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 combinatorical 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 14.5*x* 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 the 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. This poses a complex problem which is usually solved with a processing pipeline as illustrated in Figure **??**:

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].

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].

CSG-tree generation: TODO

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].

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 combinatorical 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 (≤ 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 group overlapping primitives and to compute partial per-group results that are later on merged to a single CSG-tree.
In particular, this paper makes the following contributions in the field of GA-based CSG-tree recovery from point clouds:

- 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)

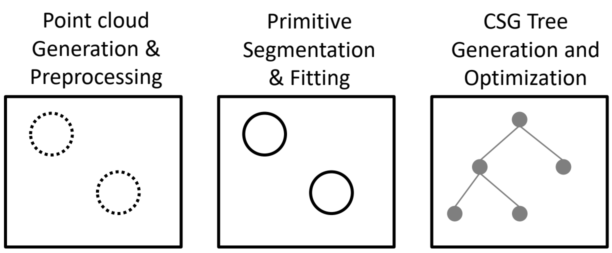


Figure 1: Insert caption to place caption below figure.

2 BACKGROUND

2.1 Constructive Solid Geometry

2.2 Primitive Description

Primitives are basic shapes located at CSG-tree leaves. A primitive p is fully described by its totally differentiable 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 to 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2.3 Boolean Set-Operations

The set-operations intersection, union, complement and subtraction can be 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: $\backslash(S_1, S_2) := \cap(/(S_1), S_2)$

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

2.4 Evolutionary Algorithms

TODO: Short description maybe with pseudo code.

3 RELATED WORK

The problem under consideration is related to the problem of boundary representation (B-Rep) to CSG conversion. It was first investigated in two-dimensional space 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]. An improved algorithm was later proposed by Buchele and Crawford in [BC04]. These methods rely on the fact that surfaces are composed of quadric surface patches (for computing separators, for factoring dominating halfspaces). The algorithm described in [SV91a] has exponential time complexity. The algorithm described in [BC04] has cubic (in the number of primitives) time complexity, however the authors remark that the worst time complexity could be exponential.
Another issue of these approaches is the handling of inexact representations. The 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 the methods described in [SV91a, SV93, BC04] rely.

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 based on spatial knowledge: The problem can be solved for groups of overlapping primitives separately. Results are then combined in a subsequent merge step without loss of result quality (see Section 5.3).

5.1 GA-Based CSG-Tree Extraction

As 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 $\leq h_{max}$. 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 main difference of the described GA variant compared to the GA proposed in [FP16] is the use of truncated solid primitives instead of implicitly defined surfaces of infinite extend in at least one dimension (eg. planes, cylinders). This simplifies the algorithm since no additional limiting primitives must be introduced prior to extraction.

5.2 Primitive Overlap Graph

For expressing spatial relationships between primitives, the Primitive Overlap (PO)-Graph is introduced. It represents spatial overlap between primitives using an undirected graph $G = (F, O)$, where $F = \{f_1, \ldots, f_{n_f}\}$ is the set of n_f primitives as vertices and O is the edge set that contains 2-tuples of overlapping primitives $o = (f_i, f_j)$, where $i, j \in \{1, \ldots, n_f\} \wedge i \neq j$.
The PO-Graph is generated based on the location, orientation and geometric shape of the primitives, see Figure 3b 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 $\mathcal{O}({n_f}^2)$ (overlap check between each primitive and each other primitive) to $\mathcal{O}(n_f \log(n_f))$ using hierarchical space partitioning schemes like Octrees [Mea82].

5.3 Search Space Partitioning

With known primitives and their spatial relations given by the PO-graph, the goal is now to to find independent search space partitions and to solve the problem for each partition separately in a divide-and-conquer manner. The proposed spatial scheme works on a per-primitive basis and finds all maximum complete subgraphs (maximum cliques) in the PO-graph where each containing primitive has at least one overlapping volume with all other primitives in the subgraph. This is equivalent to the problem of enumerating all maximal cliques in an undirected graph.
Having per-partition solutions, an additional merge step is necessary in order to obtain a single tree as a complete solution. The following details this pipeline:

1. **PO-Graph Generation** (Figure 3b)**:** The PO graph G is generated as described in Chapter 5.2 for a set of primitives (Figure 3a).

