DOI: 10.1111/cgf.13504
Eurographics Symposium on Geometry Processing 2018
T. Ju and A. Vaxman
(Guest Editors)

Volume 37 (2018), Number 5

Constructing 3D CSG Models from 3D Raw Point Clouds

Q. Wu[1], K. Xu[2] and J. Wang[†1]

[1] Nanjing University of Aeronautics and Astronautics, China
[2]National University of Defense Technology, China

Abstract

The Constructive Solid Geometry (CSG) tree, encoding the generative process of an object by a recursive compositional structure of bounded primitives, constitutes an important structural representation of 3D objects. Therefore, automatically recovering such a compositional structure from the raw point cloud of an object represents a high-level reverse engineering problem, finding applications from structure and functionality analysis to creative redesign. We propose an effective method to construct CSG models and trees directly over raw point clouds. Specifically, a large number of hypothetical bounded primitive candidates are first extracted from raw scans, followed by a carefully designed pruning strategy. We then choose to approximate the target CSG model by the combination of a subset of these candidates with corresponding Boolean operations using a binary optimization technique, from which the corresponding CSG tree can be derived. Our method attempts to consider the minimal description length concept in the point cloud analysis setting, where the objective function is designed to minimize the construction error and complexity simultaneously. We demonstrate the effectiveness and robustness of our method with extensive experiments on real scan data with various complexities and styles.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Shape Modeling—Procedural modeling

1. Introduction

Constructing 3D models from raw point clouds is one of the most fundamental tasks in computer aided design and computer graphics. The ultimate goals encompass three aspects: reproduction, quality control, and redesign and modification applications [LMM04]. Typically, different goals require different degrees of recovering. For the third aspect, some high-level design information embedded in the data points, such as the geometric structures and the logic of construction, is desired, which is the focus of our work. More precisely, to support a higher level of interaction with constructions, we claim to parameterize the compositional structures of models and synchronously record the entire logics of the constructions.

The CSG scheme based on bounded primitives is an effective parameterized representation, in which each model possesses a CSG tree structure with solid primitives as the leaves and operations as the internal nodes of the tree. The CSG tree records the non-trivial collection of generative processes and describes the advanced object structures well. Existing researches [RV85, SV91, Sha01, LFP14] study CSG construction from Boundary representations (B-reps) of 3D models, which is essentially a representation conversion problem. Different from these works, our work aims to construct CSG models directly from raw 3D point clouds (see Figure 1). In many real scenarios, there are no digital CAD models for targets, but only physical objects. Reconstructing digital models from 3D point cloud data is hence practically useful. This poses

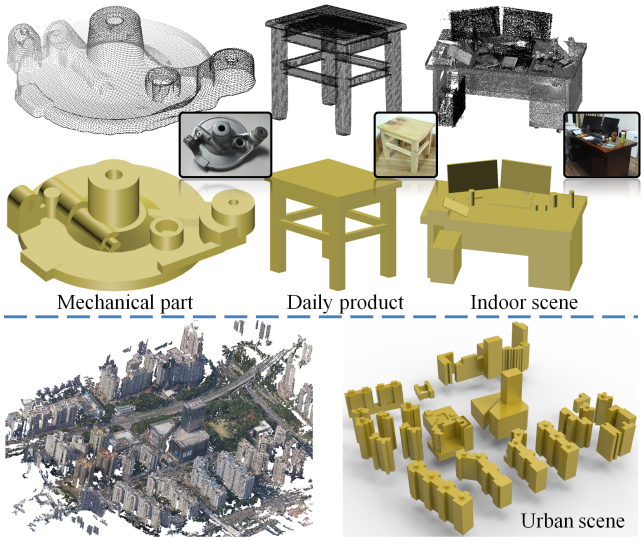

Figure 1: *CSG constructions for various raw point clouds.*

† Corresponding author: junwang@outlook.com

a number of challenges. First, raw point clouds capturing object surfaces are unstructured without well-defined topology, thus inferring geometrically interpretable explanations of shapes in terms of primitives, is exceedingly difficult. Our method approaches the problem by first constructing surface patches as an intermediate representation, based on the method in [LWC*11]. Surface patch representation is a more suited entity, on which geometric properties and constraints can be reliably imposed. We then devise a robust algorithm to derive all potential primitives for subsequent CSG construction based on these surface patches. Second, the orders of primitives and their Boolean operations matter during CSG construction. From this aspect, we propose a solution with two successive constructions to avoid the adverse effect of orders and employ some techniques to optimize the solution. An optimal subset of primitives with corresponding Boolean operations is finally provided to achieve a faithful construction. Third, a challenge is that we would like to model a scan, without relying on a vast number of primitives. Inspired by the smallest grammar for processing a string of characters [CLL*05], we incorporate the minimal description length concept into our CSG construction framework, and thereby design the objective function for CSG construction by balancing the construction accuracy and complexity simultaneously. This, however, does not immediately lead to the most compact representation, since [FP16] can provide a compact representation for raw scans as well and there is no discrimination between the two representations in terms of compactness.

Specifically, the proposed method consists of the CSG primitive generation and the CSG model construction. In the generation stage, a number of CSG bounded primitive hypotheses are extracted from raw point clouds. These CSG primitives typically include cuboids, cylinders, cones, spheres and tori. The number of initial hypotheses could be huge, ranging from a few hundred for the simpler shapes to several thousand for the more complex shapes, which consequently poses high complexity on the following CSG construction. To address this issue, several effective criteria are carefully designed to filter out incorrect and redundant hypotheses. Subsequently, the algorithm is developed to determine an optimal subset of primitive candidates with corresponding Boolean operations for the final construction. These Boolean operators typically include *union* ($\bigcup$), *intersection* ($\bigcap$) and *difference* ($-$). The union operation is commutative and associative, and hence the order of operands does not affect the result. The order of the difference and intersection operations can also be ignored, when the start operand is determined previously, e.g., $(A-B\bigcap C)$ is equal to $(A\bigcap C-B)$. Therefore, it's possible to complete a CSG construction without paying much attention to the orders. In our CSG construction, we separate *union* operation from other two Boolean operations, *intersection* and *difference*, which results in a bottom-up construction solution with two successive phases. *Intersection* and *difference* are the Boolean operations used in the first phase, while *union* can only be adopted in the second phase. In each phase, we convert the primitive selection problem to a combination optimization and employ a binary optimization technique to solve it. With this separation and conversion, we can achieve a faithful CSG construction efficiently and meantime the CSG tree derived from the whole procedure represents the original design intent well. This work can be regarded as a "deep" inverse engineering of hypostatizing real objects into CSG representations. We hope our work makes a step forward along this line of research, and would inspire more follow-up researches.

Overall, the main contributions of our work are:

1. We design an effective CSG construction framework directly over raw point clouds. The generated CSG models and CSG trees can be beneficial for structure and functionality analysis applications as well as creative redesign.
2. We propose a CSG bounded primitive generation and filtering approach, which extracts potential primitives from raw point clouds for final CSG construction.
3. We formulate the CSG construction as an energy minimization problem and come up with a bottom-up solution, which results in a CSG tree representing the original design intent well.

1.1. Related Works

An extensive number of works on model construction directly from scanned point clouds are developed. In this section, we focus on the most closely related works to ours, in particular those regarding constraint-based construction, feature-based construction, tree-based construction and the CSG conversion problem.

Constraint-based construction. The pioneering works of [WFAR98, WFRA99] present frameworks for the integration of geometric relationships in object reconstruction. The basic idea is to minimize a function containing a least-squares term and a penalty term associated with the constraints. These methods use a complex formulation of the constraint function, which heavily relies on the convexity of the constraint space. Fisher et al. [Fis04] and Thrun et al. [TW05] took advantage of prior knowledge of models, placement constraints, or extracted features, like principal directions, for construction. Their work demonstrates the effectiveness of domain knowledge of standard shapes and relationships for architectural construction, parameter estimation, and data completion using non-local relations. Rabbani et al. [RvdH04] constructed models with the constraint from point clouds, which is based on a library of CAD models. However, constructing CAD models for all objects in the real world is impractical. Given a list of fitted primitives with corresponding point sets, Li et al. [LWC*11] presented the surface construction to recover relations among both local and distant parts of man-made objects. However, their method cannot guarantee to extract a compact set of surface patch primitives that models an object, since the dimension constraints are not considered during the construction from raw point clouds. In addition, the constructed models lack additional semantics and inherent topology. Consequently, they are not available for part-based editing, feature-based NC tool path generation, and technical data package preparation, in contrast to objects built using CSG methods.

