Accelerating Evolutionary Construction Tree Extraction via Graph Partitioning

First Author
author's affiliation
1st line of address
2nd line of address
Country (ZIP) code, City, State
e@mail

Second Author
author's affiliation
1st line of address
2nd line of address
Country (ZIP) code, City, State
e@mail

Third Author
author's affiliation
1st line of address
2nd line of address
Country (ZIP) code, City, State
e@mail

ABSTRACT

Extracting a Construction Tree from potentially noisy point clouds is an important aspect of Reverse Engineering tasks in Computer Aided Design. Solutions based on algorithmic geometry impose constraints on usable model representations (e.g. quadric surfaces only) and noise robustness. Re-formulating the problem as a combinatorial optimization problem and solving it with an Evolutionary Algorithm can mitigate some of these constraints at the cost of increased computational complexity. This paper proposes a graph-based search space partitioning scheme that is able to accelerate Evolutionary Construction Tree extraction while exploiting parallelization capabilities of modern CPUs. The evaluation indicates a speed-up up to a factor of 46.6 compared to the baseline approach while resulting tree sizes increased by 25.2% to 88.6%.

Keywords

3-d Reconstruction, Reverse Engineering, Computer Aided Design, Constructive Solid Geometry, Evolutionary Algorithms, Graph Theory

1 INTRODUCTION

Reverse Engineering (RE) – i.e., the recovery of a model's geometric representation from potentially noisy and incomplete sensor data – is an important aspect of modern Computer Aided Design (CAD) pipelines. It allows for convenient model editing based on real-world physical objects, thus simplifying and accelerating the product design process. An expressive and intuitive model representation scheme extensively used in solid modeling is Constructive Solid Geometry (CSG). It describes complex rigid solids by a binary tree with regularized Boolean set-operations (e.g., union, intersection, subtraction) as inner nodes and primitive solids (e.g., cubes, spheres, cylinders and cones) as leaves. Such a tree is also known as a model's Construction Tree. Due to the popularity of CSG in CAD, it is desirable to have tools at hand that are able to reliably recover a model's CSG-tree from its point cloud representation stemming from sensor recordings. CSG-tree generation might be solved by converting the input point cloud to a Boundary Representation (B-rep) and then by conversion of the B-rep to CSG with methods based on algorithmic geometry that usually require exact geometric intersection computations [SV93, BC04]. These approaches are usually restricted to a single model representation for primitives, e.g. a surface description that uses quadrics, and can be sensitive to inexact representations.

To overcome these constraints, CSG-tree generation can be formulated as a combinatorial optimization problem over the possible permutations of primitives and set-operations for a fixed maximum CSG-tree depth. Metaheuristics, like Genetic Algorithms (GAs) can then be employed for optimization [Mit98].
One of the most severe disadvantages of GA-based solutions are computation times of minutes and hours for comparably small models (less than 10 primitives) [FP16]. This issue is addressed in this paper.
The basic idea of the described acceleration scheme is to exploit spatial relationships between primitives: Primitives that do not overlap spatially are not considered to be operands of a CSG-operation. This knowledge can be used to partition overlapping primitives and to compute partial per-partition results that are later on merged into a single CSG-tree. In particular, this paper makes the following contributions:

- An acceleration scheme based on spatial search space partitioning together with a robust merge mechanism.
- A description and analysis of parallelization strategies for the proposed algorithms.

The paper has the following structure: Section 2 discusses related work in the field of CSG-tree extraction and surface reconstruction. It is followed by an introduction to the theoretical principles of the proposed

method (Section 3). The problem to solve is detailed in Section 4. The proposed solution is described in Section 5 and evaluated in Section 6. Section 7 summarizes the results and sketches possible future work.

2 RELATED WORK

This work is related to different domains such as surface reconstruction from discrete point clouds, Reverse Engineering of solid models and conversion from B-rep to CSG. In this section, important related work in these domains is briefly discussed.

2.1 Surface Reconstruction

The problem of reconstructing a surface from a discrete point cloud has been the subject of much attention in computer graphics. The most popular methods include fitting implicit surfaces such as [OBA+03], or Poisson surface reconstruction [KH13], among others. The recent work of Berger et al. [BTS+17] presents a wide survey of the topic. Using these methods, the reconstructed objects lack information that can be used for inspection or re-use of the object in further modeling.

2.2 Reverse Engineering and B-rep to CSG Conversion

The goal of Reverse Engineering is the creation of consistent geometric models from point cloud data [VMC97, BMV01]. They usually output B-rep models made of parametric patches.

