Inferring CAD Modeling Sequences Using Zone Graphs

Xianghao Xu
Brown University

Wenzhe Peng
MIT

Chin-Yi Cheng
Autodesk Research

Karl D.D. Willis
Autodesk Research

Daniel Ritchie
Brown University

Abstract

In computer-aided design (CAD), the ability to "reverse engineer" the modeling steps used to create 3D shapes is a long-sought-after goal. This process can be decomposed into two sub-problems: converting an input mesh or point cloud into a boundary representation (or B-rep), and then inferring modeling operations which construct this B-rep. In this paper, we present a new system for solving the second sub-problem. Central to our approach is a new geometric representation: the zone graph. Zones are the set of solid regions formed by extending all B-Rep faces and partitioning space with them; a zone graph has these zones as its nodes, with edges denoting geometric adjacencies between them. Zone graphs allow us to tractably work with industry-standard CAD operations, unlike prior work using CSG with parametric primitives. We focus on CAD programs consisting of sketch + extrude + Boolean operations, which are common in CAD practice. We phrase our problem as search in the space of such extrusions permitted by the zone graph, and we train a graph neural network to score potential extrusions in order to accelerate the search. We show that our approach outperforms an existing CSG inference baseline in terms of geometric reconstruction accuracy and reconstruction time, while also creating more plausible modeling sequences.

1. Introduction

Many real-world 3D objects begin their existence as parametric CAD programs. If one could recover such programs for everyday objects, it would enable powerful editing and re-purposing abilities, with many applications in mechanical and industrial design. As such, it's unsurprising that this problem has become a popular research topic in the graphics, vision, and machine learning communities, with multiple recent papers examining how to infer CAD-like programs for an input shape [22, 7, 30, 8, 16].

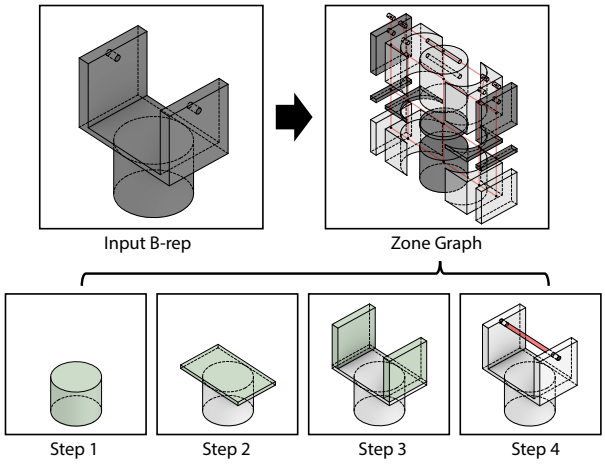


Figure 1: A solid shape (top-left) can be decomposed into a graph of *zones* which make up its interior volume and surrounding space (top-right). This representation permits efficient search for modeling sequences that reconstruct the shape, using industry standard modeling operations (bottom). Light red lines in the zone graph indicate connections between zone nodes. In the bottom modeling sequence, green denotes sketch + extrude + union and red denotes sketch + extrude + difference.

One property shared by these works is their use of constructive solid geometry (CSG) as a modeling language. In CSG, shapes are formed by combining primitive solids (e.g. spheres, cylinders, boxes) with Boolean union, intersection, and difference. Its small library of parametric primitive shapes makes it appealing for CAD program inference, as this reduces the search space of possible programs.

Unfortunately, parametric primitive CSG is not the modeling language used by CAD practitioners today. Instead, modern CAD workflows use *feature-based modeling*, in which a solid object is created by iteratively adding features such as holes, slots, or bosses [13]. Feature creation is typically performed via operations on surfaces, as these are intuitive for users to reason about: for instance, creating a slot in the surface of an object by sketching the profile of the

This work was done partially during Xianghao Xu and Wenzhe Peng's internship at Autodesk Research

slot and then specifying how deep it goes. Throughout this process, the object's geometry is stored as a *boundary representation*, or B-rep, which is a watertight assembly of surface patches which enclose the object's solid volume [27]. Feature-based modeling with B-reps supports Boolean operations between solids, making it strictly more expressive than parametric primitive CSG. This expressiveness comes at a cost for program inference, though, as the space of feature-based modeling programs is much larger.

How can we enable tractable inference of feature-based CAD programs? We first note another common property of prior work: solving CAD program inference "in one shot," going directly from input unstructured geometry to a CAD program. However, this problem actually decomposes into two sub-problems: (1) convert the input geometry into a B-rep, (2) infer a CAD program which generates that B-rep. The first sub-problem is of huge importance to the CAD industry, where it is known as *reverse engineering*. Many semi-automatic commercial tools exist [15, 14] and recent work has begun to leverage learning based approaches [23, 26]. The second problem is supported by all major CAD software using a rule-based approach, e.g. [1, 24]. However, such systems often require active input from a human designer, can fail to automatically infer earlier steps of longer modeling sequences, and struggle to generalize beyond pre-defined rules.

In this paper, we show that assuming a B-rep as input allows us to tractably attack the problem of automatically inferring industry-standard, feature-based CAD programs. Specifically, it allows us to use a new geometry representation that enables tractable program search. Extending all surface faces of the input B-rep into infinite surfaces and then partitioning space with those surfaces results in an arrangement of solid zones. The spatial connectivity of these zones forms a data structure which we call the *zone graph* (Figure 1, top-right). Searching for a CAD program that reconstructs the input shape then becomes a search for a sequence of modeling operations that fill in all the zones which are inside the input B-rep (Figure 1, bottom). This perspective reduces the search space of possible programs from an infinite set (a space of programs with many continuous parameters) to a finite set (the set of operation sequences that fill particular zones).