Feature-based construction. The feature-based strategy is widely studied to carry out model construction for industrial products [YLC*08, DRD10, BFL*10, NSZ*10]. Ye et al. [YLC*08] introduced the construction of feature-based parametric models from scanned data, which makes design and knowledge reuse possible for 3D digital design. However, the method considers only the surface patch features rather than compositional solid features. Beccari et al. [BFL*10] designed a reverse engineering method for fast

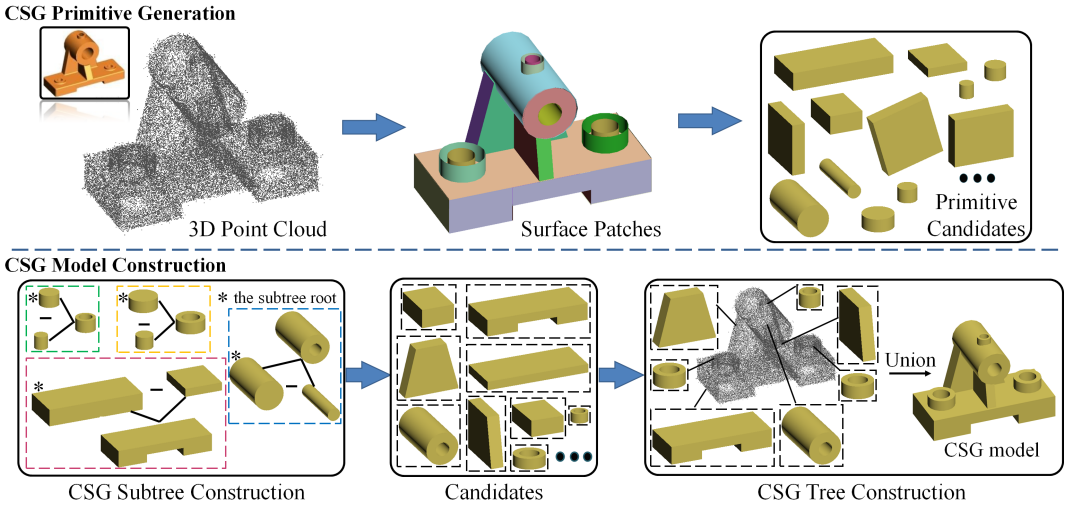

Figure 2: *Method overview. Starting from an imperfect point cloud of a mechanical part, we first construct its surface model, based on which, we then extract a great deal of geometrically interpretable shape primitives (the first row). With these primitives, we proceed to the CSG construction, with a bottom-up solution obeying the order from the subtree to the entire tree construction (the second row). Primitives with the ∗ mark in this figure or the following figures are recognized as the roots or the start primitives.*

and interactive acquisition and construction of a digital 3D model representing an existing physical object. This method takes as input the curve network of contours of the 3D model rather than the scanned point cloud, which inevitably ignores a slice of detailed features. Nan et al. [NSZ*10] designed an interactive tool, SmartBoxes, for reconstruction directly over point clouds, which assembles detailed 3D primitives by utilizing the global regularity of models and simultaneously requires a moderate amount of user intervention. Our method presents a volume representation with various CSG primitive features for input data automatically, which can be directly used for structure and functionality analysis.

Tree-based construction. To support a higher level of interaction with constructed objects, Silva et al. [SFV*05] pioneered the recovery of construction trees from input point clouds, which utilizes parsimony to control the tree size. In spite of this, the sizes of the generated trees are large, making the application limited to simple data sets. Fayolle et al. [FPK*08] took a list of fitted primitives and the corresponding point set as inputs to evolve a linear tree for real mechanical parts. Their method is illustrated by the fitting of template parameterized models to point clouds, rather than some fundamental primitives in our method. These templates with a slice of parameters are arduously adjusted to fit the data. The recent work [SGL*17] presents a neural architecture that takes as input a 2D or 3D shape and outputs a CSG parse tree that generates the shape. However, the generated CSG parse tree is linear, which means their method is difficult to be applied to some CSG models with several subtree structures. In addition, during the constructions of some CSG models, a few additional primitives are needed, which are hard to be detected in their current method. Fayolle et al. [FP16] employed an evolutionary approach to extract a construction FRep tree from a point cloud. By the fixed maximum allowed depth, the extracted tree is limited in size, which significantly improves its reusability. However, as mentioned in the paper, the tree may contain redundant information resulting from the usage of genetic programming and the recovered object may lose a slice of details (e.g., cylinders with small radii). Comparatively, we recover CSG bounded primitive features from the input scans and simultaneously explore the generative processes to build CSG trees. The CSG trees are designed to be as short as possible, without much redundant information. In addition, small details once detected could scarcely be ignored during our construction.

CSG conversion. Many efforts have been dedicated to the CSG conversion problem [RV85, SV91, SV93, BC04]. Buchele et al. [BR01] examined the conversion between Binary Space Partition (BSP) trees and half-space CSG trees. The two representations can both provide hierarchical constructions. However, constructing either representation directly from raw scans is still a challenge. For example, these representations may require some additional half-spaces or primitives not available from the surface information or from the segmentation of the input scans. In addition, the partition surface in a BSP tree or the half-space in a half-space CSG tree may contribute to the construction of different parts of the final model, which makes it unclear how useful the recovered tree is for further modification and editing. One of the attractive features of the bounded primitive CSG representation is its ability to represent a large number of complex objects, by applying Boolean operations to a few simple geometric primitives. The role of each bounded primitive is definite, which can facilitate the applications, such as editing and redesign. Therefore, the bounded primitive CSG representation is adopted in our research for raw point clouds. Shapiro et al. [SV93] converted natural quadric B-reps in Parasolid to efficient CSG representations in PADL-2 effectively. Several separating half-spaces should be constructed for the conversion, which is based on the boundary information of B-reps. McCad is a well-known geometry conversion tool developed at KIT to enable the automatic conversion of CAD models into half-space CSG. Lu

et al. [LFP14] improved the decomposition part and demonstrated greater stability and more enhanced efficiency than the original M-cCad conversion process. Compared to the conversions above, our problem is more challenging due to the unstructured inputs, raw point clouds. It may be natural to consider our CSG construction problem by first constructing B-reps from point clouds and then employing the conversions above to achieve the CSG construction. However, inferring high-level information, such as boundary, topology, bounded composition etc., directly from point clouds to construct B-reps remains a challenging problem in geometry processing. Surface patches as an intermediate representation in our algorithm are constructed without considering global dimension constraints and don't possess any connection relations. Thus, the B-rep $\rightarrow$ CSG conversion techniques cannot be used in our research.

2. Overview

In this section, we provide a brief overview of our method. The proposed approach takes as input the raw scans of real models acquired by 3D scanning devices, represented as unorganized 3D point clouds. Our goal is to automatically construct 3D CSG models and to simultaneously recover the associated CSG expression trees. This construction problem can be formulated as follows.

Problem Statement. Let $\mathbf{P}$ be the input data. We encode the model to be constructed as a context-free tree: $\mathcal{T} = \odot_i \mathbf{pr}_i \odot^* \mathbf{Pr}^*$, where $\mathbf{pr}_i$ is a CSG primitive, $\odot_i$ is the Boolean operator for $\mathbf{pr}_i$, $\mathbf{Pr}^*$ and $\odot^*$ denote the remaining primitives and Boolean operators, respectively. The construction is intended to be geometrically faithful to the input data, while the number of required primitives should be as small as possible, namely, the CSG tree should be as short as possible. On this basis, we formulate the process as:

$$\min_{\mathcal{T}} \sum_i w_i \sum_j dist^2(p_j^{(i)}, \mathbf{pr}_i) + \lambda |\{\mathbf{pr}_i\}| \qquad (1)$$

where $p_j^{(i)} \in \mathbf{P}$ is the j-th point related to the CSG primitive $\mathbf{pr}_i$; $dist(p_j^{(i)}, \mathbf{pr}_i)$ measures the Euclidean distance from the point $p_j^{(i)}$ to its CSG primitive $\mathbf{pr}_i$; $|\{\mathbf{pr}_i\}|$ stands for the number of required primitives to form the final model; w_i is the reciprocal of the number of points belonging to $\mathbf{pr}_i$; λ is the weight to balance the faithfulness of the construction and the complexity of the final tree. Figure 2 presents the overview of our proposed framework, which consists of two main stages.

CSG Primitive Generation. Taking as input the raw scan of an object, the robust fitting technique is first applied to extract the surface patches, among which the globally mutual relations are enforced. On this basis, all possible CSG bounded primitives are constructed automatically in Section 3, denoted as $\mathbf{Pr}$, functioning as fundamental candidates for our CSG construction algorithm. Specifically, we generate cuboid hypotheses from the fitted planar patches, and devise several effective filters to prune the false hypotheses. For the quadratic primitives, the fitted patches well represent their shapes, which are constructed directly. Furthermore, for each fitted quadratic patch, we construct an *additional primitive* to expand our candidate set for more faithful CSG construction.

CSG Model Construction. Our framework aims to capture the design intents from the raw scans together with the CSG primitive candidates $\mathbf{Pr}$, and then hypostatizes them in some CSG models and CSG trees. We proceed with the CSG model construction by obeying the order of CSG tree recovery from the subtrees to the entirety in Section 4. We propose to generate subtree candidates for the final CSG model based on a binary optimization approach. The objective function is designed to minimize the primitive truncation error and the complexity of the subtree simultaneously. After combining these subtree structures with original primitive candidates, we choose to approximate the target scan by the assembly of an optimal subset of these candidates. Eventually, we can obtain the model $\mathcal{M}$ which best describes the underlining geometry of the input point set. As the construction process is recorded, the CSG tree $\mathcal{T}$ is readily obtained for further model editing and redesign.

