Less is More: Learning Compact and General CSG Representations via Algebraic Normal Form

Kaichen Liu Fazhi He Rubin Fan Yuxin Liu Ruibo Wan Jianfei Liang
Yixiang Lei
School of Computer Science, Wuhan University
Wuhan, China
fzhe@whu.edu.cn

Abstract

Learning compact and general Constructive Solid Geometry (CSG) representations in a deep-learning-friendly manner remains a fundamental yet challenging problem. Existing methods often produce overly complex CSG expressions with heavy model architectures, whereas lightweight models tend to lack generality. To address this challenge, we propose ANF-CSG, an unsupervised neural framework based on the Algebraic Normal Form (ANF) of Boolean expressions. Firstly, we explicitly derive how standard CSG operations can be expressed in ANF-based form. Secondly, by leveraging only intersection and exclusive-or operations, ANF-CSG naturally produces CSG representations that are both structurally compact and provably general. This formulation enables the design of a shallow but still expressive neural architecture, substantially reducing model complexity while preserving representational power. Finally, extensive experiments demonstrate that ANF-CSG outperforms state-of-the-art methods. Our work provides a new perspective on neural CSG modeling by connecting Boolean algebra with deep learning.

***Keywords:** Constructive Solid Geometry (CSG), Algebraic Normal Form (ANF), Geometry Modeling and Processing, Deep Learning.*

1. Introduction

With the rapid advancement of deep learning, researchers have explored various forms of 3D shape representations, such as voxels [18, 32, 34, 28, 27], meshes [8, 10, 14, 19, 22], point clouds [33, 15, 35, 31], and Constructive Solid Geometry [30, 29, 12, 2, 5, 37, 36].

Among them, CSG stands out as an expressive representation, enabling the construction of complex geometries from basic primitives via Boolean operations (e.g., union, intersection, and difference). Consequently, learning effective representations of CSG models has been a long-standing and fundamental problem in computer vision, computer graphics, and geometry modeling and processing.

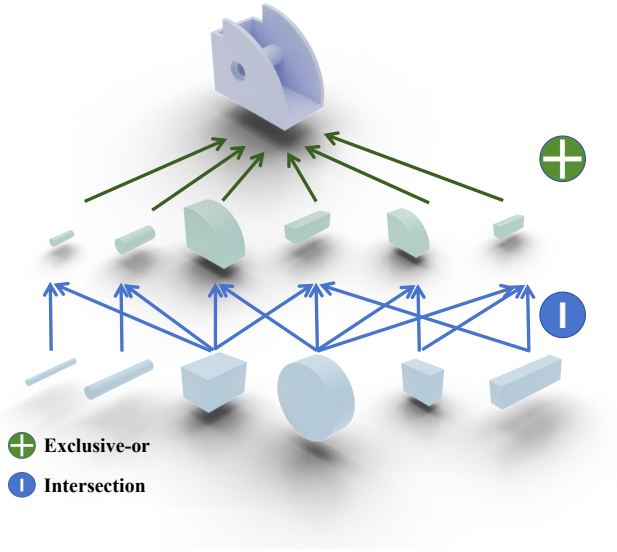

Figure 1. ANF-CSG composition of 3D shapes. Each shape is constructed using two types of operations: intersection and exclusive-or. This representation yields a concise two-layer structure suitable for deep learning.

In this work, we aim to learn *compact* and *general* CSG representations. By *compactness*, we mean structural simplicity realized by the fixed two-layer Boolean structure of ANF-CSG. This property arises from the formulation itself, rather than from optimization strategies or dataset characteristics. *Generality* refers to the capability to represent any valid CSG shape composed from the basic Boolean operations: union, intersection, and difference. In particular, representations without the difference operation cannot efficiently represent shapes with holes or cavities, limiting their practical generality. Such representations are crucial for downstream tasks, including model reconstruction, geometry editing, and shape generation. Achieving a balance

between compactness and generality remains an open challenge. Addressing this trade-off is both theoretically meaningful and practically valuable.

Existing approaches to learning Constructive Solid Geometry representations can be broadly categorized based on how the Boolean expression structures are organized. The first category focuses on variable CSG trees. For instance, CSGNet [30] represents shapes as hierarchical CSG trees and predicts expressions in a sequential manner, while UCSG-Net [12] introduces specialized CSG layers to generate interpretable shape programs. These tree-based approaches are flexible and can represent a wide variety of shapes. However, they often suffer from several practical limitations, including training instability, poor scalability to complex geometries, and structural redundancy arising from the dynamic nature of the generated programs. The second category adopts a fixed operation order to simplify expression generation. Methods such as BSP-Net [2], CVXNet [5], and CAPRI-Net [37] approximate shapes using predefined sets of primitives, such as hyperplanes or quadratic surfaces, and generate representations following a fixed sequence of operations. While these approaches benefit from architectural regularity, their outputs are often difficult to interpret or edit. CSG-Stump [25] and D^{2}CSG [36] are provably general under their respective designs. Although both guarantee that any valid CSG expression can be represented, their architectures remain relatively heavy.

Motivated by Occam's Razor—the principle that "entities should not be multiplied beyond necessity"—we adopt a "less is more" approach to directly learn CSG expressions from 3D shapes. Our goal is to obtain a representation that is both structurally compact and provably general. To this end, we introduce **ANF-CSG**, a lightweight and principled framework based on the *Algebraic Normal Form* from Boolean algebra (see Figure 1). Unlike prior methods, ANF-CSG represents shapes solely using intersection (AND) and exclusive-or (XOR) operations, organized in a concise two-layer hierarchy: primitives are first composed through intersection, then combined via exclusive-or. This formulation is both minimal in structure and provably general: any CSG expression can be equivalently converted to ANF.

A key property of ANF is its *uniqueness*: every Boolean function has a single canonical form under the ANF formulation [38]. This property removes equivalent or redundant logical structures, significantly simplifying the representation. Consequently, the network trains more efficiently and exhibits greater stability compared to prior CSG-based learning methods. Together, these properties make ANF-CSG a framework that is both structurally compact and general for learning CSG expressions.

In summary, the paper has the following contributions:

- We introduce **ANF-CSG**, a novel unsupervised framework that leverages the Algebraic Normal Form of Boolean expressions to represent arbitrary CSG programs in a compact and general manner.
- By leveraging the *uniqueness* property of ANF, our formulation can help reduce redundant equivalent programs and provide a deep-learning-friendly representation that requires only two layers of Boolean operations (intersection and exclusive-or).
- Extensive experiments show that ANF-CSG outperforms existing methods in reconstruction quality, produces more structurally compact representations, and enables faster training and fine-tuning.