In this paper, we focus on modeling via sketch + extrude + Boolean operations, which are commonly-used in CAD workflows to create new solid masses and cut holes and slots out of existing ones. In addition, we *learn* how to guide the search using a large dataset of CAD modeling sequences. We train a graph neural network that takes the zone graph as input and predicts scores for different candidate modeling operations, allowing search to focus on higher-scoring options first. This allows our approach to infer modeling sequences which use meaningful design patterns without active guidance from a human designer.

In experiments on real-world CAD shapes, our zone graph method outperforms a recent state-of-the-art CSG inference method: it achieves better reconstruction accuracy by inferring more plausible programs that use industry-standard CAD operations.

In summary, the contributions of this paper are:

- The zone graph representation of B-rep solids, which reduces CAD program search space to a finite size.
- A search algorithm for inferring CAD modeling sequences from zone graphs.
- A graph neural network for learning to score candidate CAD operations during program search.

2. Related Work

Space Partitioning The zone graph is a partition of space formally known as an arrangement of surfaces [12]. Use of spatial partitioning data structures is common in computer graphics to accelerate spatial queries, particularly in ray tracing [3]. A more recent body of work has explored space partitioning for geometry reconstruction tasks. Polyfit [18] performs surface reconstruction from point clouds by extracting and intersecting planar primitives; other recent reconstruction works use a similar method [10, 2]. Learning-based methods have also begun to use space partitioning representations [5, 6]. BSP-Net [5] uses binary space partitioning to build up a constructive solid geometry (CSG) tree for compact mesh generation. The zone graph representation differs from prior work as it builds up an incidence graph of an arrangement of parametric surfaces and considers curved surfaces in addition to planar surfaces. We also apply zone graphs to a different reconstruction task: finding a sequence of modeling operations to reproduce a shape.

CAD Reconstruction CAD reconstruction involves recovering a sequence of CAD modeling operations from meshes, point clouds, or B-rep models. Such sequences are critical for preserving editability of CAD models, enabling downstream edits, such as model simplification for simulation, or adjusting tolerances for manufacturing. Although CAD reconstruction has been the subject of significant research [21], it remains a challenging problem due to the diverse ways that CAD models are constructed. Commercial CAD software uses rule-based feature recognition, often with user assistance, to detect and remove features such as holes, pockets, and fillets before re-applying them parametrically [1]. This strategy can recover some modeling operations but may fail to completely rebuild the parametric modeling history from the first step. We focus on a fully automatic approach that can recover the entire construction sequence in a manner consistent with human designs.

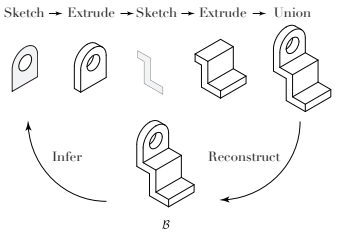
CAD reconstruction can also be framed as a program synthesis problem. InverseCSG [7] is one example; it uses a

constraint-based program synthesizer to find CSG programs whose output is consistent with the input geometry. More recently, learning-based approaches have been developed. CSGNet [22] leverages a neural network to infer a sequence of CSG operations on simple primitives such as spheres and cuboids. Other works in this area [8, 25, 16] also utilize simple primitives with CSG operations. However, professional mechanical design tools use a different paradigm, first creating 2D engineering sketches then lifting them to 3D using operations such as extrude, revolve, and sweep.

CAD reconstruction is a *visual program induction* problem [4]. One common approach for such problems is *neurally-guided search*, in which a neural network guides a search algorithm by prioritizing search options [20, 8, 9]. We also leverage neurally-guided search, developing novel search proposal and ranking algorithms for zone graphs.

3. Task & Approach

Given a 3D shape specified as a B-rep $\mathcal{B}$, our goal is to find a sequence of modeling operations $[\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n]$ which reproduce $\mathcal{B}$. We focus on sketch + extrude + Boolean operations, in which the user (a) sketches one or more closed profile curves on a plane (b) extrudes this sketch to form one or more solid regions, and (c) applies the extrusion to the current partial B-rep by union or difference (see inset).



To solve this problem, we first turn the input B-rep $\mathcal{B}$ into a zone graph $\mathcal{G}$ (Section 4). We then search for a sequence of operations that reproduces this zone graph by enumerating and evaluating sketch + extrude + Boolean operations that are consistent with the zone graph (Section 5). As there may be a large number of such operations, we use a learned guidance network to prioritize ones that are similar to those seen in a large dataset of example CAD programs (Section 5.2). We leverage the recently-released Fusion 360 Gallery reconstruction dataset that provides ground-truth, human-designed CAD programs created with sketch and extrude modeling operations [29].

