Fast CSG Tree Reconstruction from Triangle Meshes via User-guided Partitioning

Daniel Ströter[1], Miguel Gonzalez-Nothnagel[1], Marcus Stegemann[1,2], Sebastian Besler[1,2], Johannes S. Mueller-Roemer[1,2], Markus Friedrich[3], Pierre-Alain Fayolle[4], André Stork[1,2]

[1]Technical University of Darmstadt,
[2]Fraunhofer IGD,
[3]Munich University of Applied Sciences,
[4]The University of Aizu,

Corresponding author: Daniel Ströter, daniel_jan.stroeter@tu-darmstadt.de

Abstract. The use of triangle meshes to represent geometric designs is ubiquitous in virtual prototyping. While geometric design is typically performed in computer-aided design (CAD) systems, triangle meshes usually do not allow for convenient shape modeling. Therefore, adjustment of the prototype's geometric design can be laborious and complex, if its CAD data is missing. To recover CAD data from a triangle mesh, we present a user-guided reconstruction method to obtain a constructive solid geometry (CSG) tree representing the geometry of the triangle mesh. In an interactive step, the user partitions identifiable design parts selecting mesh segments of common curvature. The user-guided partitioning allows for localized, per-partition reconstruction in a substantially reduced search space. To boost efficiency, we present GPU-accelerated primitive shape fitting as well as genetic evolution of CSG trees. As a result, our method enables fast CSG reconstruction improving run time performance by up to two orders of magnitude compared to state-of-the-art methods that require many minutes or even hours for complex reconstruction tasks. In addition, our method provides accurate reconstruction results reducing reconstruction errors by up to one order of magnitude compared to state-of-the-art methods.

Keywords: Virtual Prototyping, CSG, Reverse Engineering, Genetic Algorithms, GPU
DOI: https://doi.org/10.14733/cadaps.2027.36-54

1 INTRODUCTION

Today, virtual prototyping is an indispensable technique for rapid and cost efficient product development. Virtual prototyping typically involves many cycles of shape modeling, discretization, simulation, and analysis to investigate whether the prototype requires further shape editing or not. In many virtual prototyping workflows, triangle meshes originate from discretizing parametric representations defined in CAD systems. Using a CAD-native representation, users can easily edit geometric designs performing parametric editing operations such as

© 2027 U-turn Press LLC, http://www.cad-journal.net

extruding or filleting. However, if the parametric representation of the geometry is unavailable or lost, users cannot conveniently edit geometric designs, as triangle meshes typically provide only little to no parametric or semantic data.

Due to the need for recovery of parametric representations, a vast amount of literature addresses the reconstruction of CAD data from unstructured discrete representations [7]. While feature-based modeling of boundary representations is common in CAD, CSG combines parametric primitives with Boolean operations. Because of its small number of primitives and operators, the search space for reconstructing a CSG expression from a mesh is substantially smaller than that for reconstructing a feature-based modeling expression [35]. Despite the reduced search space, current methods require many minutes or even hours to reconstruct complex geometries. As the reconstruction process is non-deterministic, many attempts may be required to obtain a suitable result. Over the course of many attempts, a lot of time may be required to repeatedly evaluate complex reconstruction results and re-initiate the reconstruction process.

Reducing the search space by pre-segmenting the geometry [8] and partitioning the reconstruction [11] provides faster performance and improved robustness. Therefore, we propose a method that enables users to form partitions as a selected set of segments. Our method enables users to conveniently partition clearly identifiable design parts such as a strut that connects two assemblies. After user-guided partitioning, our method performs quick per-partition reconstructions. For faster performance, all steps in our pipeline from segmentation to CSG reconstruction are accelerated using the graphics processing unit (GPU). Our work provides the following contributions.

- An interactive partitioning strategy to enable fine-grained control over the CSG reconstruction process
- GPU-accelerated fitting and genetic evolution for CSG reconstruction
- A user-guided approach for fast and accurate CSG reconstruction

2 RELATED WORK

The literature on reconstruction of parametric data is extensive. We briefly review primitive fitting in Section 2.1 and discuss different CSG reconstruction types in Sections 2.2 to 2.5.

2.1 Primitive Fitting for Reconstruction

Random sample consensus (RANSAC) [26] is a popular choice for repeatedly fitting and creating primitive candidates robust to outliers. As alternate fitting and creation of primitives impose undesirable computational cost, our system relies on massively parallel fitting of primitives to a curvature-based pre-segmentation of the mesh. Besides RANSAC, a trained neural network can detect primitives. Li et al. [20] train a neural network to detect primitives and determine their parameters by fitting. Likewise, Yan et al. [36] predict parameters of the relations between neighboring points to detect sharp edges and geometric similarities. In contrast to trained neural networks, our method performs geometric fitting and does not rely on a training process.