2. **Maximal Clique Extraction** (Figure 3c)**:** For finding the set C of maximal cliques in G, the Bron-Kerbosch Algorithm (BKA) [BK73] is employed. 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.

3. **Per-Clique Tree Recovery** (Figure 3d)**:** For each clique (subgraph of G) in C, the GA-based algorithm described in Chapter 5.1 is executed resulting in a CSG-tree for each clique in C. As an optimization for a clique size of 2, all 4 possible combinations are tried without running the GA.

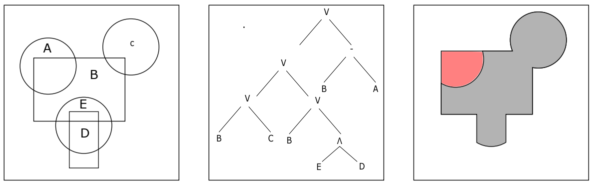
4. **Per-Clique Tree Merge** (Figure 3e)**:** Merging all trees corresponding to cliques in C in a single tree is not trivial. A simple union of all tree root nodes leads to incorrect results if primitives that are part of multiple cliques are not splitted, see Figure 2a for an example. Split operations on arbitrary primitive shapes tend to be complex and thus should be avoided, see e.g. Figure 2b. 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:

   - All trees are inserted in a list L_t.
   - Two trees t_0 and t_1 are removed from the end of L_t, 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. If t_{lcs} is empty, t_1 is inserted at the begin of L_t and a new tree candidate t_1 is removed from the end of L_t. This step is then repeated.
   - Each node in t_{tlc} exists twice: Once in t_0 and once in t_1. For each leaf node in t_{tlc} it is checked if its corresponding node in t_0 and t_1 is a merge candidate. This is done by traversing t_0 and t_1 from root to leaves following Algorithm 1. If the node is reached that way, it is a valid candidate in t_0 or t_1. Once a valid merge node candidate is found in one tree, it is replaced by the root of the other tree resulting in a merged tree t_m. If the merge node candidate is valid in both trees, the candidate of the larger tree is replaced by the root of the smaller tree.
   - t_m is inserted at the end of L_t.
   - The merge process is continued until there is only a single node left in L_t. Since the model to reconstruct is by definition connected, the merge process always terminates.

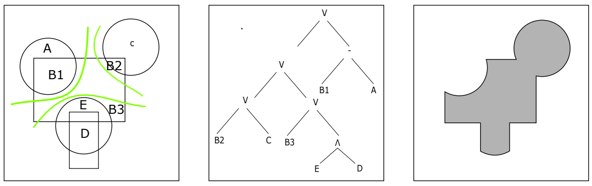
The merge process has a time complexity of $\mathcal{O}(|L_t|^2 \cdot \mathrm{size}(t))$ Note that the proposed algorithm does not guarantee to find the t_m with the minimal number of nodes possible.

