A Flexible Pipeline for the Optimization of CSG Trees

Markus Friedrich, Christoph Roch, Sebastian Feld,
Carsten Hahn

Institute for Computer Science
LMU Munich
Oettingenstr. 67
80538 Munich, Germany
{first name.last name}@ifi.lmu.de

Pierre-Alain Fayolle

Division of Information and Systems
The University of Aizu
Aizu-Wakamatsu City
965-8580 Fukushima, Japan

fayolle@u-aizu.ac.jp

ABSTRACT

CSG trees are an intuitive, yet powerful technique for the representation of geometry using a combination of Boolean set-operations and geometric primitives. In general, there exists an infinite number of trees all describing the same 3D solid. However, some trees are optimal regarding the number of used operations, their shape or other attributes, like their suitability for intuitive, human-controlled editing. In this paper, we present a systematic comparison of newly developed and existing tree optimization methods and propose a flexible processing pipeline with a focus on tree editability. The pipeline uses a redundancy removal and decomposition stage for complexity reduction and different (meta-)heuristics for remaining tree optimization. We also introduce a new quantitative measure for CSG tree editability and show how it can be used as a constraint in the optimization process.

Keywords

Evolutionary Algorithms, Quantum Annealing, Geometry Processing, CAD, CSG, Combinatorial Optimization

1 INTRODUCTION

Constructive Solid Geometry (CSG) trees are a powerful representation scheme for 3D geometry and an important building-block of 3D modelling software [Req80]. While the creation, reconstruction and conversion from other representations of CSG trees has been covered in the literature, few works deal with the optimization of a given tree.

The nature of CSG tree expressions as a combination of geometric primitives (so-called halfspaces) with Boolean set-operations (union, intersection, complement) suggests a deeper investigation of related methods from *switching function minimization*. This paper investigates an adaption of already proposed, as well, as newly developed methods for the CSG tree optimization problem and extends them to a robust and flexible pipeline. Furthermore, it introduces the idea that CSG tree optimization does not have to be restricted to the reduction of the tree size but should include improvements to the tree's *editability*.

We consider our CSG tree optimization pipeline as a solution to the following problems: Given a hand-modeled CSG tree with sufficient complexity, users have difficulty keeping track of potentially redundant parts. An automatic, but manually triggered, optimization procedure comes in handy. Furthermore, automatic CSG tree reconstruction methods [FP16, WXW18] might result in trees that are not optimal in size and hard to edit manually. Thus, our method can be beneficial in this scenario as well.

This paper makes the following contributions:

- The description of a comprehensive pipeline for optimizing the *editability* of a CSG expression,

- A novel, sampling-based tree size optimization procedure suitable for Quantum Annealing hardware,

- A recursively defined measure of spatial subtree proximity as an indicator for CSG tree *editability*,

- A multi-objective optimization using a Genetic Algorithm that aims to minimize the CSG expression size and maximize its proximity value in order to improve tree *editability*.

The paper is organized as follows: Section 3 gives basic definitions and concepts. Section 2 provides references to related works, whereas Section 4 defines the problem to solve. Our approach is described in Section 5 and evaluated in Section 6. Section 7 concludes the paper.

2 RELATED WORK

Construction of CSG trees from the Boundary Representation (B-Rep) of a solid was considered by Shapiro and Vossler in [SV91a, SV91b, SV93]. The approach is

based on: identifying a set of halfspaces sufficient for representing the input solid, building a CSG expression by considering all products of halfspaces, or their complement, that are inside the input solid (so-called fundamental products), and minimizing this expression. In particular, [SV91a] considers different approaches for the minimization of two-level expressions, either coming from switching theory [Qui52, MJ56, O'R82], or based on geometric considerations.

Related to these works, Buchele and Crawford [BC04] propose an algorithm for producing a CSG expression from the Boundary Representation of a solid by considering early factoring of dominant halfspaces. Such a factoring should help in limiting the size of the produced CSG expression. Andrews proposes to simplify a CSG expression (obtained from a B-Rep) by removing from each fundamental product cell, (spatially) distant primitives [And13]. The reason is that distant primitives may be viewed by the user as unrelated to a given fundamental product cell, and thus could lead to unintuitive results.

Recently, works on reconstructing a CSG expression from a point-cloud have become popular, see for example [FP16, WXW18] , among others. The approach described in [FP16] tries to minimize the size of the CSG expressions produced by a GA by penalizing large expressions in the objective function. The method proposed in [FFGLP19] is using multi-objective optimization to prevent the growth of the generated CSG expressions. It also uses a decomposition scheme that prevents spatially distant primitives to be used in unrelated CSG sub-expressions.

Other works related to the optimization, or manipulation, of CSG expressions, such as [Ros11], try to improve the rendering time of the model, and do not necessarily help in minimizing the size of the expression or improving its editability.

3 BACKGROUND

3.1 CSG Tree Representation

3.1.1 Formal Representation

We follow the formal definition of CSG trees from Shapiro et al. [SV91a]: Given a solid's point-set S, its boundary ∂S consists of patches of halfspaces H_S. Halfspaces are regular sub-sets of the universal point-set W usually described by signed distance functions (SDF) F_H: $\{x \in \mathbb{R}^3 : F_H(x) = 0\}$. A CSG tree expression Φ (in the following, upper-case Greek letters are used for CSG tree expressions) consists of halfspace literals $\{h_0, h_1, ...\}$ and symbols for regularized set-operations $\{\cup^*, \cap^*, \setminus^*, -^*\}$. Applying Φ to the set of halfspaces H_S results in a CSG representation $\Phi(H_S)$ of S iff $|\Phi(H_s)| = S$, where $|\cdot|$ denotes the point-set induced by a CSG representation.

A CSG representation of S is in disjunctive normal form

(DNF) if it contains a sum ($\cup^*$) of halfspace products ($\cap^*$-combined halfspaces or negated halfspaces, so-called implicants). If each implicant of a DNF expression contains all halfspaces (or their negations), it is in a so-called disjunctive canonical form (DCF). In that case, an implicant is called canonical intersection term (CIT) or fundamental product (FP). For example, the universal set W can be decomposed in 2^n CITs in case of n halfspaces being used.