4. The Zone Graph Representation

CAD modeling operates via the addition and removal of solid volumes to create 3D shapes, using intuitive feature-based operations. To search for CAD programs that can reconstruct a given target shape, one could explore the space of all parameterizations of all such modeling operations. This is a huge search space. Instead, we note that the target

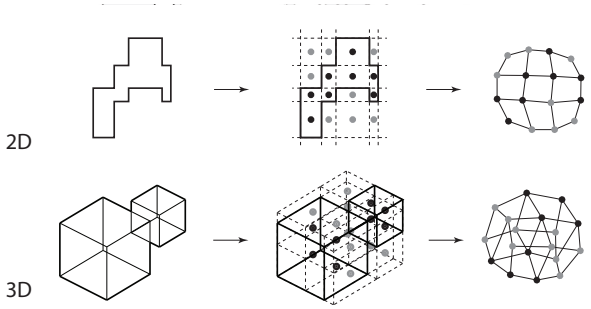


Figure 2: *(Left)* Input 2D and 3D shapes. *(Middle)* Shape decomposition into solid zones. *(Right)* The zone graph.

shape contains strong "clues" as to the operations used to create it. For example, suppose that a hole was created in a shape by subtracting some solid volume. While that volume is not present in the target, the interior faces of the hole it created reveal its shape. Our key idea is to formalize this notion of such "hidden volumes" and then restrict search to consider only the modeling operations that produce them or target shape interior volumes.

Our fist step is to construct from each input B-rep $\mathcal{B}$ a spatial data structure $\mathcal{G}$ that we call a *zone graph* (Figure 2). While we focus on 3D shapes in this paper, zone graphs are also well-defined in 2D; Figure 2 includes a 2D example for illustrative purposes. A zone graph in 3D (2D) is a graph $\mathcal{G} = (\mathcal{Z}, \mathcal{E})$ whose nodes $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_n\}$ represent solid regions (areas) called *zones* and whose edges $\mathcal{E}$ represent surface patches (curves) connecting those regions. A zone graph has the property that the union of all zones is the axis-aligned bounding box (AABB) of the input B-rep. In Figure 2, zones are colored black if they are *interior zones* $Z^\bullet$ that fill inside the volume enclosed by $\mathcal{B}$, and gray if they are *exterior zones* Z°.

Face Extension Zone graph construction begins by extending faces of $\mathcal{B}$ into infinite surfaces that partition space:

- *Planar faces* are extended into an infinite plane.
- *Generalized cylindrical faces* are extended along the face's direction of zero curvature.
- *Spherical faces* are extended into a complete sphere.
- *Free form faces* are not extended, as there is no known parametric form by which to extend them.

After extension, all faces are clipped by AABB($\mathcal{B}$).

Simplification The number of zones increases superlinearly with $\mathcal{B}$'s face count, leading to a larger search space. It helps to skip face extensions that are not likely to be useful for extrusion-based modeling. We search $\mathcal{B}$ for "face loops," sets of faces where (a) the faces form a cycle, (b)

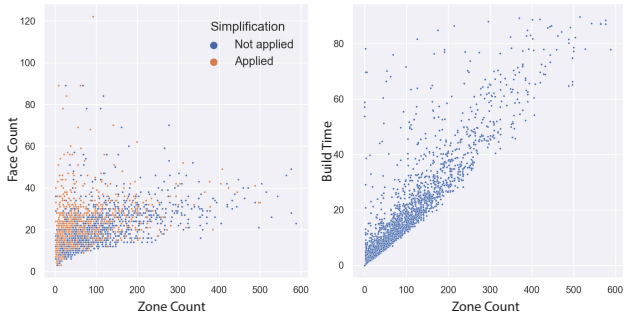


Figure 3: Zone graph construction statistics. *(Left)* Zone count vs. B-rep face count. *(Right)* Zone count vs. zone graph construction time (in seconds).

the edges shared by each consecutive pair of faces are parallel (we call this direction the *extrusion direction*), and (c) each face has exactly 4 edges in its outer wire. Faces in a loop are only extended along their extrusion direction. See the supplement for details.

Figure 3 visualizes some statistics for zone graphs extracted from the Fusion 360 Gallery reconstruction dataset [29]. On the left, we plot zone count against face count. On the right, we plot zone count against the time required to construct the zone graph (in seconds). Build time increases linearly with zone count, with most zone graphs taking under a minute. The build tests were run on an Intel Core i5-8259U processor using OpenCASCADE's general fuse algorithm (GFA) for solid partitioning via FreeCAD.

Our method can reconstruct 6900/8625 models in the dataset. Failure cases are due to (1) erroneous output from GFA caused by numerical robustness issues (such issues are well known [28, 17] and the subject of ongoing research) and (2) unsupported operations such as tapered extrude and revolve. Of these 6900 models, the zone graph can represent the ground truth modeling sequence for 5175 of them. Failure cases are due to (1) sequences not captured by our extrusion proposals (Section 5.1) and (2) sequences not expressible by the zone graph because some operations do not leave any trace in the target shape (see Section 7 for an example). See the supplemental material for more details, including ablation studies on zone graph simplification.

5. Searching for Modeling Sequences

Given a zone graph $\mathcal{G}$ for an input B-rep $\mathcal{B}$, our goal is to search for a sequence of CAD modeling operations $[\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n]$ that satisfies two properties:

1. Executing this sequence produces the input shape $\mathcal{B}$, i.e. $\text{exec}([\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n]) = \bigcup Z_i^\bullet$
2. Each step produces a shape which is a combination of zones in $\mathcal{G}$, i.e. $\forall j, \exists \mathcal{Z}' \subset \mathcal{Z}$ s.t. $\text{exec}([\mathbf{o}_1, \ldots, \mathbf{o}_j]) = \bigcup (Z_i \in \mathcal{Z}')$.

