Accelerating Evolutionary Object Construction Tree Recovery

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

Recovering Construction Trees 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 and noise robustness. Re-formulating the problem as a combinatorial optimization problem and solving it with an Evolutionary Algorithm mitigates these constraints at the cost of increased computation times. This paper proposes a detailed analysis of the associated optimization problem and a search space partitioning scheme that is able to accelerate Evolutionary Algorithm based Construction Tree recovery while exploiting parallelization capabilities of modern CPUs. The evaluation indicates a speed-up of up to $14.3x$ compared to the baseline approach while resulting tree sizes increase by TODO% on average.

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 heavily used in solid modeling is Constructive Solid Geometry (CSG). It describes complex rigid solids by a binary tree with regularized boolean set-operations (eg. union, intersection, subtraction) as inner nodes and primitive solids (e.g. cubes, spheres, cylinders and cones) as leaves. This 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 with methods based on algorithmic geometry that usually require exact geometric intersection computations [SV93, BC04]. These approaches are usualy restricted to a single model representation for primitives, e.g. a surface description that uses quadrics.

To overcome this constraint, 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 severest disadvantages of GA-based solutions are computation times of minutes and hours for comparably small models ($\leq$ 10 primitives) [FP16]. This issue is addressed by the approaches proposed 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 to 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 is structured as follows: (TODO)

2 RELATED WORKS

This work is related to different domains such as surface reconstruction from discrete point clouds, reverse engineering of solid models or conversion from B-Rep to CSG. In this section, we briefly list some related works in these domains.

Surface reconstruction

The problem of reconstructing a surface from a discrete point cloud has been the subject of lots of 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.

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 problem of B-Rep to CSG conversion was first investigated in two-dimension for linear polygons, then later extended by Shapiro 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]. These works rely on the fact that surfaces are composed of quadric surface patches. One issue of these algorithms is that their worst time complexity can be

exponential (the authors in [BC04] states a cubic time complexity in practice, while remarking that the worst time complexity could be exponential). 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 would impact the cellular classification on which these methods rely.

Point cloud to CSG reverse construction

Close to our work are methods that handles noisy and incomplete point clouds such as [SWK07] for fitting primitives and methods that try to convert them to higher level representation such as [FP16]. See also [BTS+17, Sections 7 and 8] for further references. One of our goals in this work is to improve the running time of the evolutionary algorithm used in [FP16] via geometric consideration (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 & 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 & primitive fitting:** The point cloud must be segmented and primitive parameters 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], or via evolutionary approaches such as [FP16] 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 [Wei09, 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, it can be approximated using the method of central differences:

$$\nabla f_p(x) \approx \frac{f_p(x-h) + f_p(x+h)}{2h}, \qquad (1)$$

where h is a small constant step size.

3.3 Boolean Set-Operations

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

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

In the following, the considered boolean set-operations are $\{\text{intersection}, \text{union}, \text{subtraction}\}$.

3.4 Evolutionary Algorithms

Evolutionary Algorithms are biology-inspired, stochastic metaheuristics for solving optimization problems.
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 Figure 1 for an overview.
Evolutionary Algorithms are especially useful for solving combinatorial optimization problems [ES+03].

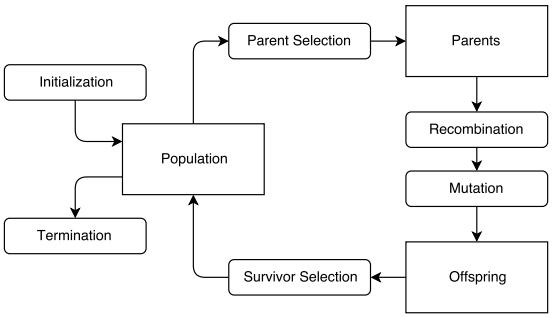


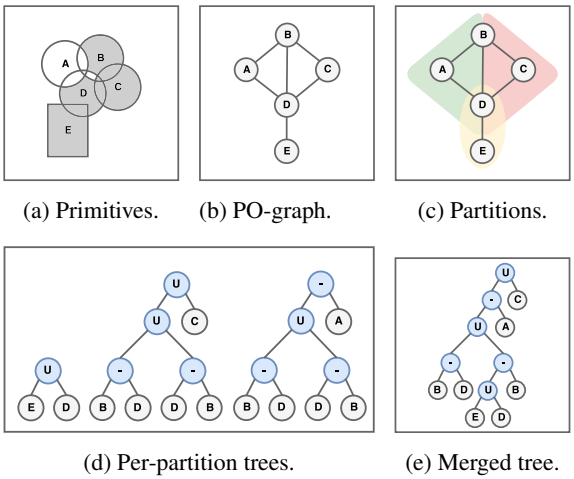
Figure 1: The optimization process described by an Evolutionary Algorithm (derived from [ES+03]).