An implicant Ψ is a so-called prime implicant of S if $|\Psi| \in S$ and the removal of a single halfspace from Ψ results in $|\Psi| \notin S$. A dominant halfspace (DH) $g \in H_S$ is a halfspace for which $S = g \cup^* S$ is always true (g is then also a prime implicant of S).

Where necessary, we use the following abbreviations for CSG expressions (halfspaces and set-operations): $|h_0| \cup^* |h_1| := h_0 + h_1$, $|h_0| \cap^* |h_1| := h_0 \cdot h_1$, $\setminus^* |h_0| := \overline{h_0}$ and $|h_0| -^* |h_1| := h_0 - h_1$.

3.1.2 SDF-based implementation

In order to compute the point-set of a CSG representation (e.g., $|\Phi(H_s)|$) we use an SDF-based approximation with min- and max-functions [Ric73, PASS95, Sha07]:

- Intersection: $|\Phi| \cap^* |\Psi| := \max(F_\Phi, F_\Psi)$

- Union: $|\Phi| \cup^* |\Psi| := \min(F_\Phi, F_\Psi)$

- Complement: $\setminus^* |\Phi| := -F_\Phi$

- Difference: $|\Phi| -^* |\Psi| := \max(F_\Phi, -F_\Psi)$

Here, F_Φ and F_Ψ are the SDFs corresponding to expression Φ and Ψ. We assume, as a convention, that $F < 0$ in the interior of the corresponding solid S.

Note that min- and max-functions are not regularized set-operations in the strict sense [Sha99] but a sufficient approximation for our purposes.

3.2 Metaheuristics for Combinatorial Problems

3.2.1 Genetic Algorithms

Genetic Algorithms (GA) are metaheuristics for solving discrete or continuous optimization problems. The process is inspired by biology and consists of evolving a population of creatures. Each creature represents a candidate solution to the problem. Starting from a randomly initialized population, a GA produces an updated population at each iteration by: a) ranking creatures according to a fitness function, b) generating new creatures by mutating a selected creature from the previous population, or by combining a selected pair of creatures from the previous population, c) selecting a few creatures to be preserved into the next population (elitism). Selection is performed based on the rank of each creature. The process is iteratively repeated until a termination criterion is met.

3.2.2 *Quantum Annealing*

Quantum Annealing (QA) is another metaheuristic for solving (in general, discrete) optimization problems. It is based on quantum physics to find low energy states of a system corresponding to the optimal solution of a problem. The QA algorithm is described by a time-dependent Hamiltonian $\mathscr{H}(t)$:

$$\mathscr{H}(t) = s(t)\mathscr{H}_I + (1 - s(t))\mathscr{H}_P$$

The QA process starts in the lowest-energy state of a so-called initial Hamiltonian $\mathscr{H}_I$. During the annealing process, the problem Hamiltonian $\mathscr{H}_P$ is introduced and the influence of the initial Hamiltonian is reduced (described by $s(t)$, which decreases from 1 to 0). At the end of the annealing process, one ends up in an eigenstate of the problem Hamiltonian, which actually encodes the objective function of the problem. If this transition is executed sufficiently slowly, the probability to find the lowest energy state of the problem Hamiltonian is close to 1, w.r.t the adiabatic theorem [AL18].

To perform QA on D-Wave Systems Quantum Annealing hardware, one needs to encode the problem ($\mathscr{H}_P$) in a so-called Quadratic Unconstrained Binary Optimization (QUBO) problem, which is a unifying model for representing a wide range of combinatorial optimization problems. The functional form of the QUBO the quantum annealer is designed to minimize is:

$$\min \mathrm{x}^T Q \mathrm{x} \quad \text{with } \mathrm{x} \in \{0,1\}^{n_Q}, \tag{1}$$

where x is a vector of binary variables of size n_Q, and Q is an $n_Q \times n_Q$ real-valued matrix describing the relationship between the variables. Given the matrix Q, the annealing process tries to find binary variable assignments to minimize the objective function (Eq. 1).

4 PROBLEM STATEMENT

We focus on the optimization of a CSG tree's *editability*: Given a solid's point-set S, a halfspace set H_S and a CSG tree expression Φ with $|\Phi(H_S)| = S$, find the CSG tree expression Φ_{opt} with the best *editability* which is assumed to be determined by two quantitative metrics:

- **Size:** The amount of literals and operations in Φ_{opt}.

- **Proximity:** The ratio between the number of operations of Φ_{opt} whose operands imply point-sets that overlap to the number of operations with operands that imply disjoint point-sets. This property is defined recursively as follows: Given a node Ψ of Φ_{opt}, either Ψ is a leaf (an halfspace) or it has two children (operands) Ψ_1 and Ψ_2 such that the implied solids intersect (that is: $|\Psi_1| \cap^* |\Psi_2| \neq \varnothing$).

A size-optimal tree has no redundant operands which makes tree modification easier and a tree with a high degree of proximity leads to more predictable behavior when sub-trees are transformed spatially.

5 CONCEPT

The CSG tree optimization process is depicted in Fig. 1: First, redundant sub-expressions are removed (orange, Section 5.1). Then, a recursive decomposition scheme is applied that further shrinks the expression size (grey, Section 5.2). If an unoptimized expression (solid) remains, it is optimized with a separate optimization method (blue or purple, Section 5.3) which results – after another run of the redundancy removal method – in the final optimized CSG tree.
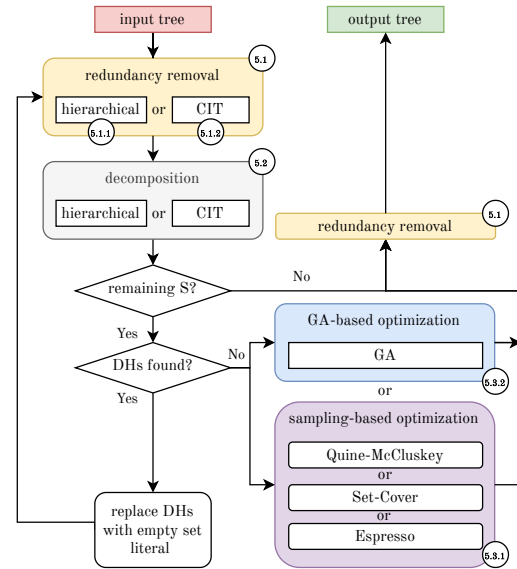

Figure 1: The proposed CSG tree optimization pipeline.