2. Related Work

This section reviews two lines of work closely related to ours: general 3D shape representations used in computer vision and graphics, and recent approaches for CSG learning.

Shape Representation. In the field of computer vision and graphics, a variety of shape representations have been developed, each tailored to different applications and scenarios. Volumetric representations extend the concept of 2D pixels into 3D space and thus allow many image-based techniques to be readily adapted to 3D. However, the cubic growth of memory and computational cost with resolution imposes severe limits on the level of geometric detail they can capture [32, 34, 28]. Mesh representations, one of the most widely used formats in geometry modeling and processing, approximate surfaces using polygonal facets. They can effectively capture both the shape surface and the topological structure. Nonetheless, the non-uniformity of mesh poses significant challenges for learning-based methods [18, 8, 10, 14, 19, 22]. With the advancement of 3D scanning technologies, point cloud-based learning has become increasingly popular. Many recent methods employ multilayer perceptrons or convolutional architectures to process unordered point sets. While point clouds are simple and easy to acquire, they lack explicit topological structure and may fail to capture critical local geometry due to their sparse and irregular sampling [33, 9, 15, 35, 31].

More recently, implicit representations, such as Signed Distance Field or Occupancy, have gained traction due to their resolution-independence and compatibility with neural networks [3, 4, 11, 21, 24]. However, these methods typically require additional post-processing steps (e.g., Marching Cubes [20]) to extract surface meshes from the implicit field, adding complexity to the pipeline.

CSG Learning. Constructive Solid Geometry represents complex shapes by recursively applying Boolean operations over geometric primitives, forming a hierarchical tree structure. This symbolic representation is widely used in industrial design tools such as OpenSCAD [23] due to its compactness, editability, and expressive power. Recently,

there has been increasing interest in reconstructing shapes as CSG programs. CSG-Net [30] is an early effort that employs a neural encoder-decoder architecture to predict CSG trees from voxel inputs in a supervised manner.

Subsequent works have explored unsupervised CSG reconstruction by leveraging implicit representations to define reconstruction losses. For example, BAE-Net [3] introduces a branching structure, where each branch captures different shape parts, enabling unsupervised shape decomposition. BSP-Net [2] and CvxNet [5] leverage the insight that shapes can be decomposed into unions of convex regions, approximated via learned half-spaces. However, achieving high-fidelity reconstructions often requires a large number of hyperplanes, limiting their interpretability and editability. UCSG-Net [12] learns a dynamic order of Boolean operations via CSG layers, offering more flexibility but at the cost of training stability and expression compactness.

In pursuit of more general and structured CSG representations, later methods adopt alternative formulations of CSG trees. CSG-Stump [25] reformulates the CSG tree into a three-layer program structure and is provably general. However, it relies on a non-differentiable inverse layer, which may limit its ability to learn difference operations accurately. CAPRI-Net [37] extends BSP-Net by incorporating quadric surfaces as primitives and uses a fixed operation order. Although more expressive than planar-based methods, it cannot learn nested difference operations. This limits its generality and its ability to produce compact CSG programs. D^{2}CSG [36] is also provably general, using dual complementary branches to produce expressive CSG programs. From a Boolean perspective, its representation can be interpreted as composing two disjunctive normal forms followed by a global subtraction. While expressive, this design requires maintaining separate structures for additive and subtractive components, resulting in a relatively complex architecture. Meanwhile, ExtrudeNet [26] and SECAD-Net [17] focus on sketch-and-extrusion modeling, while SfmCAD [16] further incorporates extrusion, lofting, sweeping, and revolving operations. These methods offer intuitive control but compromise generality and Boolean expressiveness.

Despite steady progress, existing CSG learning frameworks still face a fundamental trade-off between compactness and generality. Our proposed **ANF-CSG** framework addresses this challenge by introducing a CSG formulation that is both structurally compact and provably general, relying solely on intersection and exclusive-or operations. Employing Boolean operations in the Algebraic Normal Form, our method generates interpretable CSG programs while preserving full representational power.

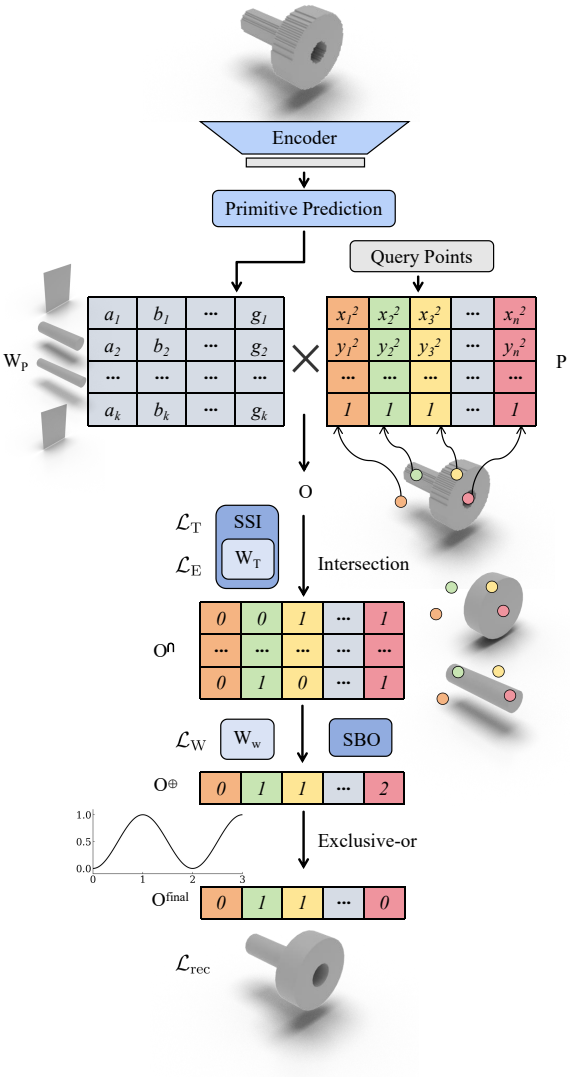

Figure 2. **Architecture of ANF-CSG.** The voxel input is first encoded into a latent vector, which predicts the shape primitives via the matrix $\mathbf{W}_\mathrm{P}$. Query points are then augmented to form $\mathbf{P}$, and multiplying $\mathbf{W}_\mathrm{P}$ with $\mathbf{P}$ produces the occupancy matrix $\mathbf{O}$. Intersections are computed by combining $\mathbf{O}$ with the SSI-enhanced matrix $\mathbf{W}_\mathrm{T}$, yielding $\mathbf{O}^\cap$ that represents intermediate shapes. The SBO module is then applied to $\mathbf{O}^\cap$, followed by the weighting with $\mathbf{W}_\mathrm{w}$, producing $\mathbf{O}^\oplus$, which stores the occupancy counts. Finally, a cosine-based parity mapping implements the XOR operation to obtain the final occupancy values.