The conversion from B-rep to CSG was first investigated for two-dimensional, linear polygons, then later extended by Shapiro et al. for handling curved polygons [SV91b, Sha01]. The extension to three-dimensional objects was initially solved by Shapiro and Vossler in [SV91a, SV93] and later improved by Buchele and Crawford in [BC04] and by Andrews in [And13]. These works rely on the fact that surfaces are composed of quadric surface patches. Another issue is the handling of inexact representations. These methods work under the assumption that the patches form a clean partition of the target solid. However, in practice, we are dealing with input point clouds that are potentially noisy, contain holes, or have additional details and thus the fitted primitives may not fit perfectly. This could impact the cellular classification on which these methods rely.

2.3 Point Cloud to CSG Construction

Close to the proposed approach are methods that handle noisy and incomplete point clouds such as [SWK07] for fitting primitives and methods that try to convert them to a higher level representation such as [FP16], see also [BTS+17, Sections 7 and 8]. One of the goals of this work is to improve the running time of the Evolutionary Algorithm used in [FP16] via geometric consideration, i.e. the overlapping in space of primitives.

3 BACKGROUND

3.1 Point Cloud to CSG-Tree Pipeline

The extraction of a CSG-tree from a point cloud poses a complex problem which is usually solved with a processing pipeline that comprises the following steps:

1. **Point cloud generation and pre-processing:** Point clouds are generated by laser scanners or tactile measurement devices. Other techniques use photogrammetric algorithms to gather depth information from (un-)calibrated camera images [HZ03]. Measured point clouds usually contain significant amounts of noise and outliers. These can be trimmed from the data-set using e.g. statistical approaches [RC11].

2. **Point cloud segmentation and primitive fitting:** The point cloud must be segmented and primitive parameters have to be fitted to the corresponding points. Approaches that fulfill both tasks for simple geometric shapes are e.g. specialized variants of the Random Sample Consensus (RANSAC) technique [SWK07].

3. **CSG-tree generation:** CSG-tree generation can be done with methods based on algorithmic geometry such as [SV93, BC04, And13], or by formulating the problem as an optimization problem. For the latter, solutions can then be obtained eg. by evolutionary [FP16] or greedy [XF14] approaches for handling inexact representations.

4. **CSG-tree optimization:** The resulting CSG-tree might not be optimal in terms of size and depth. Additional optimization techniques can simplify the tree structure [SV91a].

3.2 Primitive Description

Primitives are basic shapes located at CSG-tree leaves. A primitive p is fully described by its signed distance function $f_p : \mathbb{R}^3 \mapsto \mathbb{R}$. The surface of p is implicitly defined by the zero-set of f_p: $\{x \in \mathbb{R}^3 : f_p(x) = 0\}$. Its surface normal at point $x \in \mathbb{R}^3$ is given by the gradient $\nabla f_p(x)$. If the gradient does not exist at x or is too expensive to compute, finite difference approximations can be used [Olv14].

3.3 Boolean Set-Operations

The set-operations intersection, union, complement and subtraction are implemented using min- and max-functions [Ric73]:

- Intersection: $S_1 \cap S_2 := \min(f_{S_1}, f_{S_2})$
- Union: $S_1 \cup S_2 := \max(f_{S_1}, f_{S_2})$
- Complement: $\overline{S} := -f_S$
- Subtraction: $S_1 \setminus S_2 := S_1 \cap \overline{S_2}$

where S_i is the solid corresponding to the set $\{x \in \mathbb{R}^3 : f_{S_i} \geq 0\}$ $(i = 1,2)$. In the following, the considered Boolean set-operations are intersection, union, complement and subtraction.

3.4 Evolutionary Algorithms

Evolutionary Algorithms are biology-inspired, stochastic metaheuristics for solving optimization problems [ES+03].

The optimization process starts with a randomly initialized population of individual candidates sampled from the problem's search space (initialization). In each iteration, candidates are ranked according to their fitness by evaluating the so-called fitness function. The best candidates are selected to be the next generation's parents (parent selection). Parents are then recombined (crossover) and mutated (mutation) to create offspring. The new population is then filled with the offspring together with selected surviving individuals (survivor selection) from the current population. This procedure is repeated until a certain termination criteria is met (termination). See Fig. 1 for an overview.

Evolutionary Algorithms are especially useful for solving combinatorial optimization problems [ES+03].

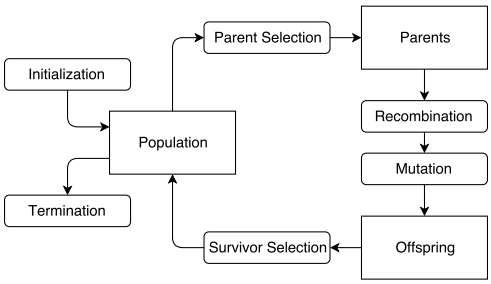


Figure 1: The optimization process described by an Evolutionary Algorithm (derived from [ES+03]).