5.1 Redundancy Removal

Our redundancy removal approach is inspired by a method described in [Til84]. It uses the spatial information additionally given by the halfspaces used in the CSG tree (e.g., a sphere's location and radius):

- If the sets described by the operands (halfspaces or subtrees) of an intersection operation do not have elements in common, i.e., the operand sets do not spatially overlap, the expression is replaced with the empty set expression $\varnothing$.

- If the sets described by the operands of a union operation are identical, the expression is replaced with one of the operand expressions.

Empty set $\varnothing$ as well as universal set W expressions are then replaced based on the following rules:

- If one operand of an intersection expression is the empty set $\varnothing$, the expression is replaced with the empty set.

- If one operand of an intersection expression is the universal set W, the expression is replaced with the other operand.

- If one operand of a union expression is the empty set $\varnothing$, the expression is replaced with the other operand.

- If one operand of a union expression is the universal set W, the expression is replaced with W.

In addition, the complement of a complement operation is replaced with the operand of the inner complement operation. The redundancy removal algorithm continuously iterates over the whole CSG tree until no rule applies anymore to the current result.

Especially relevant for the algorithm is the fast and robust evaluation of the empty set and identical set decision algorithms. Since the identical set decision can be expressed as an empty set decision ($|\Phi| = |\Psi| \Longleftrightarrow |\Phi| \cap^* (\backslash^* |\Psi|) = \varnothing \wedge |\Psi| \cap^* (\backslash^* |\Phi|) = \varnothing$), an empty set decision algorithm is sufficient.

We use a sampling-based approach: If the tree represents an empty set, its SDF value is positive (*outside*) in the complete sampling domain. Two different sampling strategies are proposed: hierarchical sampling and CIT-based sampling.

5.1.1   Hierarchical Sampling

This sampling method is an Octree-based, hierarchical sampling of the CSG tree's SDF. The sampling point-set is defined by the width, height and depth of the axis-aligned bounding box (AABB) dimensions of the used halfspaces (w_0, h_0, r_0) and the user-defined minimum sampling cell size $(w_{min}, h_{min}, r_{min})$. The coarse-to-fine hierarchical sampling methodology, as depicted in Fig. 2 allows for early stopping in case of a non-positive SDF value. To further speed-up the process, a lookup-table for already proven empty set expressions is used.
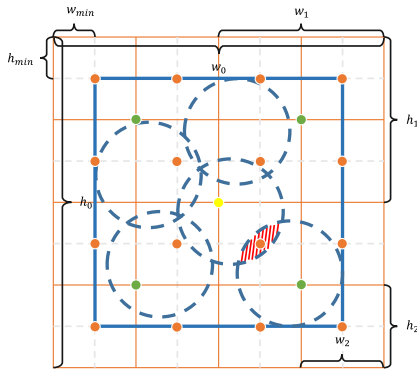

Figure 2: Hierarchical sampling strategy. The space defined by (w_0, h_0, r_0) is consecutively subdivided in sub-quadrants of sizes $s_i = (\frac{w_{i-1}}{2}, \frac{h_{i-1}}{2}, \frac{r_{i-1}}{2}), i = 1, 2, ...$ until $w_i \leq w_{min} \vee h_i \leq h_{min} \vee r_i \leq r_{min}$. The center of each quadrant is the sampling position (yellow, green, orange). In the example, early stopping is possible since the red-hatched area, which marks the non-empty volume, is sampled at quadrant size (w_2, h_2, r_2) which is one level above the smallest possible quadrant size.

5.1.2   CIT-Based Sampling

This sampling method uses a sampling point-set consisting of a point for each CIT located inside the tree.

CITs are retrieved with the method explained in Section 5.3.1. For each point we check if the tree's SDF is positive. The approach can potentially be faster than hierarchical sampling since the used sampling point-set is usually smaller.

5.2   Decomposition

The decomposition of a solid S (that is described by a CSG tree Φ and halfspaces H_S) as proposed in [SV91a] is a tree expression that consists of a chain of halfspaces from H_S that either dominate S or $\overline{S}$ and a (potentially empty) remaining solid S_{rem}:

$$S = |((...(S_{rem} \oplus d_1) \oplus ...) \oplus d_2) \oplus d_n|, \qquad (2)$$

where $\{d_1, ..., d_n\}$ is the set of dominating halfspaces and $\oplus$ is either $+$ if the following halfspace dominates S or $-$ if it dominates $\overline{S}$. This decomposition is a size-optimal tree expression for S since each halfspace appears only once [SV91a]. For example, in Fig. 3, h_0, h_2 dominate $\overline{S}$ and h_3 dominates S, resulting in the decomposition $S = |((h_4 - h_0) - h_2) + h_3|$. Here, S_{rem} contains h_4 and h_1 but since h_1 has no impact on the result, S_{rem} equals h_4 and thus is replaced in the result expression. Please note that we use the example in Fig. 3 differently than in Section 5.3.1: The solid shown is considered S, which is decomposed to get S_{rem}, while in Section 5.3.1 the solid shown is considered S_{rem} which is then further optimized.

Recursive Decomposition. If S_{rem} is not empty, a size-optimal expression for it has to be found. Therefore, a (potentially not size-optimal) expression Φ_{rem} is computed by replacing all appearances of previously found dominating halfspaces in Φ with the empty set literal and applying the redundancy removal method (see Section 5.1). This is followed by another run of the decomposition technique. This recursive process is continued until either the current S_{rem} is empty, or no more dominating halfspaces can be found during decomposition. In the latter case, an optimal expression for S_{rem} has to be found using different methods, see Section 5.3.

Sampling-Based Search for Dominant Halfspaces. For the identification of the dominant halfspaces of S, we propose two different sampling strategies as already described in Section 5.1: Firstly, a hierarchical sampling strategy similar to that proposed in Section 5.1.1 can be used. But instead of taking the whole AABB of S as sampling volume, each halfspace h in H_S is separately tested using its corresponding AABB. If all sampling points inside $|h|$ are inside S as well, h dominates S. If all sampling points inside $|h|$ are not elements of S, h dominates $\overline{S}$. The early-stopping criteria is met if a sampling point is in $|h|$ but not in S. Secondly, a CIT-based sampling strategy as discussed in Section 5.1.2 can be used.

