, and Kai Xu. Im2struct: Recovering 3d shape structure from a single RGB image. In *CVPR*, 2018. 1

[40] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 1, 3

[41] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo-Martin Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020. 3

[42] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *CVPR*, 2019. 1, 3

[43] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. *arXiv preprint arXiv:2003.04618*, 2020. 3

[44] Tahir Rabbani and Frank Van Den Heuvel. Efficient hough transform for automatic detection of cylinders in point clouds. *Isprs Wg Iii/3, Iii/4*, 2005. 2

[45] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer graphics forum*, 2007. 2

[46] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: neural shape parser for constructive solid geometry. In *CVPR*, 2018. 3

[47] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *ECCV*, 2020. 3

[48] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020. 2, 3, 8

[49] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. *arXiv preprint arXiv:2011.12026*, 2020. 3

[50] Binil Starly and Yong Chen. "FabWave" - a pilot manufacturing cyberinfrastructure for shareable access to information rich product manufacturing data. https://www.dimelab.org/fabwave. 1

[51] Minhyuk Sung, Hao Su, Vladimir G Kim, Siddhartha Chaudhuri, and Leonidas Guibas. Complementme: Weakly-supervised component suggestions for 3d modeling. *ACM Transactions on Graphics (TOG)*, 2017. 3

[52] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020. 3

[53] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv preprint arXiv:1901.02875*, 2019. 3

[54] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017. 1, 3

[55] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017. 3

[56] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 2019. 3

[57] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad reconstruction, 2020. 1

[58] Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6772–6782, October 2021. 3

[59] Rundi Wu, Yixin Zhuang, Kai Xu, Hao Zhang, and Baoquan Chen. PQ-Net: A generative part seq2seq network for 3d shapes. In *CVPR*, 2020. 1, 3

[60] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1

[61] Zhijie Wu, Xiang Wang, Di Lin, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Sagnet: Structure-aware generative network for 3d-shape modeling. *ACM Transactions on Graphics (TOG)*, 2019. 3

[62] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. *arXiv preprint arXiv:2011.12950*, 2020. 3

[63] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomír Mech, and Ulrich Neumann. DISN: deep implicit surface network for high-quality single-view 3d reconstruction. *NeurIPS*, 2019. 1, 2

[64] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3D-PRNN: Generating shape primitives with recurrent neural networks. In *ICCV*, 2017. 1