4 PROBLEM STATEMENT

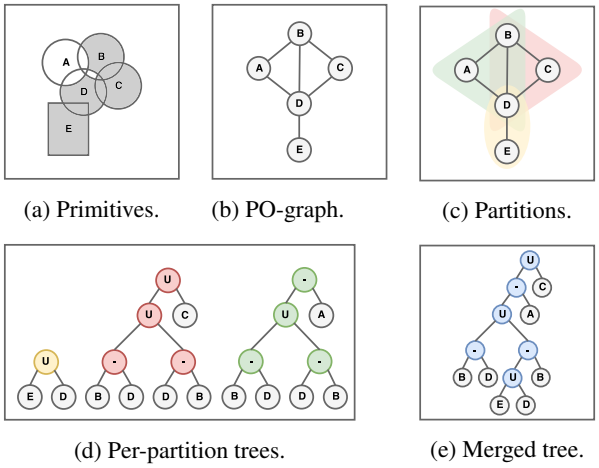
The problem of accelerating GA-based CSG-tree extraction from point clouds is considered as the open research question addressed by this paper. Focus is on the CSG-tree generation and optimization step of the reconstruction pipeline described in Section 2.3. As input, a point-set of potentially noisy 3-d measurements of a connected geometric model together with already segmented and fitted primitives is considered. Thus, the first two pipeline steps are assumed to be solved.

The desired output is a CSG-tree that represents the scanned real-world model as accurately as possible. A measure for accuracy is given by the distance between the CSG-tree induced surface and the points of the input point cloud. CSG-tree extraction approaches based on a GA [FP16] can handle inaccuracies but come with the disadvantage of potentially high computation times.

5 CONCEPT

The basic idea for accelerating CSG-tree extraction is to partition the search space into independent groups of spatially overlapping primitives. This exploits the fact that primitives that do not overlap are not considered to be operands of a CSG-operation. CSG-tree extraction is then conducted on a per-partition level. Finally, resulting trees are combined in a subsequent merge step without loss of result quality and correctness.

An overview of the full CSG-tree extraction pipeline is depicted in Fig. 2. Each of the steps is described in detail in the following sub-sections, following the order of execution.



(a) Primitives. (b) PO-graph. (c) Partitions.

(d) Per-partition trees. (e) Merged tree.

Figure 2: The search space partitioning pipeline.

5.1 Primitive Overlap Graph Generation

For expressing spatial relationships between primitives, the Primitive Overlap (PO)-graph is introduced. It represents spatial overlap between primitives using an undirected graph $G = (P, O)$, where $P = \{p_1, \ldots, p_{n_p}\}$ is the set of n_p primitives as vertices and O is the edge-set that contains 2-tuples of overlapping primitives $o = (p_i, p_j)$, where $i, j \in \{1, \ldots, n_p\}$ with $i \neq j$.

The PO-graph is generated based on the location, orientation and geometric shape of the primitives, see Fig. 2b. Complex shapes can be approximated with

simpler bounding volumes like Oriented Bounding Boxes (OBBs) or the convex hull of the corresponding point-set [PH77].

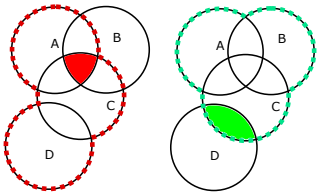
For better scalability, the computational complexity can be reduced from $\mathcal{O}({n_p}^2)$ (overlap check between each primitive and each other primitive) to $\mathcal{O}(n_p \log(n_p))$ using hierarchical space partitioning schemes like e.g. Octrees [Mea82].

5.2 Search Space Partitioning

With known primitives and their spatial relations given by the PO-graph, the goal is now to find independent search space partitions.

A partition is a set of primitives in which each primitive has an overlap with each other primitive. In this context, independence means that per-partition solutions are not influenced by the solutions of other partitions. See Fig. 3 for explanatory examples.

The problem of finding all independent search space partitions is equivalent to the problem of finding all maximum complete subgraphs (maximum cliques) in G. For finding the set of maximum cliques in G, the Bron-Kerbosch Algorithm (BKA)[BK73] is employed due to its behavior on random graphs: It was experimentally shown in [BK73] that the computational complexity of BKA is almost independent of graph size for random graphs. In a worst case scenario (using Moon-Moser Graphs [MM65]), computational complexity is proportional to $(3.14)^{\frac{n}{3}}$, where n is the size of the graph. Note that, if there is only a single partition for a particular PO-graph, the search space partitioning method degenerates to standard GA-based CSG-tree extraction.