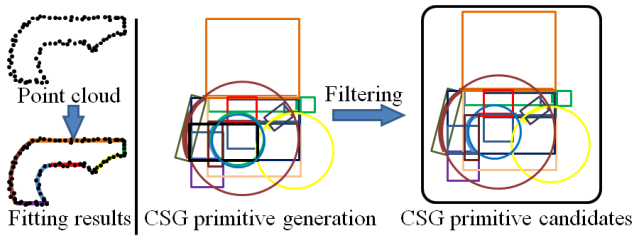

Figure 3: *2D CSG primitive generation process. Given the 2D point cloud, we first fit lines and arcs under the global constraints in [LWC*11] (the left). On this basis, many 2D CSG primitives, such as rectangles and circles, can be constructed (the middle). We design some criteria to filter out unreasonable and redundant primitives and finally get the CSG primitive candidates (the right).*

3. CSG Primitive Generation

In this section, we discuss how to generate the CSG bounded primitives from the input point cloud, which are used later for the construction of the final CSG model. Given the input data, we exploit the robust fitting technique [LWC*11] to construct the element surfaces. The element surfaces consist of the planar, cylindrical, conical, spherical and toroidal patches. Based on these fitted surface patches, we generate their corresponding CSG bounded primitives. Note that there can be many alternative ways to fit patches, such as [GG04] or [SWK07]. In our research, we employ GlobFit [LWC*11], where the patches are constrained by some global relations, including orientation, placement and equality alignments. Figure 3 presents the 2D CSG primitive generation process.

3.1. Cuboid Generation

We generate cuboid hypotheses from the fitted planar patches, which is based on the mutual relationships among the orientation directions of a cuboid. Planar patches can be combined to construct cuboid hypotheses, only if their normal orientations are orthogonal or parallel in pairs. Therefore, we first group planar patches. Each group represents a kind of cuboid hypothesis, of which the six face normal vectors are considered as its symbol. In such a case, the scale of cuboid generation is restricted in one group. We then propose to construct cuboid hypotheses in each group based

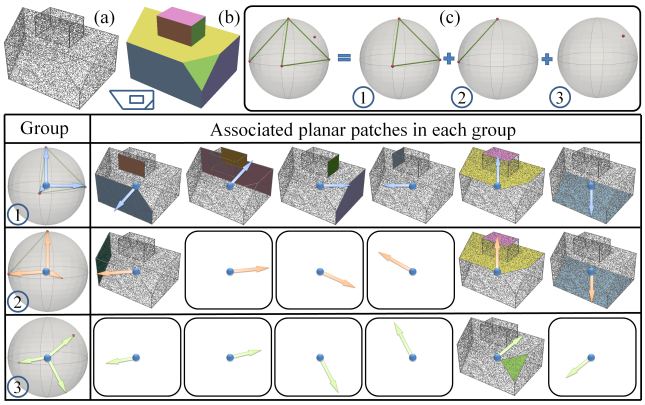

Figure 4: *Graph-based grouping of a 3D model. (a) The input point cloud; (b) All fitted patches; (c) Graph construction and decomposition. The graph is participated into three groups including one triangle loop, one edge, and one single node. The table in the second row presents the grouping result of planar patches. Each row of the table represents a group and the patches in the group are distributed along six directions, namely, the group symbol.*

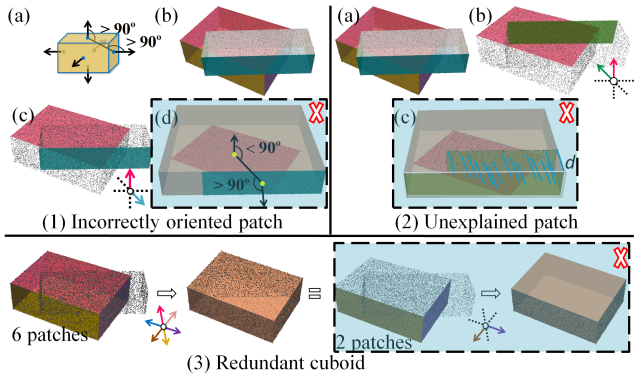

Figure 5: *Hypothesis filtering based on three different criteria. (1) We present the right oriented case in (a); two patches extracted from (b) as shown in (c) can generate a cuboid with the incorrect orientation (d). (2) The cuboid in (c) constructed by two patches in (b) extracted from (a) is rejected, since one of the related patches cannot be explained by the final cuboid. (3) The cuboid constructed by six patches (the left) is similar to the cuboid constructed by two patches (the right) and thus just one of the two is kept. All hypotheses with the light blue backgrounds are rejected.*

on a permutation and combination technique, which appropriately eliminates the uncertainty in cuboid construction. Considering the combinations with no discrimination in the treating of chosen patches, the number of initial cuboid hypotheses could be huge, which consequently poses high solution complexity on the following CSG construction. Therefore, we design three criteria to filter out unreasonable and redundant hypotheses.

Planar patch grouping. Let $\mathbf{Pl} = \{\mathrm{pl}_i\}_{i=1}^{n}$ be the set of all planar patches fitted from the input data and $\mathbf{N} = \{\mathbf{n}_i\}_{i=1}^{n}$ the corresponding normal vectors. $\mathbf{Pl}$ is classified into several sets, i.e., $\mathbf{S} = \{s_j\}_{j=1}^{m}$, based on the similarity of corresponding normal vectors. The Euclidean distance between the normals of any two patches in one set should be zero, since the global relations have been applied during surface patch fitting. For each set, the first normal vector is considered as its orientation vector. Accordingly, we construct a graph G, in which each node stands for one set $s \in \mathbf{S}$. For each pair of nodes, the edge is constructed provided that the angle between the corresponding pair of orientation vectors is equal to $\frac{\pi}{2}$. The graph is broken down obeying the ordering from triangle loops to isolated edges to scattered nodes. Each separated component represents one group. Figure 4 presents the grouping process. The graph in (c) is partitioned into three groups, which contain three nodes, two nodes and one node respectively.

Cuboid hypothesis generation. For each group c, we first determine its symbol $(\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z, -\mathbf{n}_x, -\mathbf{n}_y, -\mathbf{n}_z)$ as follows:

$$(\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z) = \begin{cases} (\mathbf{v}_{s_i}, \mathbf{v}_{s_j}, \mathbf{v}_{s_k}) & \text{if } c = \{s_i, s_j, s_k\}; \\ (\mathbf{v}_{s_i}, \mathbf{v}_{s_j}, \mathbf{v}_{s_i} \times \mathbf{v}_{s_j}) & \text{if } c = \{s_i, s_j\}; \\ (\mathbf{v}_{s_i}, \mathbf{v}^*, \mathbf{v}_{s_i} \times \mathbf{v}^*) & \text{if } c = \{s_i\}. \end{cases} \quad (2)$$

where $\mathbf{v}_{s_i}$, $\mathbf{v}_{s_j}$ and $\mathbf{v}_{s_k}$ are the orientation vectors of the nodes s_i, s_j and s_k in c, respectively. For $c = \{s_i\}$, the minimum bounding rectangle of a patch in s_i is computed and we define the direction vector of the longer edge of the rectangle as $\mathbf{v}^*$.

Subsequently, we generate the cuboid hypotheses in each group based on planar patches. Given a group c, let $\mathbf{Pl}' = \{\mathrm{pl}_i\}_{i=1}^{tn}$ be the set of the associated planar patches, $\mathbf{N}' = \{\mathbf{n}_i\}_{i=1}^{tn}$ the corresponding normal vectors of patches, which are distributed along its symbol (see the second row of Figure 4). Suppose that the number of patches (from now on just called "selections") along six directions (its symbol) are n_1, n_2, …, n_6, respectively. Apparently, $tn = \sum_{i=1}^{6} n_i$. We increase the number of selections for the i-th direction by 1, namely, $(n_i + 1)$, where "1" means no patch is chosen from this direction. By randomly choosing one selection from each direction, we are able to construct the bounding box, which is considered as a cuboid hypothesis. Therefore, by enumerating the combinations of chosen selections, the number of cuboid hypotheses $\mathcal{H}_\mathbf{C}$ generated can be theoretically defined as:

$$|\mathcal{H}_\mathbf{C}| = \prod_{i=1}^{6}(n_i + 1) + tn \quad (3)$$

As known, given a group of chosen patches, a number of bounding boxes (namely, the cuboid hypothesis) can be generated with distinct orientations. In our context, we enforce the unique bounding box to be aligned to the group symbol. In particular, when there is only one planar patch chosen, the height of the bounding box is set as a certain value (usually the maximum length of the selected planar patch) and the opposite bounding box is simultaneously constructed for this special case, which is why we add tn in Equation (3). In this manner, for each group, the corresponding cuboid hypotheses are generated accordingly.