While many methods focus on unbounded primitives, solid primitive detection provides conversion to bounded primitives. Du et al. [5] search for candidate cuboids by finding sets of three orthogonal pairs of parallel primitives. However, they consider all possible combinations of planes for cuboid detection, whereas our method involves a pre-filtering to reduce complexity. Friedrich et al. [12] estimate the height of cylinders using point-to-axis projection and construct cuboids using plane intersections, which we extend to cones.

2.2 Combinatorial CSG Tree Reconstruction

Early methods [2, 28] for the reconstruction of CSG trees are combinatorial techniques that perform iterative removal of dominating half-spaces from input boundary representations. As the resulting half-space CSG trees

© 2027 U-turn Press LLC, http://www.cad-journal.net

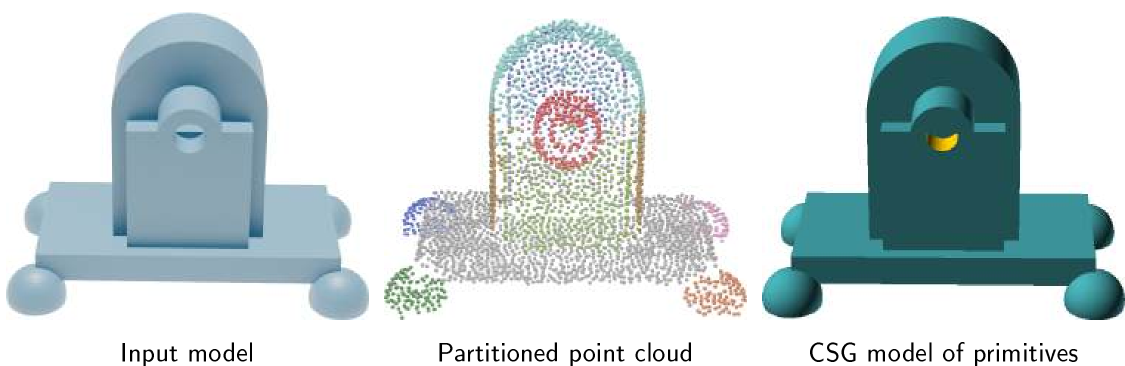


Figure 1: An input model (left) is sampled and partitioned by isolated primitive intersections (center), which allows for combining a CSG model (right) as a union of the partitions.

can explode in their size, the use of combinatorial techniques leads to slow run time performance for complex geometries. For shallower trees, Wu et al. [34] present a combination optimization of quadric primitives that includes a minimal description length concept to balance construction accuracy and tree size. While Wu et al. [34] efficiently reconstruct CSG trees from raw point clouds, we focus on guidable reconstruction from clean meshes using connectivity relations for partitioning.

2.3 Genetic Evolution of CSG Trees

To limit the size of CSG trees, Fayolle and Pasko [8] segment a point set S, apply primitive fitting to each segment, and evolve size-constrained CSG trees with a genetic algorithm (GA). The evolution is governed by evaluating the fitness of each candidate tree CSG using the following function:

$$\mathcal{E}(\mathsf{CSG}, S) = g(\mathsf{CSG}, S) - \lambda \cdot \mathsf{size}(\mathsf{CSG}), \tag{1}$$

where g rates the geometric approximation of CSG to S and the coefficient λ penalizes large CSG trees. The metric g is defined in terms of the tree's signed distance function (SDF) d_{CSG}:

$$g(\mathsf{CSG}, S) = \sum_{i=1}^{|S|} \mathrm{e}^{-(d_{\mathsf{CSG}}(\mathbf{x}_i)/\varepsilon_d)^2} + \mathrm{e}^{\frac{-1}{\alpha}\,\mathrm{acos}(\nabla \hat{d}_{\mathsf{CSG}}(\mathbf{x}_i)\cdot \mathbf{n}_i)}, \tag{2}$$

where $\nabla \hat{d}_{\mathsf{CSG}}$ is the normalized gradient of the SDF and ε_d as well as α are user-defined tuning parameters. Due to suboptimal segmentation and sequential processing, their method requires several hours for complex shapes. In contrast, our method allows for interactive partitioning based on segmentation of triangular meshes and exploits massively parallel processing for fast performance.

To partition the reconstruction, Friedrich et al. [11] compute a graph of intersecting primitives to isolate prime implicants to form partitions (see Fig. 1). After per-partition reconstruction, the CSG trees can be merged with union operations. This provides a merge-friendly partitioning but can lead to a coarse partitioning, whereas our user-guided method allows for more fine-grained partitioning for faster performance. As the use of a GA allows for extendable reconstruction systems, current reconstruction pipelines combine a GA with neural networks for fitting [12] or half-space removal [13]. Our focus is to provide a user-guided system that enables reconstruction of complex shapes, for which automated methods are inaccurate or impose slow performance.