(a) Incorrect. (b) Correct.

Figure 3: In (a), per-patition solution parts containing A and C are partially influenced by B (red area) but B is not part of the partition. In (b), D is not part of the partition and influences C only in an area (green) that does not overlap with other partition members. Thus, per-partition solutions are not influenced by D.

5.3 Per-Partition CSG-Tree Extraction

With known partitions, CSG-tree extraction is conducted for each partition separately in a divide-and-conquer manner. A variant of the GA described in [FP16] is used with the objective function

$$E(t) := \sum_{i=1}^{|S|} \left\{ e^{-d_i(t)^2} + e^{-\theta_i(t)^2} \right\} - \alpha \cdot \text{size}(t), \qquad (1)$$

where t is the tree candidate, S is the point-set corresponding to the partition's primitives and size(t) is the number of nodes in tree t weighted by a factor α. $d_i(t) = \beta \cdot f_t(s_i)$ is the signed distance between point s_i and the surface defined by tree t weighted by a factor β. $\theta_i(t) = \gamma \cdot \arccos(\nabla \hat{f}_t(s_i) \cdot n_i)$ is the angle between the point normal n_i and the normalized gradient at position s_i weighted by a factor γ. α, β and γ are user-controlled parameters. The first term in Equation 1 (under the sum) estimates how close the surface induced by t matches the point cloud, while the second term penalizes trees with a large number of nodes. The given objective function has to be maximized for t.

Initially, the population T_0 is filled with n_T randomly generated trees with a height $\leq h_{max}$. For the maximum tree height, the approximation

$$h_{max} \approx \sqrt{\pi/2 \cdot n_{pp} \cdot (n_{pp} - 1)} \qquad (2)$$

is used, where n_{pp} is the number of primitives in the partition. It is based on the average height of binary trees for a given number of internal nodes [FO82] and achieved good results in all experiments carried out.

Each GA iteration i contains the following steps:

1. The population of the previous iteration T_{i-1} is ranked according to Equation 1.
2. The current population is initialized with the n_b best candidates from T_{i-1}.
3. As long as T_i has not reached maximum population size n_T, two crossover candidates are selected from T_{i-1} via Tournament Selection parameterized with k_{ts} (size of the set of randomly chosen population members from which the best member is selected) [MG95]. During crossover, the two candidates exchange randomly selected subtrees with a probability of γ_{cr}. Then, with a probability of γ_{mu}, the resulting two trees are mutated. In the mutation process a randomly chosen subtree is replaced with a new randomly generated subtree with a probability of μ_{mu}. With a probability of $1 - \mu_{mu}$, the whole tree is replaced with a randomly generated tree.
4. The termination condition is met if the score of the best CSG-tree candidate of an iteration does not improve over n_{tc} iterations.

The most computationally expensive step in GA-based CSG-tree recovery is the evaluation of Equation 1 for each element of a candidate-set. Since evaluations can be conducted for each candidate independently, parallel processing schemes can be applied efficiently. In addition, the solution space partitioning allows for a per-partition parallelization strategy. Both options were implemented for multi-core processors. Their evaluation is part of Section 6.

5.4 Merge of Per-Partition Trees

Merging all trees corresponding to partitions into a single tree is not trivial. A simple union of all tree root nodes may lead to incorrect results if primitives that are part of multiple cliques are not splitted. Split operations on arbitrary primitive shapes tend to be complex and thus should be avoided. See Fig. 4 for examples. The proposed merge strategy does not need splits but instead tries to merge trees that have a subtree in common. It consists of the following steps:

1. All trees are inserted in a list L without adhering to any specific order. Extracted trees might contain artefacts that affect their mergeability (e.g., intersections with the same primitive for both operands). For each tree in L, artefacts are removed by traversing the tree and replacing found patterns iteratively with their simplifications (e.g., replacing an intersection with the same primitive for both operands with the primitive itself). The process is repeated until no more artefacts can be removed.

2. Two trees t_0 and t_1 are removed from the head of L, and their largest common subtree t_{lcs} is computed which has an asymptotic computational complexity of $\mathcal{O}(\max(\text{size}(t_0), \text{size}(t_1)))$. The subtree's leaf-set must be a subset of the leaf-sets of both, t_0 and t_1. The largest common subtree found might exist more than once in both trees. Thus, the root nodes of each appearance of the subtree in t_0 and t_1 are stored in the lists N_0 and N_1 (see Fig. 5a).
   If t_{lcs} is empty, t_1 is appended to L and a new tree candidate t_1 is removed from the head of L. In this case, the largest common subtree search is repeated with the new t_1.