Our search algorithm satisfies property 2 by construction: it only considers modeling operations whose output coincides with available zones. If $\mathcal{B}$ is expressible as a sequence of sketch + extrude + boolean operations, and search time is unbounded, our algorithm also satisfies property 1. Otherwise, property 1 is only approximately satisfied.

Figure 4 shows an overview of our search algorithm. The input B-rep $\mathcal{B}$ is denoted as the *target*. The algorithm maintains a *canvas*, which contains the shape constructed by the modeling operations chosen by search thus far and is initially empty. At each step, the algorithm enumerates all combinations of zones which could be produced via a valid sketch + extrude operation (Section 5.1); these are the valid next steps for search to consider. The algorithm then scores how likely each of these proposals is to lead to a correct reconstruction of the target (Section 5.2). It retains the top k highest-scoring proposals and initially chooses the top 1 to explore next (i.e. best-first search). We use $k = 5$ unless otherwise specified. If the search reaches a terminal state (i.e. no valid extrusion proposals), but the canvas does not match the target, it backtracks to the previous step and considers the next extrusion in the top k set. Search terminates when the canvas matches the target, or when a computation time budget has been exhausted (in the latter case, the canvas with the highest reconstruction IoU is returned).

5.1. Generating Candidate Modeling Operations

Figure 4 right illustrates our process for identifying candidate modeling operations. It begins by finding pairs of parallel planes—the start and end planes for an extrusion (Figure 4 top-right). We define the *extrusion direction* d as the vector from the start plane to the end plane. Next, we identify the *starting sketch* $\mathcal{S}$, a set of faces on the start plane. Considering all possible such sets is intractable. Fortunately, the zone that a face is adjacent to along d suggests the operations it might be used in:

1. Faces d-adjacent to zones in the canvas $\mathcal{C}$ but not in the target $\mathcal{T}$ could start an extrude + subtract (Figure 4 bottom-right, 1).
2. Faces d-adjacent to zones in $\mathcal{T}$ but not in $\mathcal{C}$ could start an extrude + union (Figure 4 bottom-right, 2).
3. Faces d-adjacent to zones in $\mathcal{T}$ or empty zones could start an extrude + union (Figure 4 bottom-right, 3).

Candidate starting sketches $\mathcal{S}$ are the connected components of each group above, plus the union of these from each group (e.g. Figure 4 bottom-right, 1). The supplemental material describes other candidate enumeration strategies.

Each candidate sketch is then extruded along d to create a generalized cylinder; the zones which fall within this cylinder form the proposed extrusion volume $\mathcal{X}$. If $\mathcal{X} \subset \mathcal{C}$, it is marked as a subtraction. If $\mathcal{X} \cup \mathcal{C} = \varnothing$, it is marked as a union. Otherwise, we create proposals of both types.

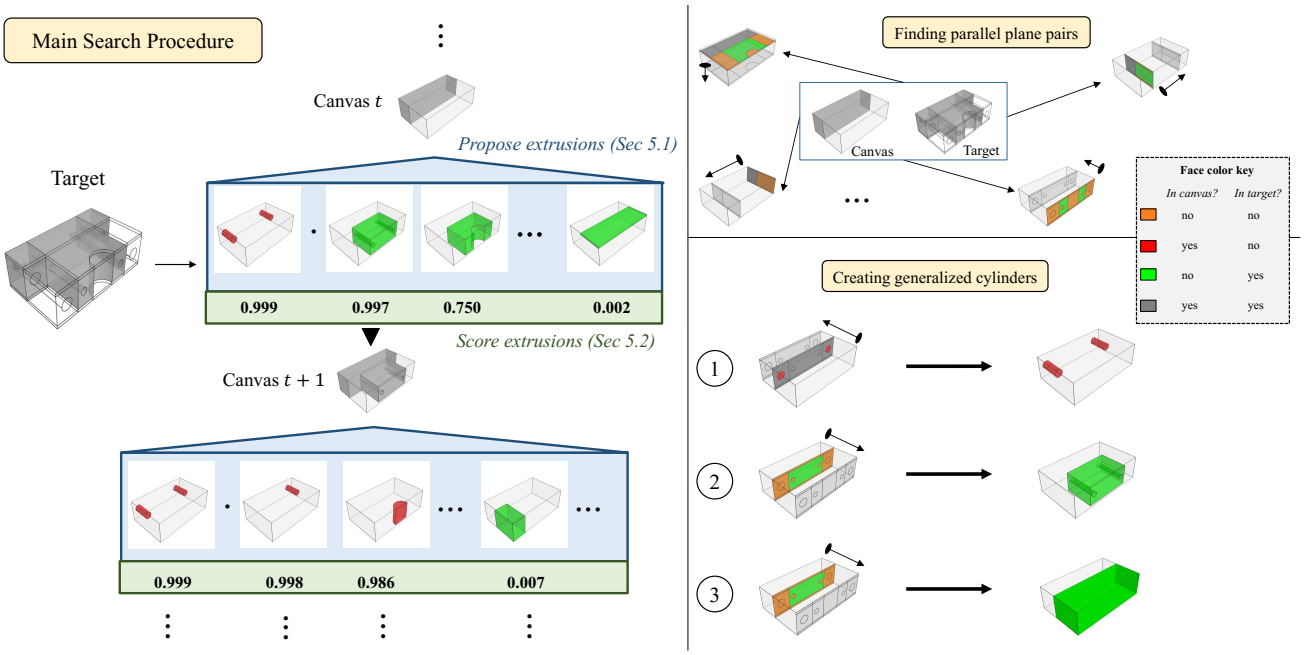


Figure 4: Searching for modeling sequences that reconstruct a Target shape. *(Left)* Search maintains a canvas (the partial shape constructed thus far), proposes possible next modeling operations, scores those operations, and then selects one of them to further explore based on its score. *(Right)* Candidate extrude operations are found by first finding all pairs of parallel planes and identifying which face groups could be used as the starting sketch (top-right) and then extruding this sketch to create generalized cylinders (bottom-right).