Improving Proximity. Although decomposition can result in size-optimal expressions, the *editability* crite-

rion is also influenced by the proximity metric which is not considered by the proposed decomposition method. To overcome this deficiency, we propose a simple spatial sorting scheme: The halfspaces in the chain of halfspaces (Eq. 2) are arranged such that operands of $\oplus$ do always spatially overlap (if possible).

5.3 Remaining Solid Optimization

After decomposition, there might be a remaining solid $S_{rem} = |\Phi_{rem}(H_{rem})|$ left, for which an optimal expression has to be found. We investigate two optimization methods: Firstly, a sampling-based technique that generates tree expressions in DCF form that are then optimized using well-known two-level logic minimization techniques (Quine-McCluskey [MJ56], Espresso logic minimizer [BHMSV84]) and a new approach based on a QUBO formulation of the Set Cover problem (Section 5.3.1). Secondly, a GA-based method is proposed that uses tree size and the proximity metric as part of its objective function (Section 5.3.2).

5.3.1 Sampling-Based Optimization

Sampling. The AABB of S_{rem} is sampled. For each sample, the tree's SDF is evaluated to decide if the point is *inside*. If it is *inside*, each halfspace SDF in H_{rem} is evaluated as well. If the sampling point is located *inside* the halfspace, the halfspace is part of a CIT. If not, then its complement is. Finally, the CIT is added with a $\cup^*$ operation to the resulting DCF expression Φ_{DCF}. See Fig. 3 for an example.
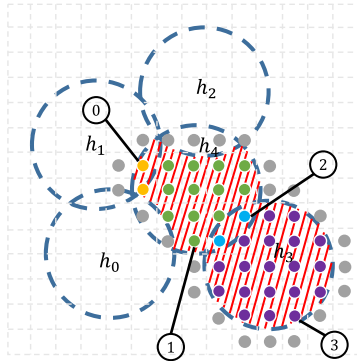

Figure 3: Example of the sampling step. The red-hatched area is the point-set to represent, $\{h_0, ..., h_4\}$ is the halfspace set H_{rem}. The grey dots indicate sampling points outside the solid. The orange dots result in the implicant $\overline{h_0} \cdot h_1 \overline{h_2 \cdot h_3} \cdot h_4$ (0), the green dots in $\overline{h_0} \cdot \overline{h_1 \cdot h_2 \cdot h_3} \cdot h_4$ (1), the light blue dots in $\overline{h_0} \cdot \overline{h_1 \cdot h_2} \cdot h_3 \cdot h_4$ (2) and the purple dots in $\overline{h_0} \cdot \overline{h_1 \cdot h_2} \cdot h_3 \cdot \overline{h_4}$ (3).

DCF Minimization. Besides the already mentioned classic methods for DNF minimization (Quine-McCluskey, Espresso logic minimizer), we propose a third option: First, the prime implicants of Φ_{DCF}, P_{DCF}, are computed by directly applying the definition of a prime implicant (see Section 3) to each CIT

in Φ_{DCF}. Using the example of Fig. 3, the prime implicants would be h_3 and $\overline{h_0} \cdot \overline{h_2} \cdot h_4$ (Note that we do not consider decomposition for this example).

The problem of finding all relevant prime implicants is then formulated as a Set Cover problem: The set to cover, U, is the set of indices of all CITs located inside $|\Phi_{DCF}|$ (in Fig. 3: $U = \{0, 1, 2, 3\}$). Each prime implicant in P_{DCF} covers a subset of U resulting in a collection of subsets V with elements $V_k \subseteq U, 1 \leq k \leq \text{card}(P_{DCF})$ (in Fig. 3: $V = \{h_3 : \{2, 3\}, \overline{h_0} \cdot \overline{h_2} \cdot h_4 : \{0, 1, 2\}\}$). Within the Set Cover problem, one has to find the smallest possible number of subsets from V, such that their union is equal to U. This problem was proven to be NP-hard [Kar72]. In [Luc14] the QUBO formulation for the Set Cover problem is given by:

$$\mathcal{H}_A = A \sum_{\alpha=1}^{\text{card}(U)} \left(1 - \sum_{m=1}^{\text{card}(P_{DCF})} x_{\alpha,m} \right)^2 +$$
$$A \sum_{\alpha=1}^{\text{card}(U)} \left(\sum_{m=1}^{\text{card}(P_{DCF})} m x_{\alpha,m} - \sum_{k:\alpha \in V_k} x_k \right)^2,$$

and

$$\mathcal{H}_B = B \sum_{k=1}^{\text{card}(P_{DCF})} x_k, \qquad (3)$$

with x_k being a binary variable which is 1, if set V_k is included within the selected sets, and 0 otherwise. $x_{\alpha,m}$ denotes a binary variable which is 1 if the number of selected subsets V_k which include element α is $m \geq 1$, and 0 elseways. The first energy term in $\mathcal{H}_A$ imposes the constraints that for any given α exactly one $x_{\alpha,m}$ must be 1, since each element of U must be included a fixed number of times. The second term in $\mathcal{H}_A$ states that the number of times that we declared α was included is in fact equal to the number of subsets V_k we have included with α as an element. A is a penalty value, which is added on top of the solution energy, described by $\mathcal{H} = \mathcal{H}_A + \mathcal{H}_B$, if a constraint was not fulfilled, i.e., one of the two terms are unequal to 0. Therefore adding a penalty value states a solution as invalid. Additionally, the Set Cover problem minimizes over the number of chosen subsets V_k, as stated in (Eq. 3). For a given problem instance, $\mathcal{H}$ is transformed into the QUBO formulation required by QA hardware (Eq. 1) and minimized.

Note that the prime implicant selection via Set Cover is not needed if H_{rem} does not contain halfspaces that fully contain other halfspaces and no separating halfspaces [SV91a] are used. Furthermore, the sampling step can be omitted if the input expression is already in DNF form.

5.3.2 GA-Based Optimization

The methods described in Section 5.3.1 are two-level minimization techniques, which in general do not re-

sult in size-optimal trees [SV91a]. Moreover, other optimization goals like, for example, the proximity metric are not considered. Their main advantages are a possible short execution time and the fact that non-optimality of sufficiently small trees (e.g., after decomposition as explained in Section 5.2) has usually smaller and thus negligible negative effects on tree size and proximity.