Cuboid hypothesis filtering. We design three criteria to filter out unreasonable and redundant hypotheses. All retained primitives are collected into **plPr**. The filtering is discussed below:

- For a valid cuboid, let $\mathbf{ctr}_1$, $\mathbf{ctr}_2$ be the centers of two adjacent

patches f_1, f_2 and $\mathbf{n}_1$, $\mathbf{n}_2$ their corresponding normals respectively. Then, $\mathbf{n}_1 \cdot (\mathbf{ctr}_2 - \mathbf{ctr}_1) < 0$ and $\mathbf{n}_2 \cdot (\mathbf{ctr}_1 - \mathbf{ctr}_2) < 0$ hold. Therefore, for each pair of patches, the vectors formed by their centers are built followed by computing the angles between the center vectors and the respective normals. If any one angle is less than $\frac{\pi}{2}$, the hypothesis would be rejected. See Figure 5(1).

- The cuboid hypothesis should be well explained by the chosen planar patches, which means that the points of all patches should be close to the hypothesis. Accordingly, we check the distances from the points of patches to the generated cuboid. If the average distance is larger than the minimum length of a voxel calculated in Section 4, the hypothesis is rejected. See Figure 5(2).
- There are a number of redundant cuboid hypotheses. A new hypothesis should be rejected when a subset of its planar patches has already constructed a cuboid, or its planar patches belong to a subset of another cuboid's planar patches. See Figure 5(3).

The primitive hypothesis filtering process is not equal to the redundancy elimination in the work of [RV88], which is designed on the basis of existing CSG models. They employed the Boolean operation information between any two primitives to accelerate redundancy elimination. However, there is no explicit Boolean operation information between two original primitives in our filtering phase and thus the redundancy elimination cannot be used in our method.

3.2. Other Primitives Generation

Quadratic primitives. For the cylinder, cone, sphere and torus, the fitted patches represent their shapes well. Therefore, it is more straightforward to generate the corresponding bounded quadratic primitives than the cuboid from planar patches as discussed above.

Additional primitives. Furthermore, the primitives detected above are not always sufficient to describe the object with a constructive approach. It is sometimes necessary to introduce *additional primitives* that have no direct relationship with fitted patches. The idea of introducing these primitives in order to describe an object by a CSG expression was introduced by the work of [SV91], in which we observe that the demand for additional primitives in 2D space just depends on quadratic curves. Inspired by this work, for each fitted quadratic patch except for one special case, we construct its minimum oriented bounding boxes. For a quarter of cylinder surface (the special case), its maximum bounding box should be constructed. These are all marked as *additional primitives* for our CSG construction. Figure 6 demonstrates the generation of quadratic primitives as well as corresponding *additional primitives*.

For these generated quadratic primitives and *additional primitives*, we discard redundant ones (e.g., cylinders with similar radius, orientations and positions are filtered to keep one of them). An *additional primitive* that is equal to the minimum bounding box of corresponding quadratic primitive should be removed as well. All remaining primitives are collected into a set, denoted as **quPr**.

4. CSG model construction

We detail our framework for CSG model $\mathcal{M}$ construction as well as CSG tree $\mathcal{T}$ recovery, based on the primitive candidate set $\mathbf{Pr} = \mathbf{plPr} \bigcup \mathbf{quPr}$ from Section 3. Our algorithm is designed to choose an optimal set of CSG primitives and Boolean operators, representing a global alignment with raw scan. To achieve the goal, we propose a bottom-up solution consisting of two phases, CSG subtree construction and CSG tree construction, obeying the general order for CSG tree recovery from the subtrees to the entity.

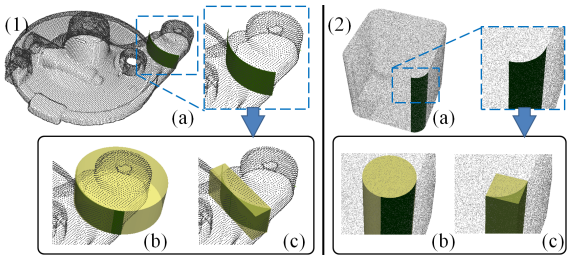

Figure 6: *Quadratic primitive and additional primitive generation. (1) A general case: for a quadratic patch (a), its bounded quadratic primitive (b) and its minimum bounding box (c) are constructed to facilitate the subsequent construction. (2) A special case: for a quarter of cylinder surface (a), its quadratic primitive (b) and its maximum bounding box (c) are constructed, respectively.*

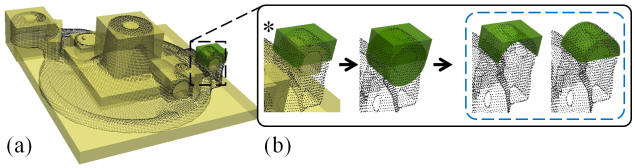

Figure 7: *Candidate set expansion. For an additional primitive extracted from (a), we first take out its quadratic primitive and then two indirect candidates can be derived based on difference and intersection operations respectively between the two primitives (b).*

4.1. CSG Subtree Construction

Given a set of primitive candidates, we generate all possible subtree candidates in this stage. Specifically, some primitive candidates are truncated to form subtrees by using a binary optimization formulation. The function is designed to minimize the truncation error and the complexity of final subtree simultaneously.

Candidate set expansion. A large subtree consists of several small components. Two types of components are used here. One is a primitive directly from **Pr**; the other is constructed by a Boolean operation between two primitives from **quPr**, known as *indirect candidate*. In this stage, we aim at preparing this type of candidate.

For each *additional primitive* $\mathbf{apr}_i \in \mathbf{quPr}$ derived in Section 3.2, we extract its quadratic primitive $\mathbf{qupr}_i$. Then, two *indirect candidates* can be obtained by $(\mathbf{apr}_i - \mathbf{qupr}_i)$ and $(\mathbf{apr}_i \bigcap \mathbf{qupr}_i)$, respectively. Once all *additional primitives* are handled, we add all *indirect candidates* into the candidate set **Pr**. Figure 7 shows an example of the *indirect candidate* generation.

Subtree root selection. All Boolean operators adopted in the subtree construction are supposed to be *intersection* and *difference*. In such a case, the order of operations does not affect a construction

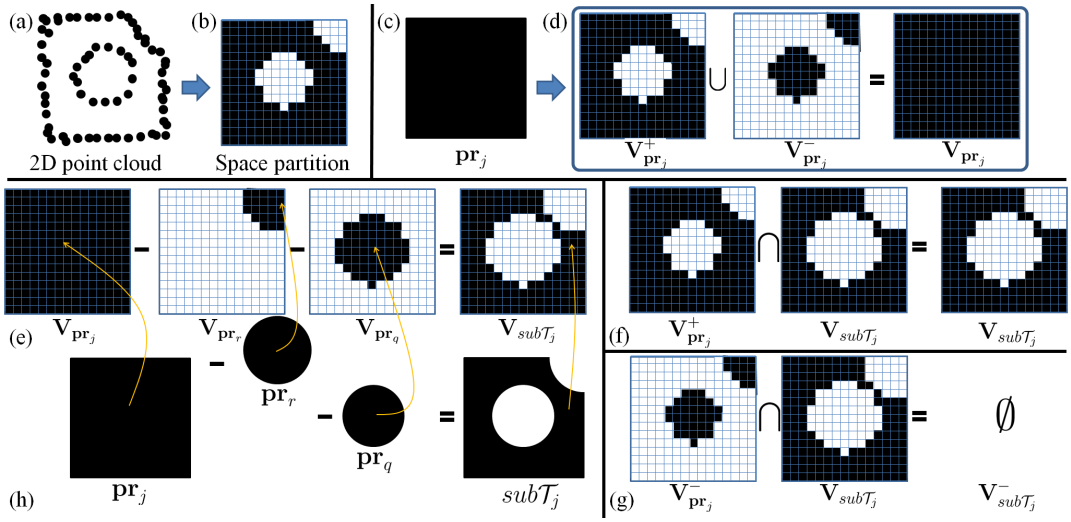

Figure 8: *2D CSG subtree construction. Given a 2D point cloud (a), we first subdivide the bounding rectangle into a set of grids and each grid is indexed and labeled internal or external in (b). Grids colored black mean the indexes of these grids are collected into the description set. For the root primitive* $\mathbf{pr}_j$ *in (c), three grid representations for its corresponding description sets are presented (d). An optimal solution for the root truncation is shown in (e). We demonstrate that it can maintain considerable internal space (f) and meantime diminish the external space (g). At last, we perform those involving geometry Boolean operations among related primitives to achieve the faithful construction (h).*

result, only if the root (start primitive) of the subtree is determined previously. The root is the primitive to be truncated and hence is characterized by possessing space out of the input scan. Let $\mathbf{P}$ be the raw scan. We first employ the normal vector distribution to recognize the internal and external space of $\mathbf{P}$.