3. For each node in N_0 and N_1 it is checked if it is a valid merge candidate. This is done by traversing the corresponding tree (t_0 or t_1) from root node to leaves following Algorithm 1. If the node can be reached this way, it is considered a valid merge candidate. The node is then replaced by the root of the other tree resulting in a merged tree t_m. If more than one valid candidate exists, the candidate corresponding to the larger tree is replaced by the root of the smaller tree. If both trees are of the same size, the candidate of t_0 is chosen (see Fig. 5b).
   If there is no valid merge candidate, the procedure is repeated with the next smaller common subtree in t_0 and t_1. If no other common subtree exists, t_1 is replaced by a new tree candidate from the head of L. Then, the largest common subtree search and its subsequent steps are repeated with the new t_1.

4. The merged tree t_m is prepended to L.

5. The merge process is continued until there is only a single node left in L. Since the model to reconstruct is connected, a pair of mergeable trees exists in each iteration. Thus, the merge process always terminates.

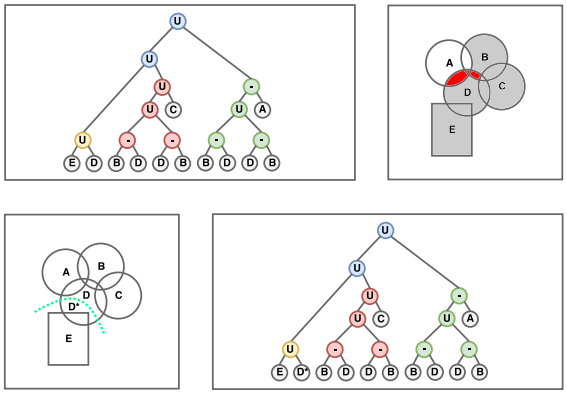
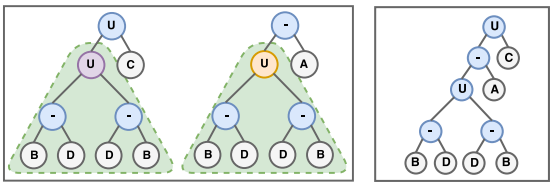


Figure 4: Merge strategies. Top: Wrong tree merge using union over all partition trees. Erroneous geometry in red (compare with Fig. 2a). Bottom: Correct tree merge using union over all partition trees with primitive splitting (green curve).



(a) (b)

Figure 5: (a) Two merge trees (t_0 left, t_1 right) with a largest common subtree (green). N_0 contains the purple node, N_1 the orange node. (b) The merged tree t_m.

The merge process has an asymptotic computational complexity of $\mathcal{O}(|L|^2)$ since in the worst case L has to be traversed for each merge.

6 EVALUATION

The proposed partitioning scheme has been evaluated on a laptop with quad core CPU and 16GB of RAM with four different models. For model M0, M1 and M2, point clouds were generated by sampling a model surface induced by a pre-defined CSG-tree that served as ground-truth. Gaussian noise ($\mu = 0.0, \sigma = 0.01$) was added to sampling points to simulate measurement errors. Model M3 is based on real measurements, and primitive fitting was conducted using RANSAC [SWK07]. Fig. 10 depicts all pipeline step results for model M1. Fig. 11 shows point clouds and renderings for model M0, M2 and M3. Table 1 contains model details.
Baseline is the GA approach proposed in [FP16] and described in Section 5.3. The parameter-set used for

```
Procedure isValid(curNode, node)
    if curNode = node then
        return true
    if curNode.nodeType = Operation then
        if curNode.operationType = Difference
         then
            return
             isValid(curNode.childs[0],
             node)
        else if curNode.operationType = Union
         then
            foreach child ∈ curNode.childs do
                if isValid(child, node) then
                    return true
    return false
1 isValid(t.root, node)
```

Algorithm 1: Checks if node *node* is a valid merge candidate in tree t.

	M0	**M1**
# Primitives	17	4
# Points (low)	11.3k	9.3k
# Points (high)	156.4k	158.4k
# Partitions	(0,8,4,0,1,1)	(0,0,2)
	M2	**M3**
# Primitives	29	18
# Points (low)	10.9k	-
# Points (high)	155.4k	55.8k
# Partitions	(0,0,0,12)	(0,7,4,1)

Table 1: Details on evaluated models. 'low' and 'high' indicate different sampling rates. Numbers of partitions are depicted per partition size. First position in parantheses indicate number of partitions of size 1 and so on.

both, baseline and partitioning scheme, is listed in Table 2. The following combinations were evaluated:

- **Baseline:** Single-threaded (BST), multi-threaded GA (BMTGA).
- **Search Space Partitioning:** Single-threaded (SST), per-partition multi-threaded (SMTP) multi-threaded GA (SMTGA), per-partition and GA multi-threaded (SMTPGA) combined.