In order to find possibly better trees (w.r.t. size and proximity), the corresponding problem is formulated as a combinatorial optimization problem over all possible trees given a set of halfspaces H_{rem} and set-operations. Let Φ_{rem} be the input CSG tree expression that needs to be optimized. A GA is used to solve this optimization problem. Each creature Φ_c in the population of the GA represents a potential CSG tree expression. The same mutation, crossover and selection as in [FP16] are used. Additional details about the GA are provided below.

Initialization. The population is initialized with a mixture of randomly generated trees with a maximum size equal to the size of the input tree Φ_{rem} and copies of the input tree.

Pre-processing. To reduce the computational effort of evaluating the fitness function in the GA, we compute a limited number of sample points. This is done by computing the CITs of Φ_{rem} via sampling as described in Section 5.3.1 with the difference that CITs that are located outside of Φ_{rem} are considered as well. During the sampling process, a point in each CIT is added to the sampling point-set S_{in} if the corresponding CIT is inside of Φ_{rem}. Otherwise, it is added to S_{out}.

Ranking. The fitness of a creature Φ_c corresponding to a given CSG tree is given by

$$f(\Phi_c) = \alpha \cdot f_{geo}(\Phi_c) + \beta \cdot f_{prox}(\Phi_c) + \gamma \cdot f_{size}(\Phi_c),$$

where $f_{geo}()$, $f_{prox}()$ and $f_{size}()$ are the geometric score, the proximity score and the size score, respectively, and α, β and γ are user-defined parameters. The geometric score counts how many points from S_{in} are elements of $|\Phi_c|$ and how many points from S_{out} are not in $|\Phi_c|$:

$$f_{geo}(\Phi_c) = f_{in}(\Phi_c) + f_{out}(\Phi_c),$$

with

$$f_{in}(\Phi_c) = \frac{1}{\text{card}(S_{in})} \sum_{s \in S_{in}} \begin{cases} 1, & \text{if } |F_{\Phi_c}(s)| \leq \varepsilon_p \\ 0, & \text{otherwise} \end{cases}$$

and

$$f_{out}(\Phi_c) = \frac{1}{\text{card}(S_{out})} \sum_{s \in S_{out}} \begin{cases} 1, & \text{if } |F_{\Phi_c}(s)| > \varepsilon_p \\ 0, & \text{otherwise} \end{cases},$$

where $\text{card}(S)$ is the cardinality of the point-set S.

The proximity score tries to enforce that two operands of a Boolean operation are spatially connected. It is an implementation of the proximity metric (Section 4):

$$f_{prox}(\Phi) = \frac{P_{rec}(\Phi)}{\#|\Phi|},$$

where $\#|\Phi|$ is the number of nodes in the tree corresponding to the creature Φ and the function

$$P_{rec}(\Phi) = \begin{cases} 1, & \text{if } \Phi \text{ is a leaf node} \\ P_{rec}(\Phi_1) + P_{rec}(\Phi_2) + \Delta(\Phi_1, \Phi_2) & \text{otherw.} \end{cases},$$

where Φ_1 and Φ_2 are the two children of Φ (if it is not a leaf node), and

$$\Delta(\Phi_1, \Phi_2) = \begin{cases} 1 & \text{if } |\Phi_1| \cap^* |\Phi_2| \neq \varnothing \\ 0 & \text{otherwise} \end{cases}.$$

Finally, the size score tries to favor the simplest (shortest) possible CSG tree and corresponds to

$$f_{size}(\Phi_c) = \frac{\#|\Phi_{rem}| - \#|\Phi_c| - f_{size}^{min}}{f_{size}^{max} - f_{size}^{min}},$$

where $\#|\Phi_{rem}|$ and $\#|\Phi_c|$ are the number of nodes in the CSG trees Φ_{rem} and Φ_c, f_{size}^{min} is the minimum and f_{size}^{max} the maximum tree size within the current iteration's population.

Termination. The GA terminates if either a user-defined maximum number of iterations is reached or the score does not improve over a number of iterations.

6 EVALUATION

6.1 Data Acquisition

We prepared 11 different, hand-crafted CAD models for our experiments (see Fig. 4) with a complexity comparable to models commonly used in CSG tree reconstruction tasks [FP16, DIP+18]. To simulate particular levels of *sub-optimality*, we have implemented a generator, which takes a CSG tree as input and iteratively adds redundant parts at random positions in the tree, based on the following strategies:

Copied Subtree Insertion (CSI): Insert a union or intersection at a random position with both operands being copies of the subtree at that position.

Double Negation Insertion (DNI): Insert two chained negations at a random position.

Distributive Law Insertion (DLI): Apply the distributive laws $A(B+C) = (A \cdot B) + (A \cdot C)$ or $A + (B \cdot C) = (A + B) \cdot (A + C)$ to a random subtree.

Absorption Law Insertion (ALI): Apply the absorption laws $A = A + (A \cdot B)$ or $A = A \cdot (A + B))$ to a random subtree.

GA-based Redundancy Insertion (GRI): Use the GA (Section 5.3.2) with negative size and proximity weight to produce trees with redundant parts.

A particular run of the generator can be parameterized by the tuple $(N_{iter}, P_{CSI}, P_{DNI}, P_{DLI}, P_{ALI}, P_{GRI})$, where N_{iter} is the number of iterations of the generator and $P_{...}$ are the probabilities of the insertion strategies. These artificially introduced redundancies cover all sorts of redundancies potentially appearing in CSG trees. Table

6.1 lists properties of the input models and their generated inflated versions. Data set 1 uses all possible inflation mechanisms with the same probability for 10 iterations $(10, 1.0, 1.0, 1.0, 1.0, 1.0)$, data set 2 uses GA inflation only for 20 iterations $(20, 0.0, 0.0, 0.0, 0.0, 1.0)$. All sampling techniques use a step size of 0.1.

6.2 Execution Times

For evaluating the execution time, we used a quad core CPU @2.80 GHz and 16GB of RAM. Set covers are computed without QA hardware acceleration.