We build the minimum bounding box of $\mathbf{P}$, which includes all of the points in $\mathbf{P}$ and also possesses the minimum volume. We subdivide the bounding box into a set of voxels in 3D, denoted as $\mathcal{V}$. The resolution of $\mathcal{V}$ should be determined, when both the efficiency and the construction accuracy are considered. Higher resolution means more computation cost and more sensitivity to noise. Lower resolution means higher modeling error. Based on a series of experiments, the voxel resolution is fixed to $64 \times 64 \times 64$, unless explicitly stated otherwise. The internal or external property of each voxel is determined by the input data. Specifically, for each voxel, we calculate its center c, and search its K (e.g., $K = 50$) nearest neighboring points $\boldsymbol{NP}(c)$ within $\mathbf{P}$. We count the number n_a of the points within $\boldsymbol{NP}(c)$, the normals of which form acute angles with the vectors from the points to the center c. The voxel is considered to be outside of $\mathbf{P}$, if $\frac{n_a}{K} > \varepsilon$; otherwise, it is an inside voxel. The parameter ε is determined by the quality of input data. $\varepsilon = 0.8$ is empirically set and validated by numerous experiments. Note that the boundary voxels are also treated as inside here. In such a way, all voxels within $\mathcal{V}$ are labelled as inside or outside.

For efficiency consideration, in our algorithm, we exploit algebraic operations to replace associated geometry operations. For example, the geometry Boolean operation between two instances can be converted to the associated algebraic Boolean operation between two description sets. For each primitive $\mathbf{pr}_i \in \mathbf{Pr}$ without an *additional primitive* sign, we extract three voxel index sets, $\mathbf{V}^+_{\mathbf{pr}_i}$, $\mathbf{V}^-_{\mathbf{pr}_i}$ and $\mathbf{V}_{\mathbf{pr}_i}$, as its three description sets. The first records all the inside voxel indexes of $\mathcal{V}$ which are surely situated in $\mathbf{pr}_i$, while the second represents all the outside voxel indexes of $\mathcal{V}$ situated in $\mathbf{pr}_i$ as well. $\mathbf{V}_{\mathbf{pr}_i} = \mathbf{V}^+_{\mathbf{pr}_i} \bigcup \mathbf{V}^-_{\mathbf{pr}_i}$ collects all voxel indexes of $\mathcal{V}$ inside $\mathbf{pr}_i$. Thus, the number of inside voxels of $\mathcal{V}$ within it, denoted as $|\mathbf{V}^+_{\mathbf{pr}_i}|$, and the number of outside voxels of $\mathcal{V}$ within it, denoted as $|\mathbf{V}^-_{\mathbf{pr}_i}|$, can be derived. If the scale of the primitive $\mathbf{pr}_i$ exceeds $\mathcal{V}$, we record $\mu_{\mathbf{pr}_i} = 1$; $\mu_{\mathbf{pr}_i} = 0$ otherwise. Primitives with positive $|\mathbf{V}^+_*|$, zero-value $|\mathbf{V}^-_*|$, and zero-value μ_* are directly gathered in a set $\mathbf{PPr}$, which can directly act on the whole CSG tree construction and never need to be truncated to form a subtree. Primitives with positive $|\mathbf{V}^+_*|$, positive $|\mathbf{V}^-_*|$, and zero-value μ_*, should be truncated here. We collect these into a subtree root set, denoted as $\mathbf{RPr}'$. Since the ultimate goal of truncation for a primitive is a subtree without outside voxels of $\mathcal{V}$, to avoid redundancy which means the subtree structure is similar to a primitive in $\mathbf{PPr}$, we should update the subtree root set by $\mathbf{RPr} = \{\mathbf{pr}_i \in \mathbf{RPr}' | \forall \mathbf{pr}_t \in \mathbf{PPr},\ \mathbf{V}^+_{\mathbf{pr}_i} \neq \mathbf{V}_{\mathbf{pr}_t}\}$.

Subtree construction. Each root in our algorithm can determine a subtree. For each root primitive $\mathbf{pr}_j \in \mathbf{RPr}$, we first compute its neighbor set $\Psi_{\mathbf{pr}_j} = \{\mathbf{pr}_k | \mathbf{V}_{\mathbf{pr}_k} \bigcap \mathbf{V}_{\mathbf{pr}_j} \neq \varnothing, \mathbf{pr}_k \in \mathbf{Pr}\}$, which is used as the candidate components for the construction of subtree $subT_j$. Starting with $subT_j = \mathbf{pr}_j$, the algorithm chooses from a number of discrete actions $\{\mathbf{a}_i\}$ to grow the subtree. An action consists of intersecting or subtracting a CSG primitive $\mathbf{pr}_k \in \Psi_{\mathbf{pr}_j}$ with or from $subT_j$, represented as $\odot_i \mathbf{pr}_k$, where $\odot_i$ is the Boolean operator for $\mathbf{pr}_k$ and $\odot_i \in \odot = \{\bigcap, -\}$. Thus, $L_j = |\odot| \times |\Psi_{\mathbf{pr}_j}|$ actions are derived for $subT_j$, where $|\cdot|$ denotes the cardinality of a set.

The goal is to choose an optimal subset of the actions to generate a subtree candidate for the CSG tree construction. We formulate the action selection as a binary optimization, since the sequence of

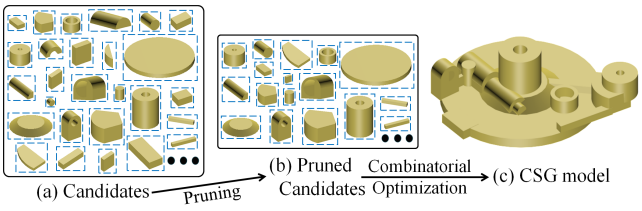

Figure 9: *CSG tree construction. After combining subtrees with original primitives, a large candidate set is obtained (a). We first prune the set to filter out nonsignificant ones (b) and then construct the final CSG model through a binary optimization (c).*

actions does not affect the ultimate subtree. Given L_j valid actions, let $\mathbf{X}_j$ denote the binary labels for all the actions and $x_i \in \mathbf{X}_j$ correspond to the binary option of action $\mathbf{a}_i$, namely $x_i \in \{0,1\}$. The solution to a subtree construction is a subset of actions that minimize the truncation error and the construction complexity simultaneously. Our objective function is as shown below:

$$E(\mathbf{X}_j) = \frac{1}{L_j}\sum_{i=1}^{L_j} x_i - \alpha \cdot \frac{|\mathbf{V}^+_{\mathbf{pr}_j} \bigcap \mathbf{V}_{sub\mathcal{T}_j}|}{|\mathbf{V}^+_{\mathbf{pr}_j} \bigcup \mathbf{V}^-_{sub\mathcal{T}_j}|} \tag{4}$$

where $sub\mathcal{T}_j = \mathbf{pr}_j(x_i\mathbf{a}_i)_{i=1}^{L_j}$, $\mathbf{V}_{sub\mathcal{T}_j} = \mathbf{V}_{\mathbf{pr}_j}(x_i \odot_i \mathbf{V}_{\mathbf{pr}_i})_{i=1}^{L_j}$, $\mathbf{V}^-_{sub\mathcal{T}_j} = \mathbf{V}_{sub\mathcal{T}_j} \bigcap \mathbf{V}^-_{\mathbf{pr}_j}$ and $(\cdot)_{i=1}^{L_j}$ is defined to list all different versions of the variables in the bracket. We define $x_i \odot_i \mathbf{V}_{\mathbf{pr}_i}$ as:

$$x_i \odot_i \mathbf{V}_{\mathbf{pr}_i} = \begin{cases} \odot_i \mathbf{V}_{\mathbf{pr}_i} & \text{if } x_i = 1; \\ \bigcup \emptyset & \text{if } x_i = 0. \end{cases} \tag{5}$$

In Equation 4, the first term discourages the number of actions being chosen, with a lower value representing a lower complexity of the final subtree; the second term encourages the choice of actions that diminish the outside voxel index set of $\mathbf{pr}_j$ to reduce the denominator, while maintaining its inside voxel index set to make the numerator as large as possible. α is a weight parameter that balances the two terms, which is empirically set to 0.9 for all the examples in this paper. Figure 8(h) presents a solution to truncate the root primitive $\mathbf{pr}_j$ in (c). We replace geometry Boolean operation between any two bounded primitives with the algebraic Boolean operation between corresponding description sets (e). The result $\mathbf{V}_{sub\mathcal{T}_j}$ should be consistent with the initial space partition of the root in (b) to achieve a faithful construction. Therefore, it is required to keep internal space (f) and reduce external space (g).

We minimize the above energy function by using the BDD-based heuristics method [BCvHY14]. After the energy is minimized, the variables in $\mathbf{X}_j$ with value 1 suggest the subset of candidate actions that are chosen to approximate the underlining structure of the subtree. As a result, by performing the above truncation for each primitive in **RPr**, we can construct all potential subtree candidates $sub\mathcal{T}$ for the final CSG model.