6.1 Computation Times

Timings for baseline and search space partitioning variants were measured for all models with high- and low-detail sampling (except for model M3 for which only a single point cloud exists). Measurements vary significantly for the same benchmark setting due to the inherently stochastic behavior of GA-based methods. In order to deal with the high variance, each experiment was repeated 5 times.

Parameter Name	**Value**
Population size n_T	150
# Best parents n_b	2
Crossover probability γ_{cr}	0.3
Mutation probability γ_{mu}	0.3
Subtree replacement probability μ_{mu}	0.5
Tournament selection parameter k_{ts}	2
Tree size weight α	log(#points)
Distance weight β	100.0
Angle weight γ	$18.0/\pi$
# Iterations w/o quality increase n_{tc}	10
Maximum tree height h_{max}	see Equation 2

Table 2: Parameters for the baseline and search space partitioning approach.

In the following, timing results for all methods in combination with high-detail sampling are discussed. See Fig. 6 and 7 for an overview of the results. For model M0, SMTGA is the fastest method. It outperforms the baseline by a factor of 15.3 (single-threaded, BST) and 7.5 (multi-threaded, BMTGA) on average. For model M1, search space partitioning performs worse than baseline: The fastest baseline method (BMTGA) is on average 1.4 times faster than the best-performing search space partitioning variant (SMTGA). This can be explained by the relatively small number of primitives (4) and partitions (2) in model M1 which eliminates the need for partitioning. For model M2, single-threaded partitioning is 38.3 times faster than single-threaded baseline and multi-threaded partitioning variants are between 43.4 and 46.6 times faster than multi-threaded baseline. The considerable difference is due to the relatively high number of partitions (12) and their equally distributed size (all contain 4 primitives). For model M3, SMTGA is again the fastest method. Compared to multi-threaded baseline it is 3.0 times faster on average.

Search space partitioning with GA parallelization (SMTGA) is in general faster than their per-partition counterparts (SMTP, SMTPGA) for all models. This is due to the granularity and regularity of the parallelization: For SMTGA, the task of ranking a population can be splitted into n_T parts, with each part having similar execution times. For per-partition variants, granularity is determined by the (potentially lower) number of partitions, and per-partition execution times may vary significantly depending on partition sizes.

Results for per-partition variants do not show timings for different pipeline steps since in all experiments, per-partition CSG-tree extraction is by far the most dominant factor. The summarized time measures for PO-graph generation, search space partitioning and tree merge make less than 1‰ of the total runtime.

6.2 Tree Sizes and Depths

Fig. 9 contains average depths and sizes of resulting trees for baseline and partitioning variants. For the latter, tree depths have increased by 25.0% (model M1) to 285.0% (model M2) compared to the input tree, while for baseline approaches, an increase of 0.0% (model M1) to 125.0% (model M2) is visible. Tree sizes show similar behavior: Partitioning variants produce 46.1% (model M2) to 68.2% (model M0) larger trees, while baseline approaches increase tree size by only 0.0% (model M0) to 16.7% (model M2). Comparing tree sizes between partitioning and baseline approaches directly reveals that the former results in 25.2% (model M2) to 88.6% (model M3) larger trees.

This adverse behavior shown by partitioning variants is due to the final merge step: In each iteration, the two trees that are close to each other in the tree list and have a common subtree of at least size 1 are merged instead of the two trees with the largest common subtree of all tree pairs in the merge list. Since the focus of this work is on performance, this is acceptable. In addition, the tree optimization strategy described in Section 5.4 (step 1) was also applied to baseline results for better comparability which has positive impact on resulting tree depths and sizes.

6.3 Scalability with Respect to Point Cloud Size

Fig. 8 depicts measurement results for the ratio

$$\frac{\#\text{points}_{high}}{\#\text{points}_{low}} : \frac{\text{duration}_{high}}{\text{duration}_{low}} \tag{3}$$

which quantifies the dependency between point cloud size and corresponding computation times. It indicates that, for larger models (model M0 and M2), the fastest partitioning approach scales up to 1.9 times better than the best performing baseline approach with respect to point cloud size.