6.2.1 Pipeline

For the pipeline steps *Redundancy Removal* and *Decomposition*, different options are possible: a) decomposition or b) the redundancy removal method can use the hierarchical sampling strategy (0) or CIT-based sampling (1) and c) redundancy removal can be used (1) or not (0). Options a) - c) can be combined resulting in 6 possible configurations. A particular configuration is identified by a binary 3-tuple, e.g., $(1, 0, 1)$ for decomposition with CIT-based sampling and redundancy removal with hierarchical sampling. All cases use Espresso for the remaining solid optimizer. Results are given in Fig. 5 and 6. For both data sets, CIT-based sampling is inferior in any configuration due to its processing time. The initial redundancy removal does not have any positive impact on execution times as well (exception: model 3 and model 10 in both data sets).

6.2.2 Remaining Solid Optimization

Results for data set 1 are given in Table 6.2.2. For data set 2 results are similar and thus are not shown due to space constraints. Overall the fastest method is Espresso, followed by the Quine-McCluskey method (exception: model 10 where Set Cover is faster). Within the Set Cover method, the prime implicant generation part is the most time expensive whereas solving the Set Cover problem is neglectable ($\sim$ 1-3ms). The slowest method is the GA-based optimization with the exception of model 8, where Set Cover is the slowest.

6.3 Optimization Characteristics

All experiments use the pipeline configuration $(0, 0, 1)$.

6.3.1 Pipeline

For both data sets, resulting size (Fig. 7) and proximity (Fig. 8) for models 1, 3, 4, 5, 6, 7 and 9 are the same for all methods since for these models, the remaining solids are empty after decomposition and thus no remaining solid optimization is necessary.
For both data sets, the GA produces the best size results (exception: model 11) but not always the best proximity results (e.g., data set 2, model 2). This can be explained by two factors: The GA's objective function's

size weight used in our experiments is greater than its proximity weight (10 vs. 1) and larger trees have more redundancies and thus tend to have higher subtree overlap resulting in higher proximity scores. This explains as well the results of the Set Cover method which are the worst in terms of tree size but among the best regarding proximity. Compared to the original, hand-crafted input trees, resulting trees have equally good or better size and proximity values for both data sets (exception: size for model 8, data set 2). Also worth mentioning is that the Quine-McCluskey method cannot handle the size of model 8.

6.3.2 GA Specifics

For the GA, we used a population size of 150, a mutation rate of 0.3, a crossover rate of 0.4 and a tournament selection $k = 2$. Objective function weights are $\alpha = 50, \beta = 1, \gamma = 10$. The maximum iteration count is 1000, and after 500 iteration without score change, the GA is terminated as well. Fig. 9 shows one of the main advantages of the GA approach for remaining solid optimization: It is possible to manually select the best size/proximity trade-off solution for a particular use-case after the optimization process has finished.

7 CONCLUSION

In this paper, we proposed a flexible new pipeline for the efficient optimization of a CSG tree's editability. We have evaluated different pipeline combinations with a representative set of models.
It is possible to use the sampling-based optimization method (Section 5.3.1) in situations where the input expression is not known in advance (only the solid point- and halfspace-set). In that case, another strategy for *inside-outside* decisions would be needed.
Furthermore, the decomposition technique (Section 5.2) can be also used for the task of CSG tree reconstruction from other geometry representations (e.g., point clouds or B-Rep representations). Therefore, the dominant halfspace detection must be adapted according to the geometry representation of the input and the discussed recursion would end after the first iteration without applied redundancy removal (which is not possible without a given input tree). A GA-based approach [FP16] (or the aforementioned variant of the sampling-based optimization method) could be used to find a tree expression for the remaining solid.

8 REFERENCES

[AL18]   Tameem Albash and Daniel A Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1):015002, 2018.

[And13]   James L Andrews. *User-Guided Inverse 3D Modeling*. PhD thesis, University of California, Berkeley, 2013.

Models (number of nodes, proximity score, (dimensions))	Data set 1	Data set 2
model1 (9, 0.75, (20.0×22.0×20.0))	(97, 0.442)	(78, 0.35)
model2 (13, 0.833, (15.8×31.7×11.3))	(41, 0.952)	(23, 0.727)
model3 (39, 0.474, (21.0×6.0×21.0))	(79, 0.675)	(57, 0.536)
model4 (27, 1, (13.4×13.4×12.0))	(65, 0.97)	(97, 0.708)
model5 (19, 0.667, (24.0×18.0×27.0))	(59, 0.645)	(48, 0.76)
model6 (19, 0.556, (23.1×10.0×10.0))	(99, 0.667)	(43, 0.857)
model7 (91, 0.706, (21.6×7.4×21.8))	(144, 0.725)	(151, 0.728)
model8 (73, 0.722, (31.5×12.7×4.5))	(145, 0.74)	(97, 0.813)
model9 (171, 0.471, (29.7×3.84×30.1))	(191, 0.51)	(216, 0.519)
model10 (17, 0.875, (13.0×12.5×13.0))	(41, 0.864)	(32, 0.563)
model11 (37, 0.789, (26.0×13.0×22.0))	(86, 0.894)	(66, 0.8)

Table 1: Models and their inflated variants. For each data set, (number of nodes, proximity score) is depicted.

(a) Model 1   (b) Model 2   (c) Model 3   (d) Model 4   (e) Model 5   (f) Model 6   (g) Model 7