4.2. CSG Tree Construction

The primitive candidate set **PPr** and the subtree candidate set $sub\mathcal{T}$ are combined here to provide candidates for the final CSG tree construction, denoted as $\mathcal{C} = \{\mathcal{C}_i\} = \mathbf{PPr} \bigcup sub\mathcal{T}$. In this stage, the CSG tree is constructed by a series of union operations among a subset of the candidates and thus the ordering of these candidates is non-significant. On this basis, we can still formulate the construction as a binary optimization problem.

Candidate set pruning. The number of the candidates is usually large and it is necessary to filter out those candidates that clearly do not contribute to the final construction. We observe that a large portion of the candidates still possess outside voxels of $\mathcal{V}$. These candidates are not necessary and may affect the modeling accuracy, when they are chosen as components of final CSG representation, since only *union* operator should be used in this stage. This observation motivates us to identify and remove these candidates. To this end, for each candidate $\mathcal{C}_i \in \mathcal{C}$, we compute its outside voxel index set $\mathbf{V}^-_{\mathcal{C}_i}$. $|\mathbf{V}^-_{\mathcal{C}_i}| > 0$ means candidate $\mathcal{C}_i$ is obviously unwanted and discarded. Hence, only these candidates consisting of inside voxels of $\mathcal{V}$, will be taken as input in the later construction step.

CSG tree construction. The goal in this stage is to choose an optimal subset of the candidates to assemble a compact 3D polygonal model for the product described by the point cloud. Given the pruned candidate set $\mathcal{C}$, suppose $M = |\mathcal{C}|$, $\mathbf{Z}$ represents the binary labels for all the candidates and $z_i \in \mathbf{Z}$ relates to the i-th candidate's binary option, our energy function is formulated as follows:

$$E_{final}(\mathbf{Z}) = \frac{1}{M}\sum_{i=1}^{M} z_i - \beta \cdot \frac{|\mathbf{V}^+_{\mathbf{P}} \bigcap \mathbf{V}_{\mathcal{M}}|}{|\mathbf{V}^+_{\mathbf{P}}|} \tag{6}$$

where $\mathbf{V}_{\mathcal{M}} = \bigcup_{i=1}^{M} z_i\mathbf{V}_{\mathcal{C}_i}$ and $z_i\mathbf{V}_{\mathcal{C}_i}$ is defined as:

$$z_i\mathbf{V}_{\mathcal{C}_i} = \begin{cases} \mathbf{V}_{\mathcal{C}_i} & \text{if } z_i = 1; \\ \emptyset & \text{if } z_i = 0. \end{cases} \tag{7}$$

Similarly, the first term restrains the number of selected candidates, which is designed to control the complexity of the final CSG model. The second term favors choosing candidates that result in more inside voxels of $\mathcal{V}$, which directly determines the construction accuracy. $\beta = 0.9$ is empirically set to balance the two terms, and is validated by numerous experiments. We minimize the above function to acquire an optimal subset of candidates. At last, the CSG tree $\mathcal{T}$ can be constructed based on the subset of candidates. For the mechanical part scan in Figure 6(a), Figure 9 depicts the procedures of final CSG construction.

CSG model construction. The set Boolean operations above exactly correspond to geometry Boolean operations. Thus, we perform those involving geometry Boolean operations among related primitives, obeying the order from subtree structures to the entire tree structure to construct the CSG model $\mathcal{M}$.

5. Results and Discussions

We evaluate the performance of our method on a variety of raw point clouds with different complexities and styles.

5.1. Experimental Evaluation

5.1.1. Raw data with various complexities

The evaluation starts with a simple example in Figure 10. Eight cylinders, two cones, one sphere and three cuboids are selected for

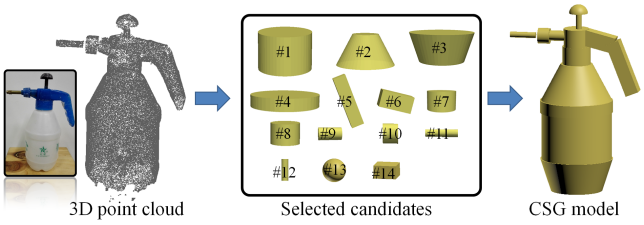

Figure 10: *CSG construction for the spray bottle model. From left to right: the input point cloud, the optimal primitive candidate subset, and the constructed model.*

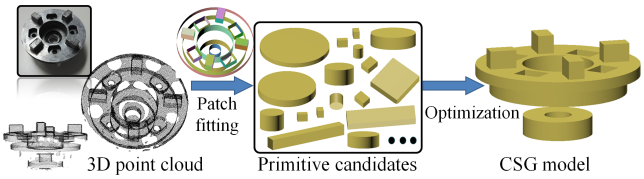

Figure 11: *CSG construction for a mechanical part. From left to right: the scan, the primitive candidates, and the CSG model.*

the construction of the spray model. Given the point set, our method constructs the CSG model as demonstrated and evolves the expression that combines the CSG primitives and Boolean operations, i.e.:

$$\mathcal{T} = Cyl_1 \cup Cone_2 \cup Cone_3 \cup Cyl_4 \cup Cub_5 \cup Cub_6 \cup Cyl_7 \cup Cyl_8 \cup Cyl_9 \cup Cyl_{10} \cup Cyl_{11} \cup Cyl_{12} \cup (Cub_{14} \cap Sphere_{13}) \quad (8)$$

Given the scan with poor quality in Figure 11, due to the material color (we were not allowed to paint the model white), only a limited number of surface patches are extracted by GlobFit [LWC*11]. With these patches, our method automatically generates all related bounded primitives and then produces a construction with our proposed bottom-up solution. The result is faithful to the input scan, but contrasts with the real model presented in the image. It is unacceptable since several structures are missing. In Figure 12, the scene is scanned by a Microsoft Kinect scanner with significant structured noise and data deficiency. The result of GlobFit [LWC*11] consists of some surface patches, based on which our method can infer missing information to some extent and achieve the compact construction. Figure 13 gives a fairly challenging example. The structure of the mechanical part is complex. It contains a large number of fitting patches, which inevitably increases the scale of CSG primitive candidates. Among the great variety of CSG primitives, our approach succeeds in constructing the accurate CSG model as well as the CSG tree.

5.1.2. Comparisons

To further evaluate the performance of our method, we first compare our CSG construction algorithm to some ground truth (GT). We artificially sample points from CSG models, which are used as the input to our algorithm. In Figure 14, from left to right, we present the ground truth and our final CSG construction, respectively. As can be seen, the simple CSG model can be constructed

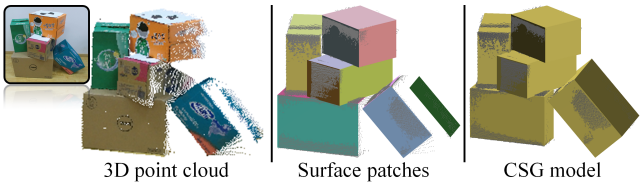

Figure 12: *CSG construction for the noisy scene characterized by incomplete or corrupted surface patches.*

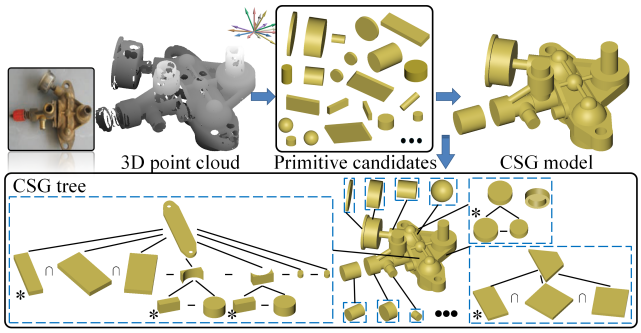

Figure 13: *CSG construction for the challenging mechanical part, of which the structure is fairly complex with a host of primitive candidates. We succeed in constructing the CSG model.*

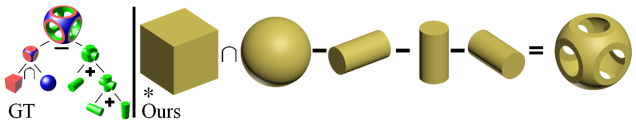

Figure 14: *CSG construction comparison to ground truth (GT). Based on the same primitives as GT, the CSG model is generated at our subtree construction phase.*

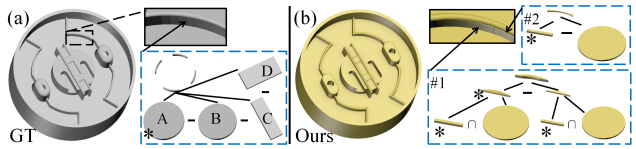

Figure 15: *CSG construction comparison to ground truth (GT). (a) Four primitives with three Boolean operations are exploited to construct the inwall sections. (b) Without these critical primitives C and D in (a), our method constructs every inwall section by combining subtree structures* #1 *and* #2.

with different processes based on the same primitives. Figure 15 demonstrates the CSG constructions with different primitives. The ground truth is provided by an expert user and there are several differences between the two constructions. We present the inwall construction difference in Figure 15. As shown, in GT, four inwall sections are simultaneously built by three Boolean operations among four primitives. Our construction is more complicated, since some primitives, such as primitives C and D in (a), cannot be detected in our method. However, based on our extracted primitives, the CSG model can still be successfully constructed in (b).