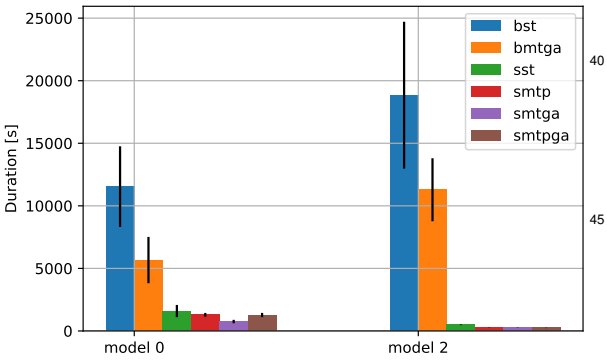


Figure 6: Timings for all approach combinations and models M0 and M2 with high-detail sampling (black lines: standard deviations).

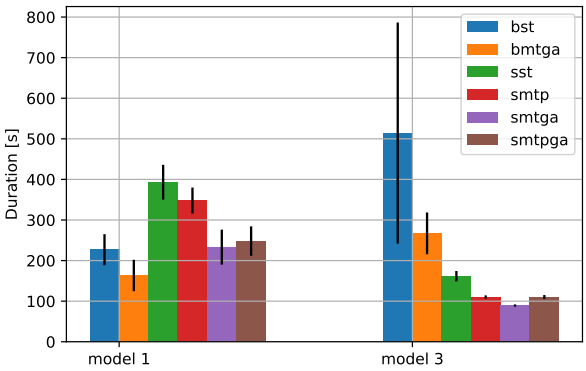


Figure 7: Timings for all approach combinations and models M1 and M3 with high-detail sampling (black lines: standard deviations).

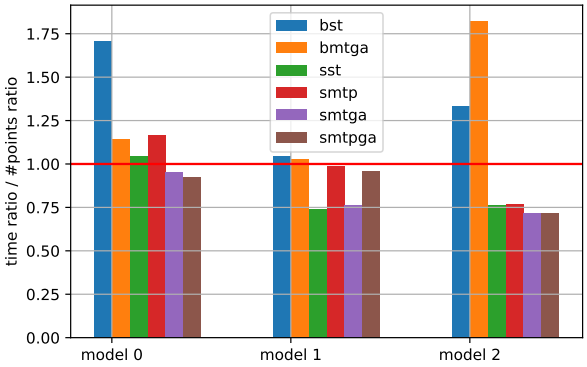


Figure 8: Ratio between high-detail and low-detail point cloud size factor and corresponding timing factors for all models (see Equation 3). The red line indicates linear scaling with a slope of 1 with respect to point cloud size. Model M3 exists only in high-detail.

7 CONCLUSION

In this work, a technique for accelerating an Evolutionary Algorithm for extracting a CSG-tree from a point cloud was proposed. It is based on a partitioning of the search space obtained from computing the maximum cliques of a graph of overlapping primitives, and on merging CSG-trees extracted for each partition. The experimental evaluation indicated a significant speed-up over the baseline approach (the Evolutionary Algorithm) for different modes of parallelization.

One possible direction for future work is the implementation of the GA for massively parallel computing hardware, combined with the proposed partitioning approach. A decreased tree size in the partitioning approach could also be achieved by improving the merge process. Finally, since the partitioning (and merge) approach described in this work is independent of the technique used for the CSG-tree construction, the same approach could potentially be used with the CSG-tree conversion approaches in [SV91a, BC04].

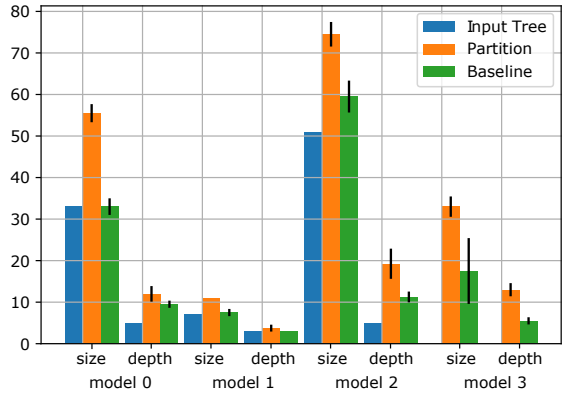


Figure 9: Average tree size and depth for baseline and partitioning methods (black lines: standard deviations).

8 REFERENCES

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

[BC04] Suzanne F Buchele and Richard H Crawford. Three-dimensional halfspace constructive solid geometry tree construction from implicit boundary representations. *Computer-Aided Design*, 36(11):1063–1073, 2004.

[BK73] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[BMV01] Pál Benkő, Ralph R Martin, and Tamás Várady. Algorithms for reverse engineering boundary representation models. *Computer-Aided Design*, 33(11):839–851, 2001.

[BTS+17] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 36(1):301–329, 2017.

[ES+03] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[FO82] Philippe Flajolet and Andrew M. Odlyzko. The average height of binary trees and other simple trees. *J. Comput. Syst. Sci.*, 25:171–213, 1982.