3. The Proposed Method

This section describes the architecture of our **ANF-CSG**. Figure 2 shows an overview of the proposed framework. A discretized shape (*e.g.*, voxel representation) is first encoded into a latent code that captures its geometric structure. The latent code is then decoded by a *primitive prediction network* to predict the parameters of a set of geometric primi-

tives. For each primitive, the occupancy value of any query point can be computed to determine whether the point lies inside or outside the primitive.

The ANF connection matrix contains two submatrices corresponding to the intersection and exclusive-or layers. By sequentially combining the predicted occupancy values through these layers, the network produces the final occupancy representing the target shape.

3.1. Equivalence of ANF-CSG and CSG-Tree

Leveraging the classical Algebraic Normal Form representation from Boolean algebra, we show how Boolean expressions represented by standard CSG trees [12, 29, 30] can be systematically expressed in our two-layer ANF structure using only intersection and exclusive-or operations. This establishes the representational completeness of ANF-CSG with respect to standard CSG formulations.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ denote the set of primitive shapes. The Boolean operations considered are union ($\cup$), intersection ($\cap$), and difference ($\setminus$).

The goal is to prove that any CSG shape constructed from primitives $\mathcal{P}$ and Boolean operations $\{\cup, \cap, \setminus\}$ can be equivalently represented by an ANF-CSG expression, i.e., a finite XOR ($\oplus$) of intersections of primitives.

Base Case: For $k = 1$ with a single primitive p_1, the shape trivially admits an ANF-CSG representation as p_1, which directly corresponds to a base ANF term.

For two primitives p_1, p_2, the Boolean operations are expressible in ANF-CSG form as follows:

$$p_1 \cup p_2 = p_1 \oplus p_2 \oplus (p_1 \cap p_2), \tag{1}$$
$$p_1 \cap p_2 = p_1 \cap p_2, \tag{2}$$
$$p_1 \setminus p_2 = p_1 \oplus (p_1 \cap p_2), \tag{3}$$

Where all right-hand side expressions are either XORs or intersections of primitives, conforming to ANF-CSG.

Inductive Hypothesis: Assume that any CSG shape constructed from at most n primitives can be equivalently represented by an ANF-CSG expression.

Inductive Step: Consider an arbitrary CSG shape β composed from $n + 1$ primitives. The root Boolean operation splits β into two sub-shapes:

$$\beta = \beta_1 \odot \beta_2, \tag{4}$$

where $\odot \in \{\cup, \cap, \setminus\}$ and each β_i is a CSG shape formed from at most n primitives.

By the inductive hypothesis, β_1 and β_2 admit ANF-CSG representations:

$$\beta_1 = \bigoplus_{i=1}^{m} A_i, \quad \beta_2 = \bigoplus_{j=1}^{l} B_j, \tag{5}$$

where each A_i and B_j is an intersection of primitives.

We analyze each Boolean operation:

- Intersection:

  Recall that the XOR operation can be expressed as

  $$A \oplus B = (A \setminus B) \cup (B \setminus A).$$

  Then, for any set X, we have

  $$\begin{aligned} X \cap (A \oplus B) &= X \cap \big((A \setminus B) \cup (B \setminus A)\big) \\ &= \big(X \cap (A \setminus B)\big) \cup \big(X \cap (B \setminus A)\big) \\ &= (X \cap A) \oplus (X \cap B). \end{aligned} \tag{6}$$

  This derivation shows that intersection distributes over XOR. Consequently, for two ANF-CSG expressions $\beta_1 = \bigoplus_{i=1}^{m} A_i$ and $\beta_2 = \bigoplus_{j=1}^{l} B_j$, we obtain

  $$\beta_1 \cap \beta_2 = \bigoplus_{i=1}^{m} \bigoplus_{j=1}^{l} (A_i \cap B_j), \tag{7}$$

  which remains in ANF-CSG form.

- Union: Using the identity

  $$\beta_1 \cup \beta_2 = \beta_1 \oplus \beta_2 \oplus (\beta_1 \cap \beta_2), \tag{8}$$

  and noting that

  $$\beta_1 \cap \beta_2 = \bigoplus_{i=1}^{m} \bigoplus_{j=1}^{l} (A_i \cap B_j), \tag{9}$$

  The union β is expressible as a finite XOR of intersections of primitives.

- Difference: By definition,

  $$\beta_1 \setminus \beta_2 = \beta_1 \cap \overline{\beta_2}. \tag{10}$$

  The complement can be expressed as

  $$\overline{\beta_2} = \mathbb{U} \oplus \beta_2, \tag{11}$$

  where $\mathbb{U}$ denotes the universal shape. Hence,

  $$\begin{aligned} \beta_1 \setminus \beta_2 &= \beta_1 \cap \overline{\beta_2} \\ &= \beta_1 \cap (\mathbb{U} \oplus \beta_2) \\ &= (\beta_1 \cap \mathbb{U}) \oplus (\beta_1 \cap \beta_2) \\ &= \beta_1 \oplus (\beta_1 \cap \beta_2). \end{aligned} \tag{12}$$

  Since β_1 and $\beta_1 \cap \beta_2$ are both representable in ANF-CSG form, the difference is also representable.

Conclusion: Any CSG shape composed through Boolean operations over primitives can be equivalently formulated in the ANF-CSG representation.

The proposed ANF-CSG formulation exhibits several theoretical properties. First, it is *closed* under Boolean operations: applying union, intersection, or difference to ANF-expressible primitives produces a result that remains representable in ANF form. Second, ANF-CSG is *provably general*: it possesses the same expressive power as standard CSG trees, guaranteeing no loss of modeling capability when adopting ANF-based structures.

3.2. Primitive Prediction and Occupancy Representation

The primitive prediction network maps a latent vector to a set of parametric primitives. Each primitive is represented as a quadratic surface [37], which can model common convex shapes such as spheres, planes, cylinders, and cones. Formally, each primitive satisfies the quadratic equation

$$ax^2 + by^2 + cz^2 + dx + ey + fz + g = 0, \tag{13}$$

where the seven parameters (a, b, c, d, e, f, g) are stored in the primitive parameter matrix $W_{\text{P}} \in \mathbb{R}^{K\times 7}$, with K denoting the total number of primitives.