Computation of these proposals is memorized, as proposals frequently re-occur at different search iterations. To prevent search cycles, we discard a proposed extrusion if it is the inverse of one performed earlier in search (e.g. re-adding a volume that was previously subtracted).

While we have focused on extrude, a similar procedure could be applied to construct proposals for other operations. For example, fillet, chamfer, and taper all affect the zone graph in well-defined ways; it is possible to identify combinations of zones which can result from such operations.

5.2. Ranking Candidate Modeling Operations

Given a set of proposed modeling operations, our search algorithm must decide which ones to prioritize in its best-first exploration. We propose to *learn* what operations are best from a dataset of CAD modeling sequences. We train a neural network that takes as input the zone graph $\mathcal{G}$, the current canvas $\mathcal{C} \subset \mathcal{Z}$ (i.e. which zones are "filled"), the target shape $\mathcal{T} = \{Z_i^\bullet\}$ (i.e. the set of interior zones), and a modeling operation $\mathbf{o} = (\mathcal{X} \subset \mathcal{Z}, t \in \{\cup, -\})$, where $\mathcal{X}$ is a generalized cylinder extrusion and t denotes the type of operation (union or difference). The network's task is to predict $p(\mathbf{o}|\mathcal{G}, \mathcal{C}, \mathcal{T})$, how likely $\mathbf{o}$ is to be the next modeling operation, based on the patterns it has seen in its training set.

Network architecture Figure 5 shows our network architecture, which is a message passing graph convolutional network (GCN) [11]. The node features are derived from the geometry of the zone it represents and the information in the target shape $\mathcal{T}$, canvas $\mathcal{C}$, and proposed extrusion $\mathbf{o}$. Each zone is represented as a point cloud with per-point: positions $\mathbf{x}$, normals $\hat{\mathbf{n}}$; binary labels indicating whether the zone is part of $\mathcal{T}$, $\mathcal{C}$, and/or $\mathcal{X}$; the type t of the proposed extrusion. These point clouds are encoded using a PointNet [19] to produce the GCN initial node vectors. After 3 rounds of message passing, node vectors are aggregated via global max pooling and fed into a 3-layer MLP to produce the final output probability. The supplemental material contains ablations on the number of message passing rounds and the influence of point cloud features.

Training To create training data, we use the zone graphs constructed from modeling sequences in Section 4. We use 3000 sequences for training and hold out 440 for testing. Each step in each sequence is one training datum. One could treat the "ground truth" (GT) extrusion used at this step as a positive example, all others as negative examples, and minimize a binary cross entropy loss. This approach yields poor performance: there are often multiple approaches to construct a shape, and an approach that is not

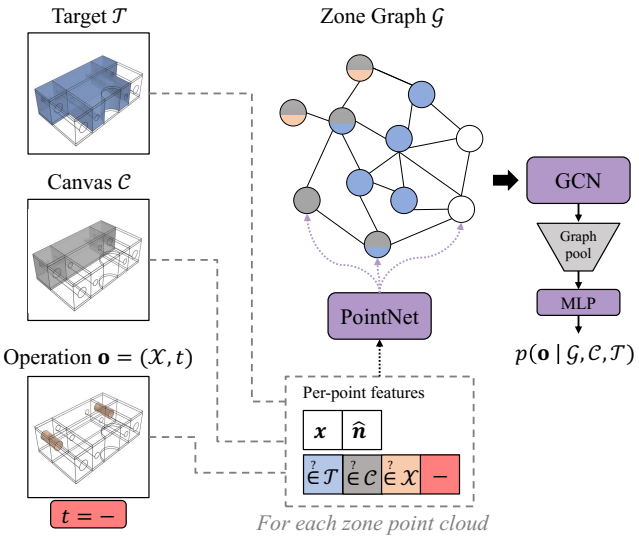


Figure 5: Architecture of our search proposal scoring network. Each zone is represented as a point cloud with positions $\mathbf{x}$ and normals $\hat{\mathbf{n}}$; labels for whether the zone is part of the target shape $\mathcal{T}$, current canvas $\mathcal{C}$, or proposed extrusion $\mathcal{X}$; and the type t of the proposed extrusion. Zones are encoded via a PointNet [19] to form the input node feature vectors for a graph convolution network (GCN). The output node vectors from the GCN are pooled and fed through an MLP to produce the final network output.

taken for one shape in the dataset may be taken for another. Treating all non-GT extrusions as negative examples thus confuses the network. We instead use ternary labels: positive, negative, and *neutral*. Positive labels are assigned to GT extrusions. Non-GT extrusions are labeled as neutral or negative via Monte Carlo tree search. For each extrusion, we perform N random modeling sequence completions starting with that extrusion (N = the number of remaining steps in the ground truth sequence) and record the percentage p of these completions which yield a match to the target shape. If any extrusion has $p = 0$, that extrusion is labeled as a negative example. Otherwise, if all extrusions have $p > 0$, the one with the smallest p is labeled as negative. All other extrusions are labeled as neutral. We then minimize a binary cross entropy loss using only positive and negative examples.