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
isValid(t.root, node)
```



(a) Simple tree merge using union over all clique trees. Erroneous geometry in red.



(b) Tree merge using primitive splitting.

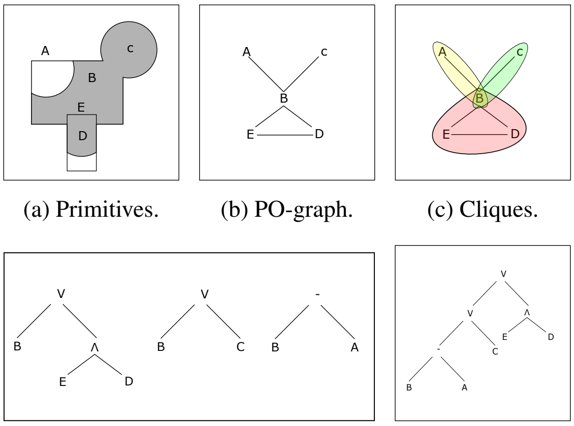
Figure 2: Merge strategies.

5.4 Parallelization

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.

6 EVALUATION

The proposed partitioning scheme was evaluated on a laptop with quad core CPU and 16GB of RAM. Pointclouds were generated by sampling a model surface induced by a pre-defined CSG-tree that serves as groundtruth. Gaussian noise was added to sampling points to simulate measurement errors.



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

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

Figure 3: The search space partitioning pipeline.

Two models were used with different point sampling rates, see Table 1 for details. Baseline is the GA ap-

	Model 1	**Model 2**	**Model 3**
# Primitives	17	4	
# Points (low)	11.3k	9.3k	
# Points (high)	156.4k	158.4k	

Table 1: Details on evaluated models. 'low' and 'high' indicate different sampling rates.

proach proposed in [FP16] and described in Chapter 5.1 The parameter set used for both, baseline and partitioning scheme, is listed in Table 2. The following combi-

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 partitioning approach.

nations were evaluated:

- Baseline: Single-threaded (BST), multi-threaded GA (BMTGA).
- Search Space Partitioning: Single-threaded (SST), per-clique multi-threaded (SMTC) multi-threaded GA (SMTGA), per-clique and GA multi-threaded (SMTCGA).

TODO: Add info for cliques in model 1 and 2.
Timings (Figure 4) for the baseline and partitioning variants were compared for model 1 and 2 with high-detail sampling.
TODO: Add discussion.
Figure 6 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.

Figure 5 shows the dependency between point cloud size and corresponding computation times. TODO: discussion.

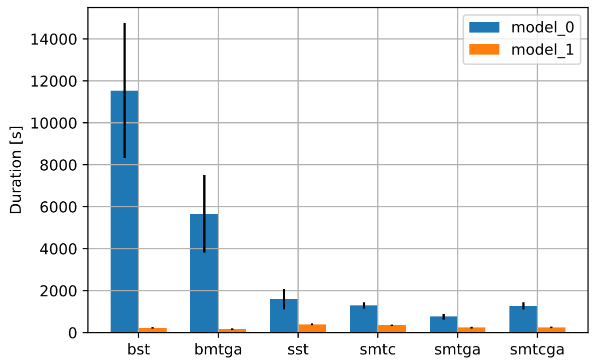


Figure 4: Timings for all approach combinations for model 1 and model 2 with high-detail sampling. Vertical black lines indicate standard deviation.

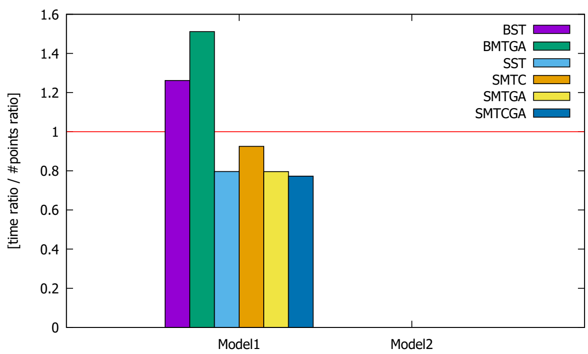


Figure 5: Ratio between high-detail and low-detail point cloud size factor and corresponding timing factors for model 1 and model 2.

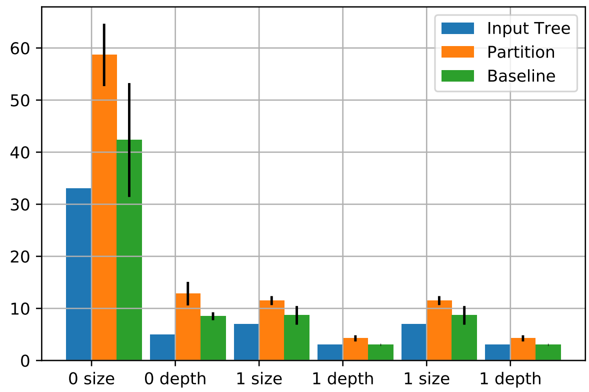


Figure 6: Average tree size and depth for baseline and search space partitioning method for model 1 and model 2 with high-detail sampling. Vertical black lines indicate standard deviation.

7 CONCLUSION

TODO: Summary
It is planned to implement the GA for massively parallel computing hardware and to combine it 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.

8 ACKNOWLEDGMENTS

9 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.

[CEJM13] Ã. Czabarka, P.L. ErdÅs, V. Johnson, and V. Moulton. Generating functions for multi-labeled trees. *Discrete Applied Mathematics*, 161(1):107 – 117, 2013.

[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.

[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*. 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.

[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. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[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.