To obtain occupancy values [21, 24, 4] for a set of query points $\mathbf{P} = \{\mathbf{p}_i\}_{i=1}^{N}$, where each point $\mathbf{p}_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, we first augment them into an extended representation. For each point, we compute a feature vector $\tilde{\mathbf{p}}_i \in \mathbb{R}^7$:

$$\tilde{\mathbf{p}}_i = \left[x_i^2,\ y_i^2,\ z_i^2,\ x_i,\ y_i,\ z_i,\ 1\right]^\top. \tag{14}$$

The augmented points are converted into primitive parameters through matrix multiplication with $\mathbf{W}_{\text{P}}$:

$$O_i = \max\left(0, \mathbf{W}_{\text{P}}\,\tilde{\mathbf{p}}_i\right), \tag{15}$$

where $O_i \in \mathbb{R}^{K\times 1}$ encodes the occupancy of the i-th query point with respect to all K primitives. Here, $O_{i,j} = 0$ indicates that the i-th point lies inside the j-th primitive p_j, while $O_{i,j} > 0$ indicates that it lies outside.

3.3. ANF connection matrix

The ANF connection matrix encodes the ANF-CSG structure with two submatrices, reflecting its compact two-layer structure. The first submatrix corresponds to the *intersection* layer, selecting geometric primitives for each intersection node. The second submatrix represents the *exclusive-or* layer, aggregating intersection results via XOR operations to produce the final shape. Let C denote the number of intersection nodes, and K the total number of primitives. The first submatrix has size $C \times K$, while the second is a row vector of size $1 \times C$.

- **Intersection layer**: uses a matrix $\mathbf{W}_{\text{T}} \in \{0,1\}^{C\times K}$, where the i-th intersection node includes primitive j if $\mathbf{W}_{\text{T}}[i, j] = 1$.
- **Exclusive-Or layer**: employs a vector $\mathbf{W}_{\text{W}} \in \{0,1\}^{C}$, with $\mathbf{W}_{\text{W}}[i] = 1$ indicating inclusion of the i-th intersection node in the XOR aggregation for the final shape.

The two connection matrices are applied to the occupancy matrix $\mathbf{O}$ to compute the final shape occupancy. Specifically, the intersection is computed as

$$\mathbf{O}^{\cap} = \text{AND}(\mathbf{W}_{\text{T}}\,\mathbf{O}), \tag{16}$$

and the occupancy counts are obtained via

$$\mathbf{O}^{\oplus} = \mathbf{W}_{\text{W}}\,\mathbf{O}^{\cap}. \tag{17}$$

The final occupancy values are then computed by

$$\mathbf{O}^{\text{final}} = \text{XOR}(\mathbf{O}^{\oplus}). \tag{18}$$

Thus, the final shape occupancy is fully determined by the K primitives and the connection matrices, which transform the occupancy matrix $\mathbf{O}$ into an ANF expression with a compact two-layer structure. Compared to traditional CSG trees, which are often deep, irregular, and difficult to optimize, ANF-CSG is more structurally compact, regular, interpretable, and compatible with deep learning.

In Boolean algebra, converting an expression into ANF can sometimes increase the number of intermediate intersection terms. In ANF-CSG, this potential growth is controlled by the network design. The number of candidate intersections is limited by the size of the intersection layer and by the number of primitives. This prevents uncontrolled combinatorial expansion. In addition, the training process naturally encourages sparsity. During training, only a small number of intersection terms actually contribute to the final shape. The others become less important over time, as the network gradually reduces their weights. Therefore, although ANF may lead to more intermediate intersection terms in theory, in our neural framework the number of active terms remains limited and stable in practice.

3.4. Differentiable ANF-CSG for Occupancy Prediction

We now describe how we train ANF-CSG in a fully differentiable manner, focusing on the *intersection* and *exclusive-or* operations.

To simulate binary matrices while preserving differentiability, the intersection weights are first transformed via a *Soft Set Indicator (SSI)* module:

$$\hat{\mathbf{W}}_{\text{T}} = \sigma\Big((\mathbf{W}_{\text{T}} - 0.5)\cdot\alpha\Big), \tag{19}$$

where $\mathbf{W}_{\text{T}}$ are the learnable intersection weights, $\sigma(\cdot)$ is the sigmoid function, and α controls the sharpness of the approximation.

Next, the transformed intersection matrix is applied to the occupancy matrix $\mathbf{O}$ to simulate the intersection operation. For a query point p_i, the resulting soft occupancy after intersection is

$$\mathbf{O}_i^{\cap} = \hat{\mathbf{W}}_{\mathrm{T}}\, \mathbf{O}_i, \tag{20}$$

where $\mathbf{O}_i \in \mathbb{R}^{K\times 1}$ encodes the occupancy of p_i with respect to all K primitives, and $\hat{\mathbf{W}}_{\mathrm{T}} \in \mathbb{R}^{C\times K}$ selects and weights the primitives involved in the intersection. Intuitively, since 0 represents a point lying inside a primitive, a query point lies inside the intersection of selected primitives only if all corresponding occupancy values are 0. This implements the intersection operation implicitly.

To implement the exclusive-or operation in ANF-CSG more effectively, we introduce a second sigmoid function, called the *Soft Boolean Operator (SBO)*:

$$\tilde{\mathbf{O}}_i = \sigma\Big((0.5 - \mathbf{O}_i^{\cap}) \cdot \beta\Big), \tag{21}$$

where β controls the sharpness. This transformation inverts the inside–outside convention, assigning values close to 1 to points inside the region and values close to 0 to those outside.

The final XOR aggregation is computed using learnable weights $\mathbf{W}_{\mathrm{W}}$:

$$\mathbf{O}_i^{\oplus} = \tilde{\mathbf{O}}_i\, \mathbf{W}_{\mathrm{W}}, \tag{22}$$

where $\mathbf{O}_i^{\oplus}$ denotes the sum occupancy of query point p_i, providing a soft count of active primitives.

Conceptually, the XOR operation in ANF-CSG can be understood as a parity check over occupancy values: a query point is assigned 1 if it is covered by an odd number of shapes, and 0 if covered by an even number. This behavior follows naturally from Boolean algebra, where XOR corresponds to a modulo-2 summation of binary inputs. To illustrate this process, we provide a 2D schematic in Figure 3. In CSG modeling, overlapping contributions from primitives may either cancel or accumulate, forming the final occupancy pattern. This mechanism allows ANF-CSG to represent complex Boolean compositions using only two layers of operations.

To convert this soft count into a differentiable occupancy value, we apply a cosine-based parity mapping:

$$\mathbf{O}_i^{\text{final}} = \frac{1}{2}\left(1 - \cos(\pi \mathbf{O}_i^{\oplus})\right), \tag{23}$$

which ensures that points corresponding to odd soft counts yield occupancies close to 1 (occupied), while points with even counts yield values close to 0 (empty).