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.
As input, a point-set of potentially noisy 3-d measurements of a connected geometric model together with segmented and fitted primitives is considered. The point-set might contain outliers and incomplete regions due to measurement errors that affect the result quality of the primitive reconstruction step.
The desired output is a CSG-tree that represents the scanned real-world model as accurately as possible. CSG-tree extraction approaches based on a GA [FP16] can handle the aforementioned inaccuracies but come with the disadvantage of high computation times.

5 CONCEPT

The basic idea for GA acceleration is to partition the search space in 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-extraction is then conducted on a per-partition level. Finally, resulting trees are combined in a subsequent merge step without loss of result quality.
An overview of the full CSG-extraction pipeline is depicted in Figure 2. Each of the following Chapters describes a particular pipeline step in detail, 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.

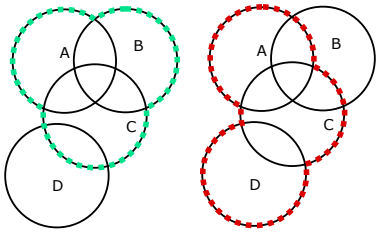
5.1 Primitive Overlap Graph Generation

For expressing spatial relationships between primitives, the 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 $i, j \in \{1, \ldots, n_p\} \wedge i \neq j$.
The PO-Graph is generated based on the location, orientation and geometric shape of the primitives, see Figure 2b for an example. Complex shapes can be approximated with simpler hull volumina like Axis-Aligned Bounding Boxes (AABBs) or Oriented Bounding Boxes (OBBs).

For better scaling, 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 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 Figure 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 maximal cliques in G, the Bron-Kerbosch Algorithm (BKA)[BK73] is employed due to its behavior on random graphs. It was shown experimentally [BK73] that computation times of BKA are almost independent on graph size for random graphs. In a worst case scenario (using Moon-Moser Graphs [MM65]), computation times are 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. The number of resulting partitions also depends on the accuracy of the hull approximation used for primitives during PO-graph generation: The inaccurate the approximation, the less partitions will emerge.



(a) correct (b) incorrect

Figure 3: In the incorrect partition (red), B influences the per-partition solution without being part of the partition.

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. As a basic building block for all acceleration schemes proposed in this paper serves a variant of the GA described in [FP16] with the objective function

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

where t is the tree candidate, S is the point-set and $size(t)$ is the number of nodes in tree t weighted by α. $d_i(t) = \beta \cdot f(s_i)$ is the signed distance between point s_i and the surface defined by tree t weighted by β. $\theta_i(t) = \gamma \cdot \arccos(\nabla \hat{f}(s_i) \cdot n_i)$ is the angle between the point normal n_i and the normalized gradient at position s_i weighted by γ. α, β and γ are user-controlled parameters. The first term in Equation 2 estimates how close the surface induced by c matches the point cloud, the second term penalizes large (in terms of number of nodes) trees. The third term penalizes large trees.
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

$$h_{max} \approx \sqrt{\pi \cdot |O|} \quad (3)$$

is used. It is based on the average tree height in binary trees for a given tree size [FO82] and worked well in all conducted experiments.
Each GA iteration i contains the following steps:

1. The population of the last iteration T_{i-1} is ranked according to Equation 2.
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 were selected from T_{i-1} via Tournament Selection [MMGG95] parametrized with k_{ts}. During crossover, the two candidates exchange randomly selected subtrees with a probability of γ_{cr}. The resulting two trees are then mutated. In the mutation process a randomly chosen subtree is replaced with a new randomly generated subtree with a probability of γ_{mu}. With a probability of $1 - \gamma_{mu}$, 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 computational expensive step in GA-based CSG-tree recovery is the evaluation of Equation 2 for each element of a candidate-set. Since evaluations can be conducted for each candidate independently, parallel processing schemes can be efficiently applied. In addition, The solution space partitioning allows for an additional per-partition parallelization strategy. Both options were implemented for multi-core processors and evaluated in Chapter 2.

5.4 Merge of Per-Partition Trees

Merging all trees corresponding to partitions in a single tree is not trivial. A simple union of all tree root nodes leads to incorrect results if primitives that are part of multiple cliques are not splitted, see Figure 5a for an example. Split operations on arbitrary primitive shapes tend to be complex and thus should be avoided, see e.g. Figure 5b. The proposed merge strategy does not need splits but instead tries to merge trees that have a common subtree. It consists of the following steps:

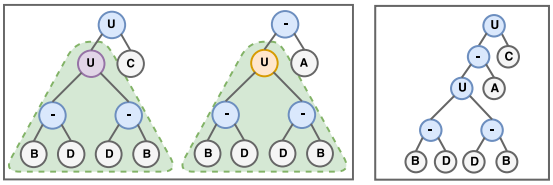
1. All trees are inserted in a list L.