(h) Model 8   (i) Model 9   (j) Model 10   (k) Model 11
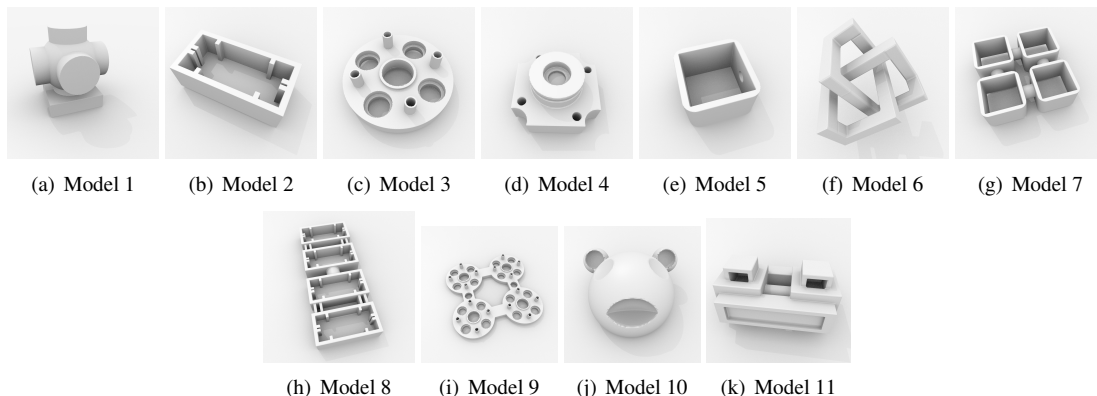
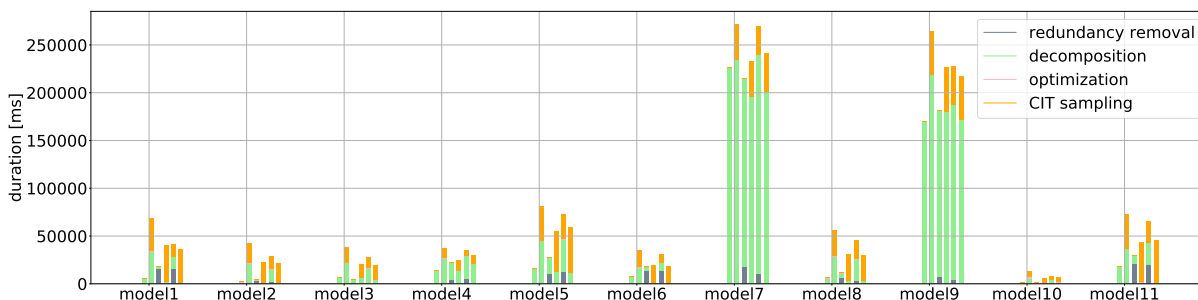Figure 4: Models used for the evaluation.

Figure 5: Timings for different pipeline configurations using data set 1. For each model configurations from left to right: $(0,0,0)$, $(1,0,0)$, $(0,0,1)$, $(0,1,1)$, $(1,0,1)$ and $(1,1,1)$.
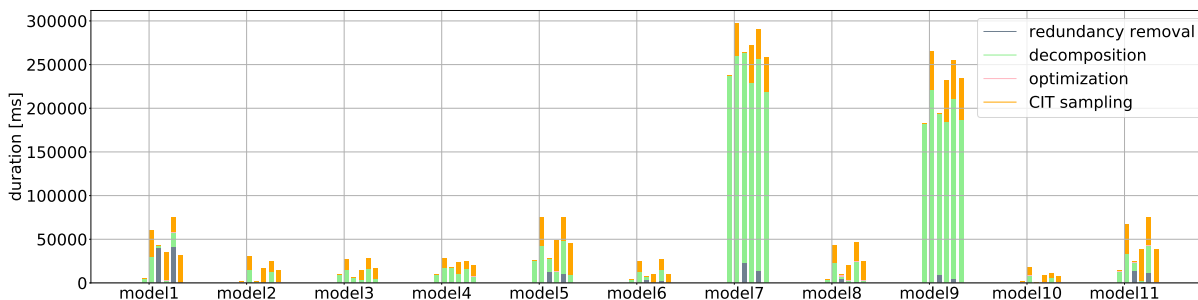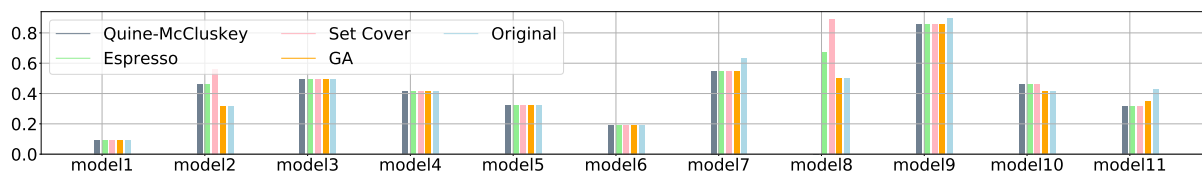

Figure 6: Timings for different pipeline configurations using data set 2. For each model configurations from left to right: $(0,0,0)$, $(1,0,0)$, $(0,0,1)$, $(0,1,1)$, $(1,0,1)$ and $(1,1,1)$.
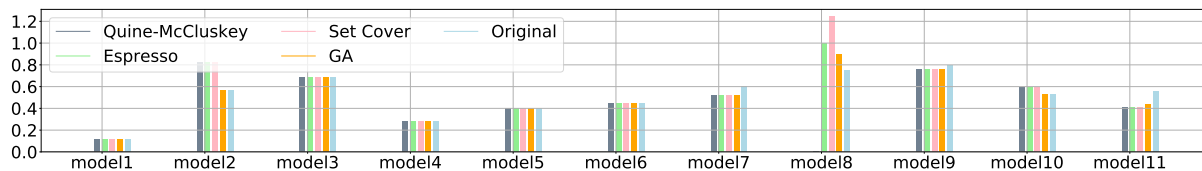
[BC04]   Suzanne F Buchele and Richard H Crawford. Three-dimensional halfspace constructive solid geometry tree con-struction from implicit boundary rep-resentations. *Computer-Aided Design*, 36(11):1063–1073, 2004.
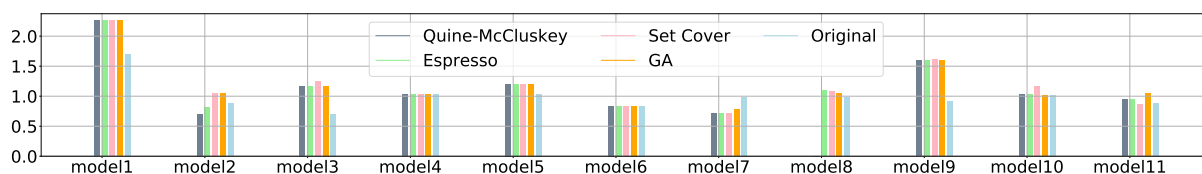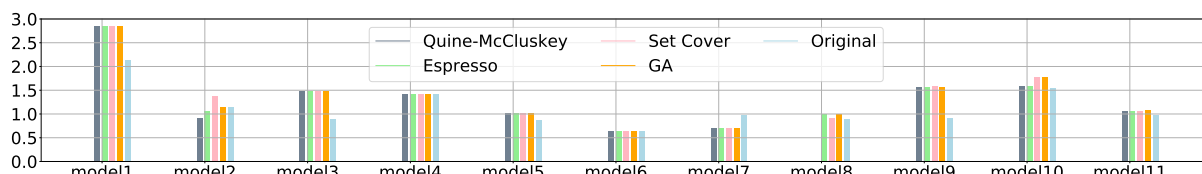
(a) Data set 1

(b) Data set 2

Figure 7: Ratio of input and output tree size for both data sets. The light blue bar 'Original' indicates the size of the initial hand-crafted expression.