Heuristic Ranking We also considered whether heuristics could work in our setting. The best heuristic we found is one that executes a candidate extrusion and, like IoU, penalizes the filled zones that the resulting canvas and target do not have in common: $\frac{|\mathcal{Z}|-(|\mathcal{T}\cup\mathcal{C}|-|\mathcal{T}\cap\mathcal{C}|)}{|\mathcal{Z}|}$. Ties are broken using volumetric IoU between canvas and target. This performs considerably better than random but not as well as our network; the next section provides more detail.

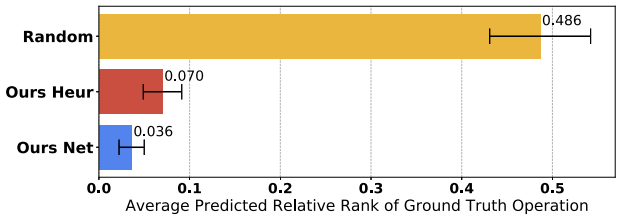


Figure 6: Comparing how different methods rank the ground truth extrusions used in modeling sequences from the Fusion 360 dataset. The x axis is $\frac{\text{predicted rank of ground truth}}{\text{number of candidate extrusions}}$ averaged over all steps in all test sequences. Lower is better. 90% confidence intervals are shown as error bars. *Ours Net* performs about 2x better than *Ours Heur*.

6. Results & Evaluation

To evaluate our method's performance, we are interested in three questions:

1. How consistent are our method's output sequences with sequences it was trained on?
2. How well does our method reconstruct new input shapes, given a particular compute budget?
3. How desirable are our method's output sequences, as judged by CAD designers?

For (1), we examine how our network scores the modeling steps used in its training data (Section 6.1). For (2), we quantify our method's tradeoff between computation budget and reconstruction accuracy (Section 6.2) and compare to a recent CSG inference system (Section 6.3). For (3), we conduct a perceptual study (Section 6.4).

6.1. Search Proposal Ranking

If our network has learned the design patterns within its training data, then for each step in a dataset sequence, when candidate extrusions are ordered by predicted score, the "ground truth" extrusion should be ranked near the top. Figure 6 shows the average relative rank of the ground truth extrusion computed by our network and other baselines. The evaluation was conducted on the held-out test sequences from the Fusion 360 Gallery reconstruction dataset [29]. We compare our network (*Our Net*) against the heuristic (*Our Heur*), as well as a method which picks a random extrusion (*Random*). Our network and the heuristic both significantly outperform the random baseline, and the network also dominates the heuristic by a factor of two. Search guided by our network should thus mimic the design patterns in the dataset. Ablation studies are shown in the supplemental material.

Figure 7 shows modeling sequences inferred under different guidance (random, heuristic, network) for the same input shape. While the heuristic produces the shortest sequence, the network produces a more intuitive one.

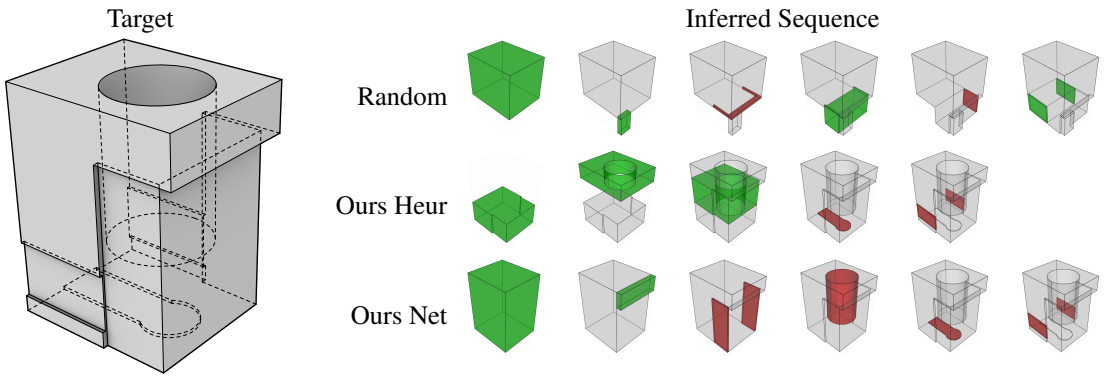


Figure 7: Modeling sequences inferred under different search guidance. Green: addition; Red: subtraction; Grey: current.

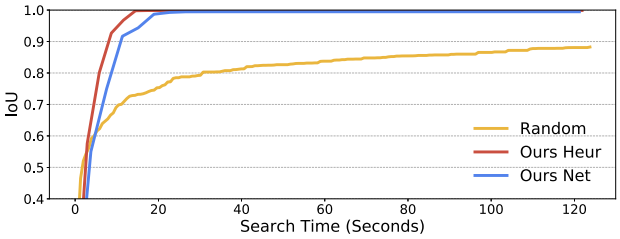


Figure 8: Reconstruction accuracy of the outputs of inferred programs vs. the time used to infer them.