2. Two trees t_0 and t_1 are removed from the end of L, and their largest common subtree t_{lcs} is computed. The subtree's leaf-set must be a subset of the leaf-sets of t_0 and t_1. A largest common subtree 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 Figure 4a for an example.
   If t_{lcs} is empty, t_1 is inserted at the begin of L and a new tree candidate t_1 is removed from the end of L. This step is then repeated.

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 is reached that way, it is considered a valid merge candidate. The found merge 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 Figure 4b for an example.

4. t_m is inserted at the end of L.

5. The merge process is continued until there is only a single node left in L. Since the model to reconstruct is by definition connected, the merge process always terminates.



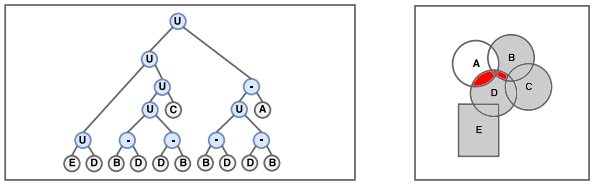
(a) (b)

Figure 4: (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.

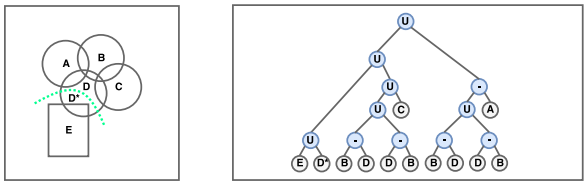
The merge process has an asymptotic computational complexity of $\mathcal{O}(|L|^2)$ since in worst case L has to be completely traversed for each merge. Note that the proposed algorithm does not guarantee to find the t_m with the minimal number of nodes possible.

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

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



(a) Wrong tree merge using union over all partition trees. Erroneous geometry in red.



(b) Correct tree merge using union over all partition trees with primitive splitting (green curve).

Figure 5: Merge strategies.

6 EVALUATION

The proposed partitioning scheme was evaluated on a laptop with quad core CPU and 16GB of RAM on four different models. For model 0,1 and 2, point clouds were generated by sampling a model surface induced by a pre-defined CSG-tree that served as ground-truth. Gaussian noise was added to sampling points to simulate measurement errors. Model 3 is based on real measurements, and primitive fitting was conducted using RANSAC [SWK07].

	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.

The three synthetic models were sampled with different rates. Table 1 contains model details. Baseline is the GA approach proposed in [FP16] and described in Chapter 5.3. The parameter set used for both, baseline and partitioning scheme, is listed in Table 2. The following combinations were evaluated:

Parameter Name	**Value**
Population size n_T	150
# Best parents n_b	2
Crossover probability μ_{cr}	0.3
Mutation probability μ_{mu}	0.3
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}	$\sqrt{\pi \cdot \lvert O \rvert}$

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

- 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).

Computation times

Timings for baseline and search space partitioning variants were measured for all models with high- and low-detail sampling (except for model 3 for which only a single point cloud exists). Measurements vary significantly for the same benchmark setting due to the inherently indeterministic behavior of GA-based methods. In order to deal with the high variance, each experiment was repeated 5 times.

In the following, timing results for all methods in combination with high-detail sampling are discussed. For model 0, SMTGA is the fastest method. It outperforms baseline by a factor of 15.3 (single-threaded, BST) and 7.5 (multi-threaded, BMTGA) on average. For model 1, 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 1 which eliminates the need for partitioning. For model 2, 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 3 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 in 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 a lot depending on partition sizes.
See Figures 6 and 7 for an overview of the results of the complete performance experiment. 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-tree generation, search space partitioning and tree merge make less then 1‰ of the total runtime.

Tree sizes and depths

Figure 9 contains average depths and sizes of resulting trees for baseline and partitioning variants. For the latter, tree depths have increased by 50-155% compared to the input tree, while for baseline approaches, an increase of only 0-80% is visible. Tree sizes show similar behavior: Partitioning variants produce 57-77% larger trees, while baseline approaches increase tree size by only 0-6%. This adverse behavior of partitioning variants is due to the final merge step: In each merge iteration, not those two trees with the largest common subtree of all trees in the merge list are merged but those that are neighbors in the merge list and have a common subtree of at least size 1. Since focus is on performance, this is acceptable behavior.

Scaling with respect to point cloud size

Figure 8 depicts measurement results for the ratio

$$\frac{\#\text{points}_{high}}{\#\text{points}_{low}} : \frac{\text{duration}_{high}}{\text{duration}_{low}} \tag{4}$$

which quantifies the dependency between point cloud size and corresponding computation times. It indicates that, for larger models (model 0 and 2), the fastest partitioning approach scales up to 1.9-times better than the best performing baseline approach with respect to point cloud size.