This cosine-based formulation offers a smooth and fully differentiable approximation of the discrete parity function. Unlike a hard XOR that changes abruptly at binary thresholds, the cosine mapping varies continuously with respect to $\mathbf{O}_i^{\oplus}$. Thus, small changes in intermediate soft values lead to gradual occupancy changes rather than sudden parity flips. This behavior is crucial during early training. The continuous formulation prevents oscillations in parity decisions and lets the network explore solutions without prematurely fixing discrete structures. As training advances, outputs gradually approach binary values, making $\mathbf{O}_i^{\oplus}$ close to integers. Eventually, the cosine mapping converges to true binary parity, enabling accurate recovery of discrete CSG programs while preserving optimization stability.

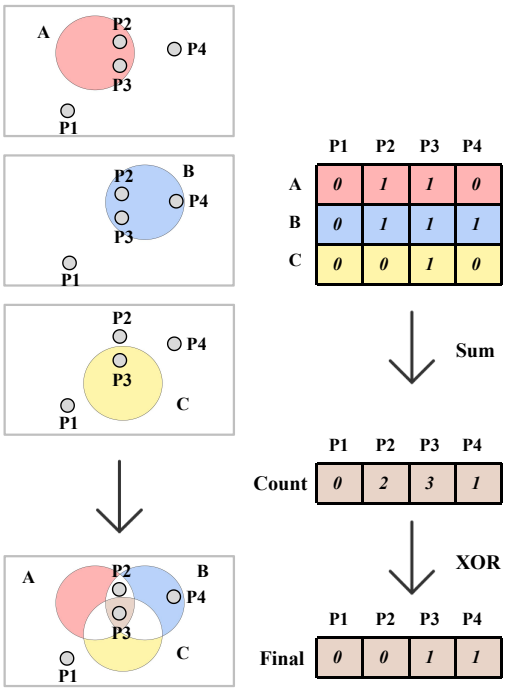

Figure 3. **XOR Computation in ANF-CSG.** Query points are sampled at corresponding locations across multiple primitive shapes. The occupancy of each point within the individual primitives is recorded. These occupancies are then summed to obtain an intermediate count. A parity check is applied to each point to produce the final occupancy values.

3.5. Multi-Stage Training

Stage 0. In this stage, we employ fully differentiable operations, since directly learning a binary selection matrix is challenging. The matrix $\mathbf{W}_{\mathrm{T}}$ is not strictly binarized, and a cosine-based parity mapping is used to approximate a differentiable XOR operation, which allows gradient-based optimization to propagate through logical combinations. Both the SSI and SBO modules are enabled, guiding the network to gradually produce near-binary outputs.

To further regularize the program structure, similar to BSP-Net [2], we introduce constraints on $\mathbf{W}_{\mathrm{T}}$ and $\mathbf{W}_{\mathrm{W}}$. Specifically, elements of $\mathbf{W}_{\mathrm{T}}$ are penalized if they fall out-

side the valid range $[0, 1]$:

$$\mathcal{L}_{\mathrm{T}} = \sum_i \max(0, (\mathbf{W}_{\mathrm{T}})_i - 1) + \sum_i \max(0, -(\mathbf{W}_{\mathrm{T}})_i), \tag{24}$$

elements of $\mathbf{W}_{\mathrm{W}}$ are encouraged to approach 1:

$$\mathcal{L}_{\mathrm{W}} = \sum_i |(\mathbf{W}_{\mathrm{W}})_i - 1|, \tag{25}$$

and to gradually transition the network from fully differentiable to binary-like outputs, an entropy penalty is applied on $\mathbf{W}_{\mathrm{T}}$:

$$\mathcal{L}_{\mathrm{E}} = -\frac{1}{N} \sum_i \Big[(\mathbf{W}_{\mathrm{T}})_i \log(\mathbf{W}_{\mathrm{T}})_i + (1 - (\mathbf{W}_{\mathrm{T}})_i) \log(1 - (\mathbf{W}_{\mathrm{T}})_i) \Big]. \tag{26}$$

This entropy penalty measures the uncertainty of each element in $\mathbf{W}_{\mathrm{T}}$, assigning low values to near-deterministic outputs and high values to uncertain outputs. Minimizing $\mathcal{L}_{\mathrm{E}}$ thus encourages binary-like outputs while maintaining differentiability for gradient-based optimization.

For each point, the predicted occupancy values $\mathbf{O}^{\text{final}}$ are compared against the corresponding ground-truth occupancies $\mathbf{O}^* \in \{0, 1\}^N$. The reconstruction loss is formulated as the mean squared error:

$$\mathcal{L}_{\mathrm{rec}} = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{O}_i^{\text{final}} - \mathbf{O}_i^* \right)^2. \tag{27}$$

The overall Stage 0 loss is therefore

$$\mathcal{L}_{\mathrm{stage0}} = \mathcal{L}_{\mathrm{rec}} + \mathcal{L}_{\mathrm{T}} + \lambda_{\mathrm{W}} \mathcal{L}_{\mathbf{W}} + \lambda_{\mathrm{E}} \mathcal{L}_{\mathrm{E}}, \tag{28}$$

where λ_{W} and λ_{E} are weighting coefficients for the regularization of $\mathbf{W}_{\mathrm{W}}$ and the entropy penalty on $\mathbf{W}_{\mathrm{T}}$, respectively. In our experiments, we set $\lambda_{\mathrm{W}} = 0.5$ and $\lambda_{\mathrm{E}} = 1.5$.

Stage 1. In this stage, training gradually drives the network outputs toward binary values. This stage primarily serves as a fine-tuning phase. The SSI module is disabled, since the selection matrix has already approached near-binary values after Stage 0.

The loss function in Stage 1 is formulated as:

$$\mathcal{L}_{\mathrm{stage1}} = \mathcal{L}_{\mathrm{rec}} + \mathcal{L}_{\mathrm{T}} + \lambda_{\mathrm{E}} \mathcal{L}_{\mathrm{E}}, \tag{29}$$

where $\lambda_{\mathrm{E}} = 5$ is used to accelerate the binarization of network outputs.

Stage 2. Stage 2 serves as the final fine-tuning phase, during which the model is required to produce fully interpretable CSG-like representations. Unlike previous stages, the selection matrix $\mathbf{W}_{\mathrm{T}}$ is binarized using a hard threshold:

$$\mathbf{W}_{\mathrm{T}} = (\mathbf{W}_{\mathrm{T}} > 0.5), \tag{30}$$

ensuring that the resulting representations correspond to standard CSG operations.

In previous stages, a cosine-based parity mapping was used to approximate a differentiable XOR operation. Since $\mathbf{W}_{\mathrm{T}}$ is now binary, the final soft counts naturally become hard counts, effectively implementing a binary XOR operation.