Method	All Models		Sketch + Extrude Models	
	Error (%)	Time (s)	Error (%)	Time (s)
Ours Net	0.72	242	0.23	242
Ours Heur	**0.50**	**197**	**0.15**	**197**
InverseCSG	0.88	900	0.79	900

Table 1: Reconstruction results comparing InverseCSG and our method using heuristic (Ours Heur) or network guided (Ours Net) search. We report the median reconstruction error computed using IoU and the median search time in seconds for all models in the InverseCSG test set and the subset of sketch + extrude models. Lower values are better.

6.2. Reconstruction Performance Ablation Study

We next evaluate how well the modeling sequences inferred by our method reconstruct the input shape (via volumetric IoU). As stated in Section 5, if a sketch + extrude + boolean sequence that reconstructs a shape exists, our method is guaranteed to find it given unbounded time; here we are interested in our method's performance under finite time constraints. Timings were collected on a machine with a GeForce RTX 2080 Ti GPU and an Intel i9-9900K CPU.

Figure 8 compares the average IoU of inferred programs on our held-out shapes (with a ground truth modeling sequence length of at least 3) using random, heuristic, and network guidance. Zone graph construction time is excluded, as this is a fixed cost incurred by all methods. Both network and heuristic guidance converge to 100% reconstruction accuracy after $\sim$ 20 seconds, whereas random guidance does not converge. Heuristic guidance is faster than the network because (a) it does not incur the cost of network evaluation, and (b) by construction it tries to use the fewest possible modeling operations. However, shorter sequences are not always better (e.g. Figure 7). Ablation studies for different search widths k are shown in the supplemental material.

6.3. Comparison to InverseCSG

We next compare to InverseCSG [7], a recent system for inferring CSG programs from complex input shapes. We evaluate on InverseCSG's test set of 50 3D shapes. As B-rep files are not provided, we hired freelance CAD designers to reproduce them using Autodesk Fusion 360. Our method successfully build zone graphs for 33 of these, which we use as our test set. From the test set, 27 shapes are expressible using sketch + extrude + boolean operations, which our method can reconstruct exactly. By contrast, InverseCSG uses parametric primitives that may introduce error when the primitive set is insufficiently expressive. As these shapes are more complex than those in prior experiments, for our method we set the search width $k = 15$ and decrease it by 0.5 for each search depth increment (giving more search options to important early steps in the modeling process). Finally, we note that InverseCSG takes a triangle mesh as input to solve the CAD reconstruction problem in one shot, whereas we focus on the second sub-problem of inferring a modeling sequence from a B-rep.

Table 1 shows the results of this experiment. Our method achieves a lower median reconstruction error (computed using IoU) and a shorter median search time in all cases. Our method reconstructs 13 of the 33 models exactly, with a reconstruction error of less than 0.01%, in comparison to 1 exact reconstruction by InverseCSG. This result indicates that our representation can more accurately represent typical CAD models when compared to parametric primitive

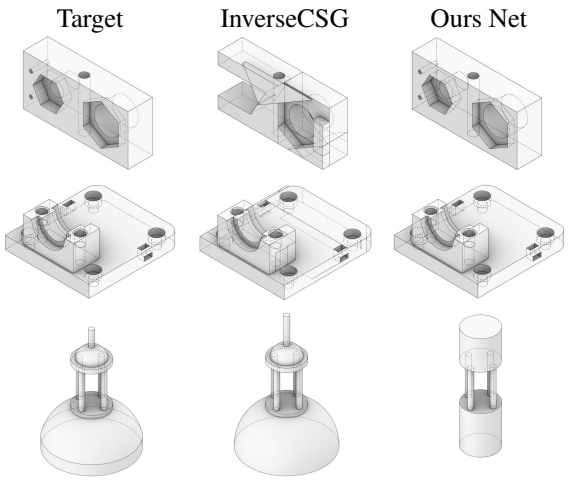


Figure 9: Qualitative comparison of the output of our model's inferred programs vs. those of InverseCSG.

CSG. We provide per model reconstruction results and additional details in the supplemental material.

Figure 9 shows reconstructions by our method and InverseCSG. The first row shows a case where our method perfectly reconstructs the shape but InverseCSG incurs some approximation error due to its use of primitives. The second row shows a case where both methods perform well. The third row shows a failure case for our method (a revolve operation is needed to express this shape).

6.4. Expert Perceptual Evaluation

Finally, we evaluate inferred programs via a perceptual study. We recruited 16 experienced CAD users from the fields of design (5) and engineering (11). Each participant is asked to describe the type of modeling practice they engaged in more often: reconstructing existing shapes, or exploratory design. Each participant then performs 45 comparisons of two modeling sequences shown in random order. In each comparison, the participant is shown a target shape along with two modeling sequences for it and asked to (a) select the one most similar to how they would have constructed that shape, or (b) indicate that they cannot decide. Comparisons were between (1) Ours Net vs. InverseCSG on the InverseCSG test set, (2) Ours Net vs. Ground Truth from the Fusion 360 Gallery test set, (3) Our Net vs. Ours Heur on the Fusion 360 Gallery test set.

Table 2 shows the results of this experiment. Participants strongly preferred Our Net sequences to those of InverseCSG. Our Net sequences were also judged as better than the ground truth about a third of the time. Participants who focus on exploratory design preferred Our Net sequences vs. Our Heur ones at a significantly higher rate than participants focused on reconstruction. This indicates that the sequences found by the learned guidance better support exploratory modification, whereas the heuristic sequences may be better suited to direct reconstruction.