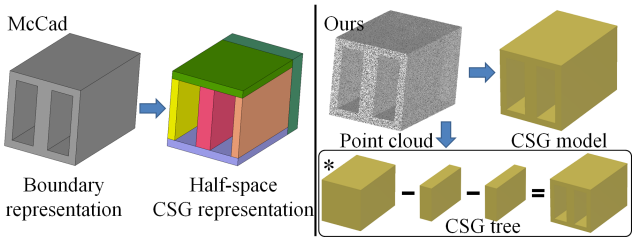

Figure 16: *CSG construction comparison to McCad. Given a B-rep model, the McCad can convert it to a half-space CSG model (the left). We sample the B-rep model and get the points. On this basis, our method constructs the CSG model and CSG tree (the right).*

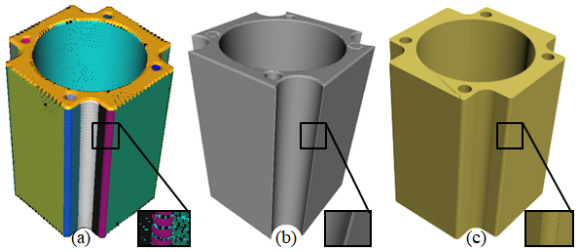

Figure 17: *CSG construction comparison to [FP16]. (a) The blend features are properly identified and fitted. (b)The result of [FP16] does not correctly construct all these features. (c) Our method presents the faithful CSG construction.*

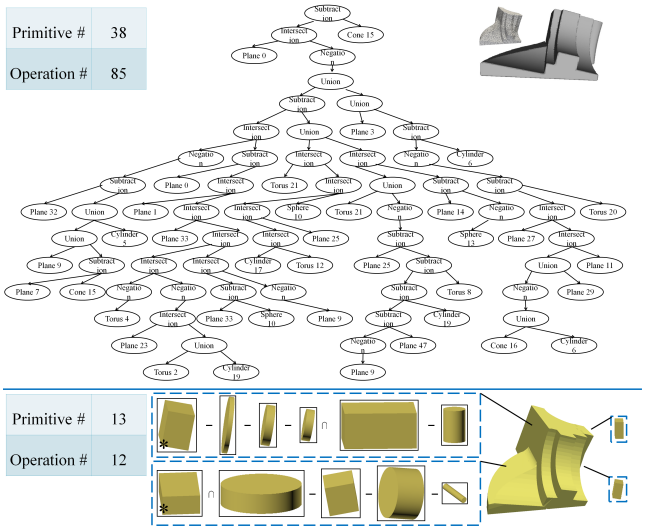

Figure 18: *CSG construction comparison to [FP16]. The tree extracted by [FP16] consists of* 38 *surface primitives with* 85 *modeling operations (the top). Our CSG tree is shown at the bottom, with just* 13 *primitives and* 12 *Boolean operations.*

Figure 16 demonstrates the comparison with the B-rep to half-space CSG conversion process. We choose McCad, a famous geometry conversion tool to automatically convert CAD models into half-space CSG representations. The input is a B-rep model and by the decomposition command, we obtain the CSG model which is constructed by the combination of six boxes and each box is constructed by a series of Boolean operations among its half-spaces. To compare with our method, we virtually scan the B-rep model into a point cloud. Taking the point cloud as input, our algorithm constructs the complete model and recovers the CSG tree simultaneously. As shown, our result contains two subtraction operations among three cuboids, which is more concise than direct B-rep to half-space CSG conversion. In the B-rep to half-space CSG conversion, the number of detected half-spaces is invariably more than the number of surface patches of a B-rep, even if the CSG minimization is considered. We construct bounded primitives based on surface patches and thus the number of primitives for final representation is less than the number of surface patches; in addition, the CSG minimization is carefully considered in our method.

Figures 17 and 18 both present the comparisons with the latest extraction method of object construction tree [FP16]. In Figure 17, two construction results are presented, based on the fitting result from [FP16]. As can be seen, cylinders with small radii are recognized and fitted, but they are not used in [FP16]. With a higher voxel resolution ($128 \times 128 \times 128$), our method constructs the feature correctly. In Figure 18, the tree of the Fandisk model from [FP16] is quite large with 85 modeling operations. It contains considerable redundant information (e.g., plane 0, plane 25 and cylinder 19 all appear twice in the tree), which makes it unclear how useful the recovered tree is for further modification and editing. Based on fitted patches, our algorithm generates all possible bounded primitives and our final CSG tree consists of only 13 primitives (including an *additional primitive*) with 12 Boolean operations, which is relatively simple and contains no redundant information. Furthermore, their method directly proceeds on the basis of *surface* patches with an evolutionary algorithm and the running time for the special part is long. In contrast, our computation cost can be decreased to dozens of seconds. The recovered CSG tree consists of bounded primitives and Boolean operations, which appropriately represents the design intent of the model and facilitates the downstream applications, such as creative editing and redesign.

5.2. Quantitative Evaluation

The results shown above have visually demonstrated the superiority, in terms of effectiveness on various types and complexities of the models. We provide some quantitative evaluations in Table 1.

RMSE. We compare our CSG trees with other construction trees generated by seven users (students and post-docs in our university). Let $\mathbf{P}$ be the input raw point set and $\mathcal{M}$ the respective modeling result. To measure the geometric fidelity of various modeling results to the input data, the root-mean-square error (*RMSE*) is employed here. The CSG model is constructed by a subset of fundamental primitives combined with the corresponding Boolean operations. For each primitive $\mathbf{pr}_i$ in the subset, the associated point subset is denoted by $\mathbf{P}_i$, in which each point is closer to $\mathbf{pr}_i$ than any other primitive. l denotes the number of primitives in the subset. The

Table 1: *Quantitative evaluation shows the fidelity and the complexity of the construction results generated by seven users and our automatic algorithm. We report the* RMSE/Complexity *value in each cell of the table.*

Data sets	*RMSE*(m)/*Complexity*							
	Ours	U1	U2	U3	U4	U5	U6	U7
Figure 2	**0.15/0.88**	0.20/0.88	0.18/0.94	0.25/1.0	0.24/1.0	0.16/0.88	0.18/0.94	0.18/0.88
Figure 11	0.24/**0.42**	0.36/1.0	0.30/1.0	0.25/1.0	**0.23**/1.0	0.37/1.0	0.32/1.0	0.31/1.0
Figure 13	**0.37/0.87**	0.39/1.0	0.45/0.95	0.39/0.96	0.42/1.0	0.41/0.96	0.48/0.96	0.49/1.0
Figure 18	**0.35/0.85**	0.36/1.0	0.40/0.85	0.38/0.85	0.35/1.0	0.42/1.0	0.42/0.85	0.38/1.0

modeling error metric is then defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{l}\sum_{\forall p\in\mathbf{P}_i} dist^2(p,\mathbf{pr}_i)}{\sum_{i=1}^{l}|\mathbf{P}_i|}} \tag{9}$$

where $dist(p,\mathbf{pr}_i)$ is the Euclidean distance between p and $\mathbf{pr}_i$; $|\mathbf{P}_i|$ is the number of points within $\mathbf{P}_i$.

Complexity. For a certain input $\mathbf{P}_{(i)}$, we have eight construction schemes (one is from our algorithm and the others are from seven users). We record the number of primitives for each scheme, denoted as $N^j_{\mathbf{P}_{(i)}}$, where $j = 1,2,\cdots,8$. To evaluate the k-th $(1 \le k \le 8)$ scheme, we define the *Complexity* function as:

$$Complexity(\mathbf{P}_{(i)}) = \frac{N^k_{\mathbf{P}_{(i)}}}{\max\{N^j_{\mathbf{P}_{(i)}}\}_{j=1}^{8}} \tag{10}$$

Table 1 presents the comparison results of the eight different schemes in terms of *RMSE* and *Complexity*. We can observe that our method always yields models with high fidelity and the simplest construction process over user-generated schemes.

5.3. Performance

Timing. We have implemented the proposed method in this work in C++, which is executed on a PC with a $i7, 3.40GHz$ processor and $16GB$ RAM. In the algorithm, given the fitted surface patches with associated point sets and normal sets, the computational cost of CSG primitive generation is negligible. For the CSG model construction process, we first formulate the construction into two successive combination optimization problems and then accelerate the searching by employing the binary optimization technique. On the other hand, we substitute simple algebraic Boolean operations for expensive geometry Boolean operations during the optimization which further decreases the running time. Table 2 gives the computational timings of our method on test data. The efficiency of our algorithm depends on the following factors: the voxel resolution during the partition of input points, the complexity of the model, the type and the number of candidate primitives. Experiments with a higher voxel resolution ($128\times128\times128$), such as Figures 15 and 17, require more computations. Simple shapes, such as Figures 10 and 14, can be constructed quickly, since the scales of primitive candidates are small and quadratic primitives are the dominant components during the constructions, which are easier to be generated and truncated. For complex shapes, such as Figures 13 and 15, a larger number of subtree candidates are built during the CSG subtree construction phase. The scales of the corresponding trees are comparatively larger, and therefore the evaluation time is relatively longer.