The loss function in Stage 2 is defined as:

$$\mathcal{L}_{\mathrm{stage2}} = \mathcal{L}_{\mathrm{rec}} + \lambda_{\mathrm{E}} \mathcal{L}_{\mathrm{E}}, \tag{31}$$

where $\lambda_{\mathrm{E}} = 25$ is used to enforce strict binarization of network outputs.

4. Experiments

In this section, we present both qualitative and quantitative evaluations of the proposed ANF-CSG model. Experiments are conducted on the ABC [13] and ShapeNet [1] datasets to assess our model's performance in reconstructing shapes from voxels. We compare our method with several baseline approaches that use CSG-like representations. We also conduct ablation studies to evaluate the impact of the Soft Set Indicator and Soft Boolean Operator modules in our network.

4.1. Setup

Dataset. We train and evaluate ANF-CSG on two datasets: ShapeNet and ABC. ShapeNet involves relatively few difference operations, while ABC contains more complex geometries that require more differences. For each model, we voxelize the shape into a 64^3 grid as input. Following IM-Net [4], we sample 24,576 points on each shape's surface as query points and obtain their occupancy values. For ShapeNet, we use ten categories for training, with 100 models per category for testing. For ABC, we use the dataset from CAPRI-Net [37], which includes 5,000 training models and 1,000 test models. For evaluation, we randomly select a subset of 100 models from the test set.

Implementation details. Our ANF-CSG model is implemented in PyTorch and optimized using the Adam optimizer with a learning rate of 10^{-4}. We train our model on an NVIDIA TITAN RTX GPU using a batch size of 128. Stage 0 of ANF-CSG is run for 1,000 epochs. During evaluation, each shape is further fine-tuned with 100 iterations per stage. To ensure fairness, the baseline methods are trained for 1,000 iterations, followed by an additional 300 iterations of fine-tuning.

Importantly, both training and fine-tuning of ANF-CSG are faster compared to baseline methods. For instance, CAPRI-Net requires approximately 9 hours for training, whereas ANF-CSG completes training in about 6 hours. During fine-tuning, CAPRI-Net takes roughly 2 minutes per shape, while ANF-CSG requires only 1.5 minutes.

category	CD ↓				NC ↑				ECD ↓			
	Ours	CAPRI	STUMP	UCSG	Ours	CAPRI	STUMP	UCSG	Ours	CAPRI	STUMP	UCSG
plane	**0.566**	0.651	0.992	0.649	**0.836**	0.788	0.802	0.792	**2.44**	3.16	3.16	3.24
bench	**0.841**	1.470	3.961	2.502	**0.790**	0.768	0.786	0.777	**1.70**	2.55	5.07	4.29
cabinet	**1.000**	1.001	2.082	1.648	**0.856**	0.856	0.854	0.829	**2.46**	2.64	3.53	4.40
car	0.647	0.736	**0.487**	0.723	0.834	0.837	**0.843**	0.841	2.62	2.87	**2.01**	3.13
display	0.843	**0.716**	1.198	2.143	**0.891**	0.882	0.866	0.866	3.66	**3.31**	3.79	5.83
rifle	**0.339**	0.454	0.534	0.885	**0.780**	0.735	0.745	0.704	**0.61**	0.98	0.89	1.68
couch	0.879	0.777	**0.647**	1.629	0.851	0.874	**0.879**	0.833	2.23	1.97	**1.58**	3.40
table	**1.185**	1.962	3.363	2.752	**0.845**	0.820	0.808	0.806	**2.06**	3.71	5.05	4.90
phone	0.284	**0.274**	1.030	0.732	**0.935**	0.935	0.923	0.929	**2.06**	2.13	3.12	2.99
vessel	**0.582**	0.593	1.102	1.184	**0.828**	0.810	0.810	0.803	**2.21**	2.67	2.98	3.53
mean	**0.717**	0.863	1.540	1.485	**0.844**	0.831	0.832	0.818	**2.20**	2.60	3.12	3.74

Table 1. Per-category evaluation on ShapeNet. For clearer comparison, both CD and ECD values are scaled by a factor of 10^3.

Evaluation metrics. We evaluate reconstruction quality using three metrics: Chamfer Distance (CD), Normal Consistency (NC), and Edge Chamfer Distance [2] (ECD). CD measures the bidirectional distance between predicted and ground-truth surfaces, reflecting overall shape accuracy. NC assesses the alignment of surface normals, capturing local smoothness and orientation consistency. ECD evaluates the preservation of sharp geometric features by quantifying the distance between edge regions of the predicted and target shapes.

Reconstructed meshes are generated using the Marching Cubes algorithm at a resolution of 64^3. To detect edge regions for ECD [2], the original method measures point sharpness within a fixed-radius neighborhood ϵ. In our implementation, we use a k-nearest neighbors (kNN) search to accelerate processing on large point clouds. Edge points are detected using a normal cross-product threshold of 0.1 on the ground-truth surface and 0.5 on the reconstructed meshes. The higher threshold on reconstructed meshes compensates for the smoothing introduced by the Marching Cubes algorithm [16]. Although this kNN-based neighborhood slightly differs from the original definition, we observe that the detected edge points and ECD values remain largely consistent while significantly reducing computational overhead.

4.2. Comparisons

We compare our method with existing approaches that generate CSG-like representations and are designed to be compatible with deep learning. Specifically, we evaluate UCSG-Net [12], CSG-Stump [25], and CAPRI-Net [37] on both the ShapeNet and ABC datasets.

Results on ShapeNet. Table 1 presents the per-category evaluation of our ANF-CSG method against state-of-the-art baselines on ShapeNet. CAPRI-Net remains one of the strongest existing approaches. In contrast, UCSG-Net and CSG-Stump lack support for difference operations between complex primitives, limiting their ability to capture geometric details.

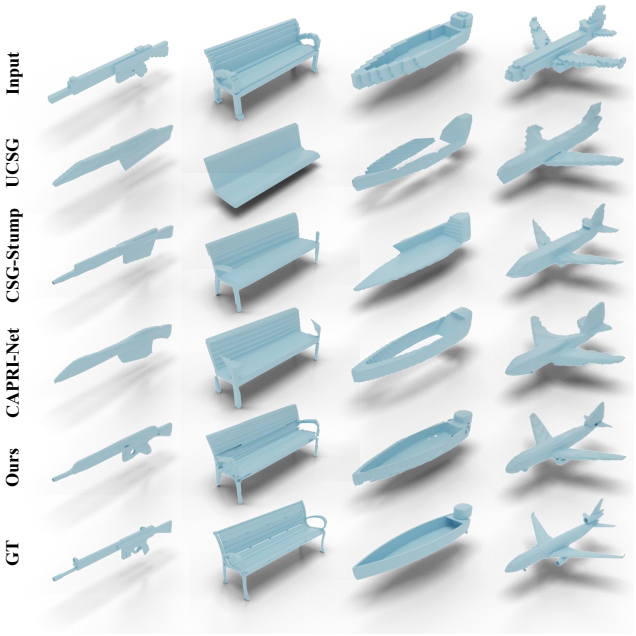

Figure 4. Reconstruction results from voxels on ShapeNet.