(a) Data set 1

(b) Data set 2

Figure 8: Ratio of input and output tree proximity for both data sets. The light blue bar 'Original' indicates the proximity of the initial hand-crafted expression.

Models	Q-McC	Espresso	Set Cover	GA
model2	1572	1066	17305	21526
model8	-	1163	111017	79829
model10	3928	940	3347	22094
model11	3879	900	8594	90971

Table 2: RSO timings for data set 1 in [ms].

[BHMSV84] Robert K Brayton, Gary D Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media, 1984.

[DIP⁺18] 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. In *SIGGRAPH Asia 2018 Technical Papers*, SIGGRAPH Asia '18, pages 213:1–213:16, New York, NY, USA, 2018. ACM.

[FFGLP19] Markus Friedrich, Pierre-Alain Fayolle, Thomas Gabor, and Claudia Linnhoff-Popien. Optimizing evolutionary csg tree extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO 2019, pages 1183–1191. Association for Computing Machinery, 2019.

[FP16] Pierre-Alain Fayolle and Alexander Pasko. An evolutionary approach to the extraction of object construction trees from 3d point clouds. *Computer-Aided Design*, 74:1–17, 2016.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[Luc14] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014.

(a) Model 2 (input and output trees are equal)

(b) Model 8
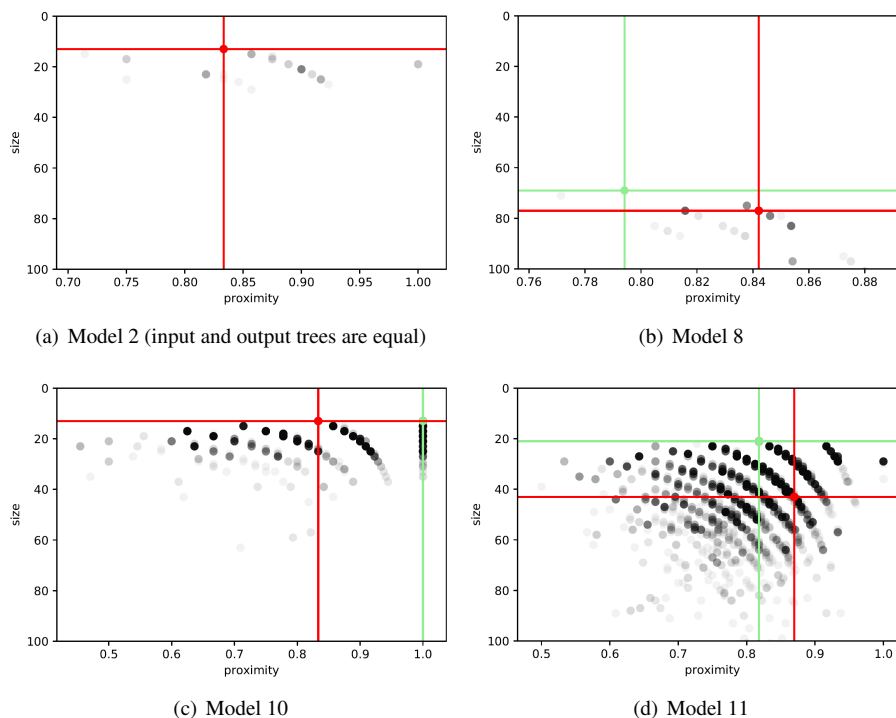
(c) Model 10

(d) Model 11

Figure 9: All trees representing the remaining solid produced by the GA for data set 2 with a geometric score of 1.0. The darker the point, the more trees have exactly this size/proximity combination. The red dot indicates the input tree that represents the remaining solid, the green one the selected resulting tree.

[MJ56] Edward J McCluskey Jr. Minimization of boolean functions. *Bell system technical Journal*, 35(6):1417–1444, 1956.

[O'R82] Joseph O'Rourke. Polygon decomposition and switching function minimization. *Computer Graphics and Image Processing*, 18(4):382 – 391, 1982.

[PASS95] Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

[Qui52] W. V. Quine. The problem of simplifying truth functions. *American Math. Monthly*, 59(8):521–531, 1952.

[Req80] Aristides G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464, December 1980.

[Ric73] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973.

[Ros11] Jarek Rossignac. Ordered boolean list (OBL): Reducing the footprint for evaluating boolean expressions. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1337–1351, 2011.

[Sha99] Vadim Shapiro. Well-formed set representations of solids. *International Journal of Computational Geometry & Applications*, 9(02):125–150, 1999.

[Sha07] Vadim Shapiro. Semi-analytic geometry with r-functions. *ACTA numerica*, 16:239–303, 2007.

[SV91a] Vadim Shapiro and Donald L. Vossler. Construction and optimization of CSG representations. *Computer-Aided Design*, 23(1):4–20, 1991.

[SV91b] Vadim Shapiro and Donald L Vossler. Efficient CSG representations of two-dimensional solids. *Journal of Mechanical Design*, 113(3):292–305, 1991.

[SV93] Vadim Shapiro and Donald L Vossler. Separation for boundary to CSG conversion. *ACM Transactions on Graphics (TOG)*, 12(1):35–55, 1993.

[Til84] Robert B. Tilove. A null-object detection algorithm for constructive solid geometry. *Commun. ACM*, 27(7):684–694, July 1984.

[WXW18] Qiaoyun Wu, Kai Xu, and Jun Wang. Constructing 3d CSG models from 3d raw point clouds. *Computer Graphics Forum*, 37(5):221–232, 2018.