[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.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[KH13] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, July 2013.

[Mea82] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129 – 147, 1982.

[MG95] Brad L Miller and David E Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[Mit98] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[MM65] John W Moon and Leo Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, Mar 1965.

[OBA+03] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[Olv14] Peter Olver. *Introduction to partial differential equations*, volume 1. Springer, 2014.

[PH77] Franco P. Preparata and Se June Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

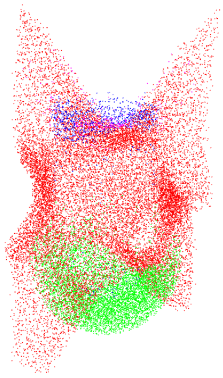
[Ric73] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973.

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

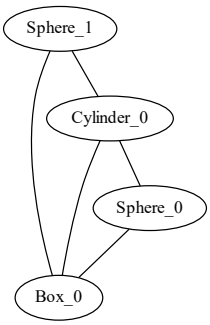
[SV91a] Vadim Shapiro and Donald L Vossler. Construction and optimization of csg representations. *Computer-Aided Design*, 23(1):4–20, 1991.

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

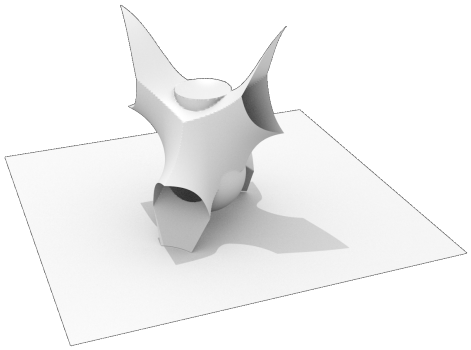
[SV93] Vadim Shapiro and Donald L Vossler. Separation for boundary to csg conversion.



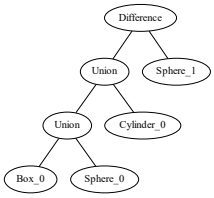
(a) Segmented point-set.



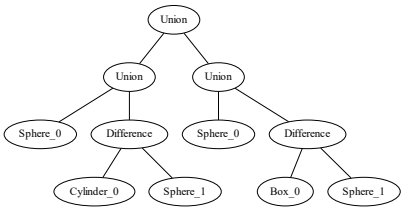
(b) PO-graph.



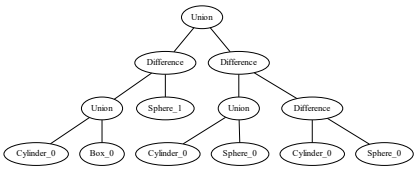
(c) Rendering of resulting model.



(d) CSG-tree ground-truth.

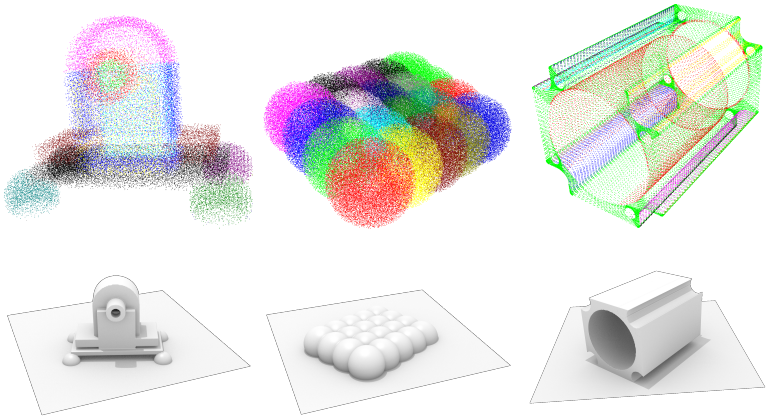


(e) CSG-tree from baseline.



(f) CSG-tree from partitioning scheme.

Figure 10: Results of all pipeline steps for model M1. The wing-like structure is based on a simple cube whose signed distance function is distorted by a sinusoidal term. This demonstrates the flexibility of the proposed approach in terms of possible model representations.



(a) Model M0. (b) Model M2. (c) Model M3.

Figure 11: Point clouds and renderings of resulting models M0, M2 and M3.

ACM Transactions on Graphics (TOG), 12(1):35–55, 1993.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer graphics forum*, 26(2):214–226, 2007.

[VMC97] Tamás Várady, Ralph R Martin, and Jordan Cox. Reverse engineering of geometric models-an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[XF14] Jianxiong Xiao and Yasutaka Furukawa. Reconstructing the world's museums. *International Journal of Computer Vision*, 110(3):243–258, Dec 2014.