Overall, ANF-CSG consistently outperforms CAPRI-Net across most categories. Specifically, it reduces the average Chamfer Distance by approximately 16.9% compared to CAPRI-Net. For Normal Consistency, ANF-CSG slightly outperforms the other methods. Notably, for Edge Chamfer Distance, which reflects the quality of edge reconstruction, our method yields a 15.4% improvement over CAPRI-Net.

The most significant Chamfer Distance improvements are observed in the *bench* and *table* categories, where ANF-CSG reduces CD by 42.8% and 39.6% compared to CAPRI-Net, respectively. For the Edge Chamfer Distance, the

Method	CD↓	NC↑	ECD↓
Ours	**0.486**	**0.883**	**5.88**
CAPRI	0.580	0.880	7.18
STUMP	0.938	0.870	6.54
UCSG	1.268	0.861	10.73

Table 2. Comparison of ANF-CSG with existing methods on ABC.

improvement is particularly notable in the *table* category, showing a 44.5% reduction relative to CAPRI-Net.

CSG-Stump achieves slightly better CD values in the *car* and *couch* categories, as these shapes can be well approximated using only a few primitives and involve limited difference operations. However, CSG-Stump struggles to reconstruct shapes containing small holes or concavities. ANF-CSG maintains robust performance across all categories, benefiting from its structurally compact and expressive shape representation.

Figure 4 provides qualitative comparisons of reconstructed shapes on ShapeNet. ANF-CSG exhibits higher sensitivity to geometric details such as drilling and slotting, producing cleaner surfaces with sharper boundaries. In contrast, UCSG-Net and CSG-Stump often fail to capture these features accurately. CAPRI-Net struggles with small holes and tends to overcut during slotting, likely due to its reliance on a prominent difference operation. An inaccurate difference can distort the geometry.

Results on ABC. The ABC dataset presents a more challenging modeling scenario, characterized by frequent Boolean difference operations and intricate B-rep geometries.

Table 2 summarizes the quantitative results. ANF-CSG achieves the best performance across all three metrics. It reduces the Chamfer Distance by 16.2% relative to CAPRI-Net and slightly improves Normal Consistency. For Edge Chamfer Distance, our method yields an 18.1% improvement over CAPRI-Net.

Figure 5 shows representative examples of shape reconstructions on the ABC dataset. While preserving the overall geometry of the models, ANF-CSG better captures drilling operations compared to other methods. These results demonstrate that ANF-CSG can effectively handle complex Boolean compositions.

4.3. Ablation Study

We perform ablation studies to evaluate the contributions of the Soft Set Indicator and Soft Boolean Operator modules by selectively removing them from ANF-CSG. The quantitative results are presented in Table 3.

Removing either module results in a significant increase in reconstruction error. Specifically, without SSI, the Chamfer Distance increases by approximately 10.9%, and without SBO, CD increases by approximately 102.8%. When both modules are removed, CD increases by 264.3%, indicating the worst performance. Moreover, the Normal Consistency metric slightly improves when both modules are removed, suggesting that the network generates smoother surfaces at the cost of quality.

These findings confirm that both SSI and SBO are crucial for achieving high-fidelity and detailed shape reconstruction in ANF-CSG.

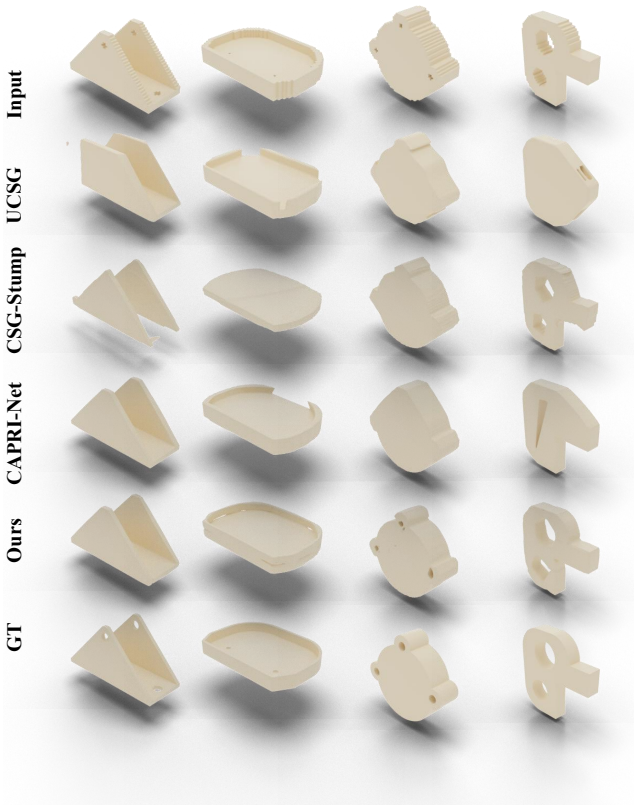

Figure 5. Reconstruction results from voxels on ABC.

Settings	CD↓	NC↑	ECD↓
Ours (full model)	**2.898**	0.747	**11.02**
Ours w/o SSI	3.213	0.726	12.20
Ours w/o SBO	5.878	0.732	13.51
Ours w/o SSI&SBO	10.558	**0.753**	17.53

Table 3. Ablation study on ANF-CSG.

5. Conclusion

We propose **ANF-CSG**, a novel and lightweight framework for learning Constructive Solid Geometry expressions directly from 3D shapes. This framework integrates Boolean algebra with deep learning, opening new avenues for interpretable and efficient shape modeling.

The proposed ANF-CSG leverages the Algebraic Normal Form to represent shapes in a fixed two-layer hierarchy. Our analysis shows that standard CSG expressions can be systematically transformed into the ANF-based form, establishing the representational completeness of our formulation with respect to conventional CSG. Specifically, the first layer assembles primitive shapes through intersections, while the second layer combines these parts using exclusive-or operations. This design simplifies the structure of CSG programs while preserving full representational power. As a result, ANF-CSG achieves a structurally compact and provably general shape representation.