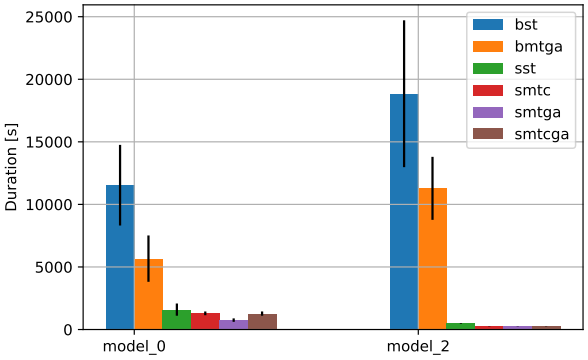


Figure 6: Timings for all approach combinations and models 0 and 2 with high-detail sampling. Vertical black lines indicate standard deviation.

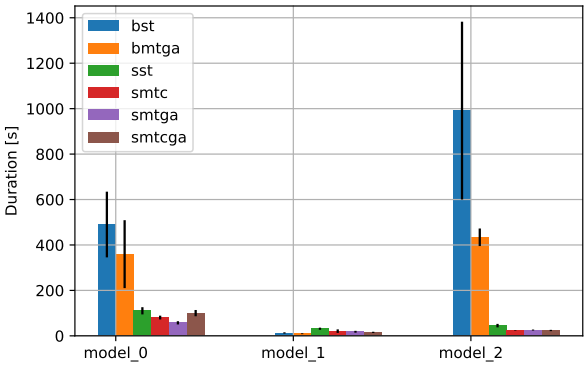


Figure 7: Timings for all approach combinations and models 1 and 3 with high-detail sampling. Vertical black lines indicate standard deviation.

7 CONCLUSION

TODO: Summary
The used GA might be implemented for massively parallel computing hardware and combined with the proposed partitioning approach. In addition, point cloud filtering based on sharp feature detection [**?**] could further increase performance. A decreased tree size in the partitioning approach could be achieved by improving the merge process.

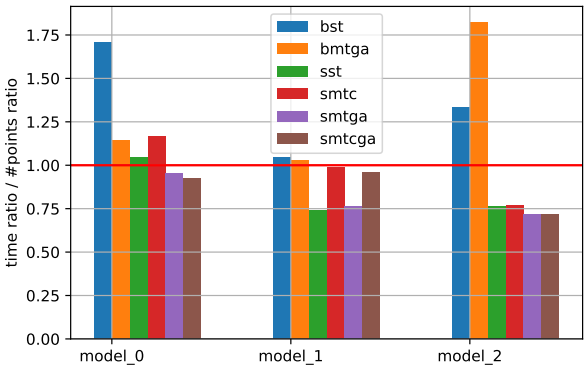


Figure 8: Ratio between high-detail and low-detail point cloud size factor and corresponding timing factors for all models (see Equation 4). The red line indicates linear scaling with a slope of 1 with respect to point cloud size. Model 3 is missing since it exists only in high-detail.

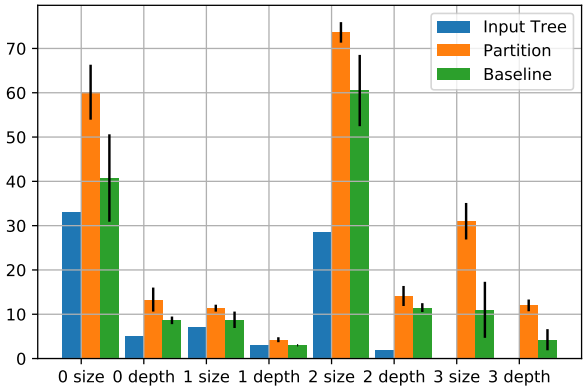


Figure 9: Average tree size and depth for baseline and search space partitioning methods for all models with high-detail sampling. Vertical black lines indicate standard deviation.

8 REFERENCES

[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] P. Benkó, R.R. Martin, and T. 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] M. Kazhdan and H. 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.

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

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

[MMGG95] Brad L. Miller, Brad L. Miller, David E. Goldberg, and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[OBA+03] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[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. *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] T. Varady, R.R. Martin, and J. Cox. Reverse engineering of geometric models-an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[Wei09] Daniel Weiss. *Geometry-based structural optimization on CAD specification trees*. PhD thesis, ETH Zurich, 2009.

Last page should be fully used by text, figures etc. Do not leave empty space, please.

Do not lock the PDF – additional text and info will be inserted, i.e. ISSN/ISBN etc.