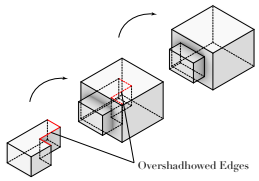
	% Chosen		
Ours Net vs.	Overall	By "explorers"	By "reconstructers"
Ground Truth	34.1	33.3	34.0
InverseCSG	84.7	76.7	86.4
Ours Heur	41.0	52.5	37.9

Table 2: Results of a perceptual study in which CAD users compared modeling sequences inferred by our method to those produced by InverseCSG.

7. Conclusion

In this paper, we presented a new representation for CAD reconstruction, the *zone graph*. We showed how it reduces the search space of CAD modeling sequences that reconstruct a shape to finite size, and we presented an algorithm for searching the space of sketch + extrude + Boolean modeling sequences. We also introduced a graph neural network that learns which search paths to explore first based on design patterns in a dataset of CAD modeling sequences. Our experiments showed that our approach reconstructs a large percentage of input shapes and does so with more desirable modeling sequences than InverseCSG.

Some modeling sequences are not recoverable from the zone graph of the output shape, as they may include one or more steps which leaves no trace of itself in the output B-rep (see inset). We can still infer alternative (and potentially equally-good) reconstruction sequences for such shapes, however.



Finally, we used a large CAD dataset to train our proposal scoring network. Aside from the Fusion 360 Gallery reconstruction dataset, such data is not widely available. It is important to investigate weakly-supervised approaches that require only a small amount of human input about what constitutes a meaningful design pattern.

Acknowledgments

We would like to thank Justin Solomon for pointing us toward the literature on arrangements of surfaces within computational geometry. Thanks also to the Autodesk designers who participated in the study. Daniel Ritchie is an advisor to Geopipe and owns equity in the company. Geopipe is a start-up that is developing 3D technology to build immersive virtual copies of the real world with applications in various fields, including games and architecture.

References

[1] Autodesk. *Inventor Feature Recognition*, 2012.

[2] Jean-Philippe Bauchet and Florent Lafarge. Kinetic shape reconstruction. *ACM Transactions on Graphics (TOG)*, 39(5):1–14, 2020.

[3] A.Y. Chang. *A survey of geometric data structures for Ray Tracing*. Polytechnic University,Department of Computer and Information Science, 2001.

[4] Siddhartha Chaudhuri, Daniel Ritchie, Jiajun Wu, Kai Xu, and Hao Zhang. Learning Generative Models of 3D Structures. *Computer Graphics Forum*, 2020.

[5] Zhiqin Chen, Andrea Tagliasacchi, and Hao 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.

[6] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 31–44, 2020.

[7] 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)*, 37(6):1–16, 2018.

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

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

[10] Hao Fang and Florent Lafarge. Connect-and-slice: an hybrid approach for reconstructing 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13490–13498, 2020.

[11] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, arXiv:1704.01212, 2017.

[12] Dan Halperin and Micha Sharir. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, Inc., USA, 1997.

[13] Christoph Martin Hoffmann. *Geometric and solid modeling*. CUMINCAD, 1989.

[14] ANSYS Inc. Reverse Engineering Software — Ansys SpaceClaim. https://www.ansys.com/products/3d-design/ansys-spaceclaim/reverse-engineering, 2020. Accessed: 2020-11-12.

[15] 3D Systems Inc. Geomagic X 3D Reverse Engineering Software. https://www.3dsystems.com/software/geomagic-design-x, 2020. Accessed: 2020-11-12.

[16] Kacper Kania, Maciej Zieba, and Tomasz Kajdanowicz. Ucsg-net–unsupervised discovering of constructive solid geometry tree. *arXiv preprint arXiv:2006.09102*, 2020.

[17] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry*, 40(1):61–78, 2008.

[18] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2353–2361, 2017.

[19] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah Goodman. Neurally-Guided Procedural Models: Amortized Inference for Procedural Graphics Programs Using Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[21] Jami J Shah, David Anderson, Yong Se Kim, and Sanjay Joshi. A discourse on geometric feature recognition from cad models. *J. Comput. Inf. Sci. Eng.*, 1(1):41–51, 2001.

[22] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. corr abs/1712.08290 (2017). *arXiv preprint arXiv:1712.08290*, 2017.

[23] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision (ECCV)*, pages 261–276. Springer, 2020.

[24] Dassault Systems. *Solidworks FeatureWorks*, 2019.

[25] 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. In *International Conference on Learning Representations*, 2019.

[26] Xiaogang Wang, Yuelang Xu, Kai Xu, Andrea Tagliasacchi, Bin Zhou, Ali Mahdavi-Amiri, and Hao Zhang. Pie-net: Parametric inference of point cloud edges. 2020.

[27] K.J. Weiler. *Topological structures for geometric modeling*. Technical report RPI, Center for Interactive Computer Graphics. University Microfilms, 1986.

[28] Kevin Weiler, Tom Duff, Steve Fortune, Chris Hoffman, and Tom Peters. Is robust geometry possible?(panel). In *ACM SIGGRAPH 98 Conference abstracts and applications*, pages 217–219, 1998.

[29] 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. *arXiv preprint arXiv:2010.02392*, 2020.

[30] Chenghui Zhou, Chun-Liang Li, and Barnabas Poczos. Unsupervised program synthesis for images using tree-structured lstm, 2020.