We validate ANF-CSG on both the ShapeNet and ABC datasets through extensive experiments. Quantitative and qualitative results show that our method consistently achieves higher reconstruction accuracy and more precise modeling of features such as holes and concavities compared to state-of-the-art baselines. The ablation study further demonstrates that both the Soft Set Indicator and Soft Boolean Operator modules significantly contribute to reconstruction quality. Moreover, ANF-CSG achieves faster training and fine-tuning, highlighting its practical advantages for shape modeling. These results suggest that ANF-CSG provides a principled and efficient solution for learning interpretable CSG programs. In the near future, we will advance the idea to industry CAD areas [6, 7].

Acknowledgement

The numerical calculations in this paper were performed on the super-computing system at the Supercomputing Center of Wuhan University, China.

References

[1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 7

[2] Z. Chen, A. Tagliasacchi, and H. Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 45–54, 2020. 1, 2, 3, 6, 8

[3] Z. Chen, K. Yin, M. Fisher, S. Chaudhuri, and H. Zhang. Bae-net: Branched autoencoder for shape co-segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8490–8499, 2019. 2, 3

[4] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5939–5948, 2019. 2, 5, 7

[5] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 31–44, 2020. 1, 2, 3

[6] R. Fan, F. He, Y. Liu, and J. Lin. A history-based parametric cad sketch dataset with advanced engineering commands. *Computer-Aided Design*, 182:103848, 2025. 10

[7] R. Fan, F. He, Y. Liu, Y. Song, L. Fan, and X. Yan. A parametric and feature-based cad dataset to support human-computer interaction for advanced 3d shape learning. *Integrated Computer-Aided Engineering*, 32(1):75–96, 2025. 10

[8] Y. Feng, Y. Feng, H. You, X. Zhao, and Y. Gao. Meshnet: Mesh neural network for 3d shape representation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 8279–8286, 2019. 1, 2

[9] S. Fung, W. Pan, X. Liu, J. Yearwood, R. Dazeley, and X. Lu. Topformer: topology-aware transformer for point cloud registration. In *International Conference on Computational Visual Media*, pages 112–128. Springer, 2024. 2

[10] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (ToG)*, 38(4):1–12, 2019. 1, 2

[11] Z. Hao, H. Averbuch-Elor, N. Snavely, and S. Belongie. Dualsdf: Semantic shape manipulation using a two-level representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7631–7641, 2020. 2

[12] K. Kania, M. Zieba, and T. Kajdanowicz. Ucsg-net-unsupervised discovering of constructive solid geometry tree. *Advances in neural information processing systems*, 33:8776–8786, 2020. 1, 2, 3, 4, 8

[13] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9601–9611, 2019. 7

[14] A. Lahav and A. Tal. Meshwalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics (TOG)*, 39(6):1–13, 2020. 1, 2

[15] E.-T. Lê, M. Sung, D. Ceylan, R. Mech, T. Boubekeur, and N. J. Mitra. Cpfn: Cascaded primitive fitting networks for high-resolution point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7457–7466, 2021. 1, 2

[16] P. Li, J. Guo, H. Li, B. Benes, and D.-M. Yan. Sfmcad: Unsupervised cad reconstruction by learning sketch-based feature modeling operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4671–4680, 2024. 3, 8

[17] P. Li, J. Guo, X. Zhang, and D.-M. Yan. Secad-net: Self-supervised cad reconstruction by learning sketch-extrude operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16816–16826, 2023. 3

[18] Y. Li, M. Tang, Y. Yang, R. Tong, S. Yang, Y. Li, B. An, and Q. Kou. Ctsn: Predicting cloth deformation for skeleton-based characters with a two-stream skinning network. *Computational Visual Media*, 10(3):471–485, 2024. 1, 2

[19] Z. Liu, Y. Wang, X. Qi, and C.-W. Fu. Towards implicit text-guided 3d shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17896–17906, 2022. 1, 2

[20] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, pages 163–169, 1987. 2

[21] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 2, 5

[22] C. Nash, Y. Ganin, S. A. Eslami, and P. Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International conference on machine learning*, pages 7220–7229. PMLR, 2020. 1, 2

[23] OpenSCAD. OpenSCAD: The Programmers Solid 3D CAD Modeller. http://www.openscad.org/, 2025. 2

[24] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. 2, 5

[25] D. Ren, J. Zheng, J. Cai, J. Li, H. Jiang, Z. Cai, J. Zhang, L. Pan, M. Zhang, H. Zhao, et al. Csg-stump: A learning friendly csg-like representation for interpretable shape parsing. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12478–12487, 2021. 2, 3, 8

[26] D. Ren, J. Zheng, J. Cai, J. Li, and J. Zhang. Extrudenet: Unsupervised inverse sketch-and-extrude for shape parsing. In *European Conference on Computer Vision*, pages 482–498. Springer, 2022. 3

[27] A. Sanghi, H. Chu, J. G. Lambourne, Y. Wang, C.-Y. Cheng, M. Fumero, and K. R. Malekshan. Clip-forge: Towards zero-shot text-to-shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18603–18613, 2022. 1

[28] R. Schönhof, J. Elstner, R. Manea, S. Tauber, R. Awad, and M. F. Huber. Simplified learning of cad features leveraging a deep residual autoencoder. *Procedia CIRP*, 109:84–88, 2022. 1, 2

[29] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5515–5523, 2018. 1, 4

[30] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji. Neural shape parsers for constructive solid geometry. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 44(05):2628–2640, 2022. 1, 2, 3, 4

[31] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 1, 2

[32] P.-S. Wang, C.-Y. Sun, Y. Liu, and X. Tong. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 1, 2

[33] W. Wang, X. Liu, H. Zhou, L. Wei, Z. Deng, M. Murshed, and X. Lu. Noise4denoise: Leveraging noise for unsupervised point cloud denoising. *Computational Visual Media*, 10(4):659–669, 2024. 1, 2

[34] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2690–2698, 2019. 1, 2

[35] S. Yan, Z. Yang, C. Ma, H. Huang, E. Vouga, and Q. Huang. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2753–2762, 2021. 1, 2

[36] F. Yu, Q. Chen, M. Tanveer, A. Mahdavi Amiri, and H. Zhang. D^2CSG: Unsupervised learning of compact csg trees with dual complements and dropouts. *Advances in Neural Information Processing Systems*, 36:22807–22819, 2023. 1, 2, 3

[37] F. Yu, Z. Chen, M. Li, A. Sanghi, H. Shayani, A. Mahdavi-Amiri, and H. Zhang. Capri-net: Learning compact cad shapes with adaptive primitive assembly. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11768–11778, 2022. 1, 2, 3, 5, 7, 8

[38] I. I. Zhegalkin. O tekhnike vychisleniy predlozheniy v simvolicheskoy logike (About a technique of computation of expressions in symbolic logic). *Mat. Sb.*, 34:9–28, 1927. 2