Table 2: *Timings of our CSG modeling algorithm on inputs of various complexities (in seconds).*

Data sets	Points	Planar patches	Quadratic patches	Time
Figure 2	52,824	20	8	47.6
Figure 9	105,223	20	15	86.4
Figure 10	436,569	9	11	26.3
Figure 11	66,005	36	5	51.7
Figure 13	529,006	24	39	94.3
Figure 14	99,982	6	4	19.5
Figure 15	700,067	39	45	136.9
Figure 17	55,808	14	16	82.1
Figure 18	120,826	7	7	32.4

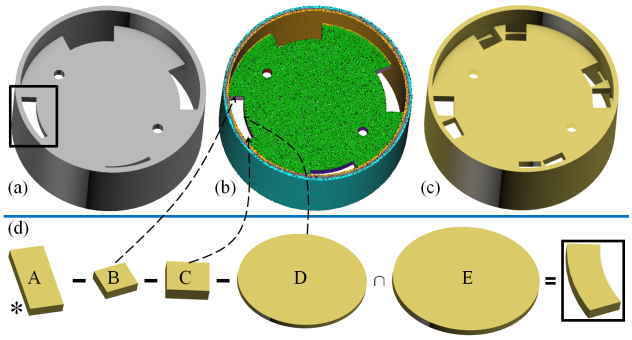

Figure 19: *Pierced CSG model construction over artificial data. We sample points from a CSG model in (a) and present the fitting result in (b). The recovered object in (c) cannot reproduce the pierced feature, since there are no fitted patches supporting the construction of two critical primitives A and E in (d).*

Limitations. There are quite a few limitations to the current algorithm. The quality of patch primitives provided by the algorithm [LWC*11] affects our construction. Figure 12 demonstrates that our framework can infer missing information to some extent; however, the dimensional information is false due to these incomplete or corrupted surface patches. In addition, the orientation of surface patches is exceedingly paramount in our algorithm. With a wrong patch orientation, several solid primitives may be constructed improperly and thus we may obtain an unfavorable construction. Fi-

nally, the algorithm is unable to deal with non-CSG models, like those with free-form surfaces, and some chambers pierced by complicated patterns. In Figure 19, the fitted patches are not enough to support the recovery of pierced patterns and thus our method fails the CSG construction. To derive these significant primitives (e.g., primitives A and E) will be a topic of further investigations.

6. Conclusions

In this research, we present a novel CSG construction framework. The volumetric representation and the corresponding expression tree are critical for engineering applications, like structure and functionality analysis and redesign. Starting with a raw scan, our algorithm automatically generates all attainable bounded primitive candidates. With these primitives, we propose the bottom-up solution to construct a CSG model and a CSG tree, considering both the construction accuracy and complexity. We demonstrate that the constructed models are geometrically consistent with the input data, and the optimal CSG trees can be reliably extracted.

It is promising to regard the results from our modeling system as a starting point for further model modification and editing. On this basis, a number of applications can be beneficial from our CSG construction framework. Since our modeling system heavily relies on primitive fitting, to seek more robust CSG primitive generation scheme would be an interesting direction of our future work.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions. The work was supported in part by NSFC (61772267, 61572507, 61532003, 61622212), the Fundamental Research Funds for the Central Universities (NE2016004).

References

[BC04] BUCHELE S. F., CRAWFORD R. H.: Three-dimensional halfspace constructive solid geometry tree construction from implicit boundary representations. *Computer-Aided Design 36*, 11 (2004), 1063–1073. 3

[BCvHY14] BERGMAN D., CIRE A. A., VAN HOEVE W.-J., YUNES T.: Bdd-based heuristics for binary optimization. *Journal of Heuristics 20*, 2 (2014), 211–234. 8

[BFL*10] BECCARI C. V., FARELLA E., LIVERANI A., MORIGI S., RUCCI M.: A fast interactive reverse-engineering system. *Computer-Aided Design 42*, 10 (2010), 860–873. 2

[BR01] BUCHELE S. F., ROLES A. C.: Binary space partitioning tree and constructive solid geometry representations for objects bounded by curved surfaces. In *CCCG* (2001), pp. 49–52. 3

[CLL*05] CHARIKAR M., LEHMAN E., LIU D., PANIGRAHY R., PRABHAKARAN M., SAHAI A., SHELAT A.: The smallest grammar problem. *IEEE Trans. Inf. Theor. 51*, 7 (July 2005), 2554–2576. 2

[DRD10] DURUPT A., REMY S., DUCELLIER G.: Kbre: a knowledge based reverse engineering for mechanical components. *Computer-Aided Design and Applications 7*, 2 (2010), 279–289. 2

[Fis04] FISHER R. B.: Applying knowledge to reverse engineering problems. *Computer-Aided Design 36*, 6 (2004), 501–510. 2

[FP16] FAYOLLE P.-A., PASKO A.: An evolutionary approach to the extraction of object construction trees from 3d point clouds. *Computer-Aided Design* (2016). 2, 3, 10

[FPK*08] FAYOLLE P. A., PASKO A., KARTASHEVA E., ROSENBERGER C., TOINARD C.: *Automation of the Volumetric Models Construction*. Springer Berlin Heidelberg, 2008. 3

[GG04] GELFAND N., GUIBAS L. J.: Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), ACM, pp. 214–223. 4

[LFP14] LU L., FISCHER U., PERESLAVTSEV P.: Improved algorithms and advanced features of the cad to mc conversion tool mccad. *Fusion Engineering and Design 89*, 9 (2014), 1885–1888. 1, 4

[LMM04] LANGBEIN F. C., MARSHALL A. D., MARTIN R. R.: Choosing consistent constraints for beautification of reverse engineered geometric models. *Computer-Aided Design 36*, 3 (2004), 261–278. 1

[LWC*11] LI Y., WU X., CHRYSATHOU Y., SHARF A., COHEN-OR D., MITRA N. J.: Globfit: Consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (TOG)* (2011), vol. 30, ACM, p. 52. 2, 4, 9, 11

[NSZ*10] NAN L., SHARF A., ZHANG H., COHEN-OR D., CHEN B.: Smartboxes for interactive urban reconstruction. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 93. 2, 3

[RV85] REQUICHA A. A., VOELCKER H. B.: Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE 73*, 1 (1985), 30–44. 1, 3

[RV88] ROSSIGNAC J. R., VOELCKER H. B.: Active zones in csg for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms. *ACM Transactions on Graphics (TOG) 8*, 1 (1988), 51–87. 6

[RvdH04] RABBANI T., VAN DEN HEUVEL F.: Methods for fitting csg models to point clouds and their comparison. In *Proceedings of the 7th IASTED International Conference on Computer Graphics and Imaging, Kauai, HI, USA* (2004), vol. 1719, p. 279284. 2

[SFV*05] SILVA S., FAYOLLE P.-A., VINCENT J., PAURON G., ROSENBERGER C., TOINARD C.: Evolutionary computation approaches for shape modelling and fitting. In *Progress in Artificial Intelligence*. Springer, 2005, pp. 144–155. 3

[SGL*17] SHARMA G., GOYAL R., LIU D., KALOGERAKIS E., MAJI S.: Csgnet: Neural shape parser for constructive solid geometry. *arXiv preprint arXiv:1712.08290* (2017). 3

[Sha01] SHAPIRO V.: A convex deficiency tree algorithm for curved polygons. *International Journal of Computational Geometry & Applications 11*, 02 (2001), 215–238. 1

[SV91] SHAPIRO V., VOSSLER D. L.: Efficient csg representations of two-dimensional solids. *Transactions of ASME, Journal of Mechanical Design 113*, 3 (1991), 292–305. 1, 3, 6

[SV93] SHAPIRO V., VOSSLER D. L.: Separation for boundary to csg conversion. *ACM Transactions on Graphics (TOG) 12*, 1 (1993), 35–55. 3

[SWK07] SCHNABEL R., WAHL R., KLEIN R.: Efficient ransac for point-cloud shape detection. In *Computer graphics forum* (2007), vol. 26, Wiley Online Library, pp. 214–226. 4

[TW05] THRUN S., WEGBREIT B.: Shape from symmetry. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 2, IEEE, pp. 1824–1831. 2

[WFAR98] WERGHI N., FISHER R., ASHBROOK A., ROBERTSON C.: Towards object modelling by incorporating geometric constraints. In *IEEE International Workshop on Model-Based 3D Image Analysis* (1998), pp. 45–53. 2

[WFRA99] WERGHI N., FISHER R., ROBERTSON C., ASHBROOK A.: Object reconstruction by incorporating geometric constraints in reverse engineering. *Computer-Aided Design 31*, 6 (1999), 363–399. 2

[YLC*08] YE X., LIU H., CHEN L., CHEN Z., PAN X., ZHANG S.: Reverse innovative design: an integrated product design methodology. *Computer-Aided Design 40*, 7 (2008), 812–827. 2