© 2027 U-turn Press LLC, http://www.cad-journal.net

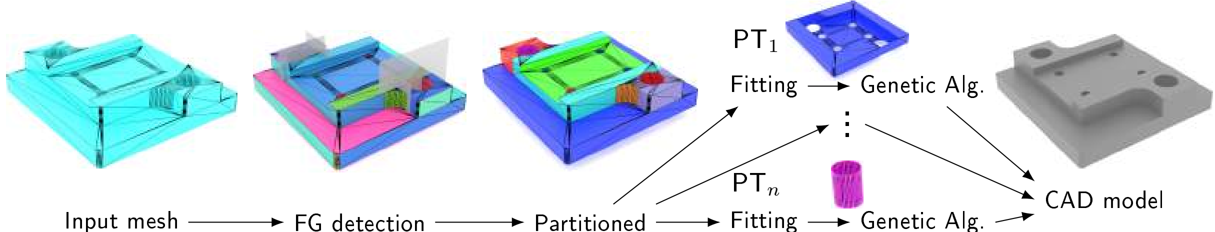


Figure 2: The input mesh is split into face groups (FGs) (separated by color) using automatic segmentation and cut planes. Then, the user selects FGs to form partitions. For each partition, our method fits primitives and finds a CSG tree. The CSG trees are combined into a CAD model.

2.4 Program Synthesis of a CSG Tree

In order to achieve a directed search for a CSG tree, Du et al.'s InverseCSG [5] relies on constraint-based program synthesis to obtain a CSG tree for a triangle mesh. InverseCSG relies on RANSAC [26] to fit primitives and the SKETCH [30] program synthesis system to reconstruct a CSG tree. A grammar for CSG trees and a set of constraints govern the program synthesis for reconstruction. While the directed search achieves robust and fast reconstruction, we show that fine-grained, per-partition reconstruction provides even faster and more accurate reconstruction. Feser et al. [9] show on 2D bitmaps experiments that program synthesis of CSG reconstruction can provide improved robustness and efficiency through a bottom-up synthesis strategy. The focus of our paper lies on per-partition reconstruction of 3D meshes to solve a significantly more complex problem than 2D CSG reconstruction.

2.5 Learning-based CSG Tree Reconstruction

A lot of recent literature address learning-based CSG reconstruction using neural network techniques. The early methods [29, 14] rely on supervised learning that typically depends on large, labeled data sets. As annotation of data with expert knowledge is difficult to scale, advanced methods [4, 23, 38] enable unsupervised learning for training on large-scale raw data. Neural networks for CSG tree reconstruction methods are prone to produce an abundant number of nodes and typically optimize a fixed tree structure, complicating downstream parametric editing. For more compact trees, Yu et al. [37] propose the use of dual complementary neural networks with weight dropout to optimize a fixed-order CSG tree. As fixed CSG tree structures can impose difficult interpretability and limited editability, our method enables users to choose the Boolean operation associated with each partition and finds a CSG tree for each partition without any restrictions on tree structure.

To avoid the need for a fixed tree structure, Liu et al. [21] present a gradient-based optimization of a boolean operator using fuzzy logic for reconstruction of crisp as well as smooth features, complementing learning-based CSG reconstruction. Although Liu et al. [21] optimize the choice of Boolean operations, the layout of the tree is fixed during the optimization and pruned at the end. This leads to complicated CSG trees for complex models, whereas our method does not fix the tree layout and enables users to specify the tree structure partially to facilitate downstream editing, albeit at the cost of an interactive step. While learning-based methods offer a high degree of automation, they typically rely on grids leading to inaccurate reconstruction of fine geometric details [3] and sharp features [39].

3 OUR USER-GUIDED CSG RECONSTRUCTION

We present a fast method for user-guided CSG tree recovery (see Fig. 2) from clean manifold triangle meshes. Our method generates CSG trees consisting of planes, spheres, cylinders, cones, and cuboids. Our first step

© 2027 U-turn Press LLC, http://www.cad-journal.net

is the pre-segmentation of the input mesh $\mathcal{T}$. For a semantic group of adjacent triangles, the mesh editing community has coined the term face group (FG), which we adopt in our work, because it describes our intent well: a user-selectable segment that is associated with a parametric shape. Our method is embedded in an interactive graphical system for user-guided partitioning. Users may bisect selected FGs along a cut plane to refine the segmentation. By selecting FGs, users select multiple triangles to form partitions $\mathsf{PT}_1, \ldots, \mathsf{PT}_n$. Each partition PT_i consists of m user-selected FGs: $\mathsf{PT}_i = \mathsf{FG}_1 \cup \cdots \cup \mathsf{FG}_m$. The resulting n partitions represent $\mathsf{CSG}_1, \ldots, \mathsf{CSG}_n$ sub trees of the intended CSG tree to limit the combinatorial complexity. For each partition, the user specifies a Boolean operation to combine the partition with the other partitions, as the interface of partitions often includes many mutually intersecting primitive shapes, e.g., multiple holes in a fillet. Our system defaults to choosing a union operation, because connected components can often be combined as unions [11]. Since the user-identified parts can usually be further decomposed into more fine-grained features, our system enables interactive recursive partitioning to reflect the user-hypothesized design intend [6]. As partitioning is performed recursively in a top-down manner, we obtain an expression CSG-expr to combine all sub trees. For each partition, we determine the best-fitting primitives to the FGs. Then, we use a GA to find a CSG sub tree for each partition:

$$(\mathsf{CSG}_1^*, \ldots, \mathsf{CSG}_n^*) = (\arg\max_{\mathsf{CSG}_1} \mathcal{E}_1(\mathsf{CSG}_1), \ldots, \arg\max_{\mathsf{CSG}_n} \mathcal{E}_n(\mathsf{CSG}_n)), \tag{3}$$

where $\mathcal{E}_i(\mathsf{CSG}) = \mathcal{E}(\mathsf{CSG}, \mathsf{PT}_i)$ evaluates how well CSG fits PT_i. The resulting sub trees are combined by the user-specified expression CSG-expr to obtain the final CSG tree:

$$\mathsf{CSG}^* = \mathsf{CSG\text{-}expr}(\mathsf{CSG}_1^*, \ldots, \mathsf{CSG}_n^*). \tag{4}$$

3.1 Face Group Segmentation

Our FG segmentation first classifies triangles based on principal curvature and subsequently performs region growing between triangles of similar curvature. Both of these steps are accelerated with the GPU.

3.1.1 Curvature Classification

We classify triangles based on an angle threshold φ_r for ridge edges and curvature thresholds $\varepsilon_{\mathsf{plane}}$, $\varepsilon_{\mathsf{sphere}}$, $\varepsilon^l_{\mathsf{cone}}$, and $\varepsilon^h_{\mathsf{cone}}$. By default, our system sets $\varphi_r = 15$, $\varepsilon_{\mathsf{plane}} = 0.05$, $\varepsilon_{\mathsf{sphere}} = 0.15$, $\varepsilon^h_{\mathsf{cone}} = 0.2$, and $\varepsilon^l_{\mathsf{cone}} = 0.8$. A ridge denotes a geometric feature edge, i.e., a sharp transition between adjacent surface regions. This differs from a mesh edge, which is purely topological and may lie inside a smooth surface patch. Our classification first detects geometric faces, ridges and corners in the mesh (see Figs. 3a to 3c). Due to its robustness, we use tensor voting [17] to assemble the quadric metric tensor $\mathbf{A_v}$ using the one-ring of surrounding vertices $\mathbf{v}_1, \ldots, \mathbf{v}_k$ and triangle (unit) normals $\mathbf{n}_1, \ldots \mathbf{n}_k$:

$$\mathbf{A_v} = \mathbf{n}_1 w_1 \mathbf{n}_1^\mathsf{T} + \cdots + \mathbf{n}_k w_k \mathbf{n}_k^\mathsf{T}, \text{ where } w_i = \left|\operatorname{atan2}(\|(\mathbf{v} - \mathbf{v}_i) \times (\mathbf{v} - \mathbf{v}_{i+1})\|_2, (\mathbf{v} - \mathbf{v}_i) \cdot (\mathbf{v} - \mathbf{v}_{i+1}))\right|. \tag{5}$$

After singular value decomposition of $\mathbf{A_v}$, we use Jiao's thresholding scheme [17] to classify $\mathbf{v}$ into face, ridge, and corner, setting φ_r as ridge angle and the other thresholds as detailed by Jiao. Next, we draw tentative boundaries between FGs, marking edges whose vertices are not classified as face, and where the triangles t and t' meet at an angle $\varphi_{t,t'} > \varphi_r$. With a pass over each $\mathbf{v}$'s one-ring of triangles, we count the marked edges that meet at $\mathbf{v}$ to obtain a tentative number $k_\mathbf{v}^{\mathsf{FG}}$ of adjacent FGs. To adjust normals in between FGs (see Fig. 3d), we compute tentative per-FG normals (see Fig. 3e) in another pass over each $\mathbf{v}$'s one-ring:

$$\mathbf{n}_\mathbf{v}^{\mathsf{FG}} = \sum_{i=1}^{k} \mathbf{n}_i w_{t_i}, \text{ where } w_{t_i} = \begin{cases} 0 & \text{if } t_i \text{ is not part of FG} \\ \operatorname{area}(t_i) & \text{otherwise.} \end{cases} \tag{6}$$

© 2027 U-turn Press LLC, http://www.cad-journal.net

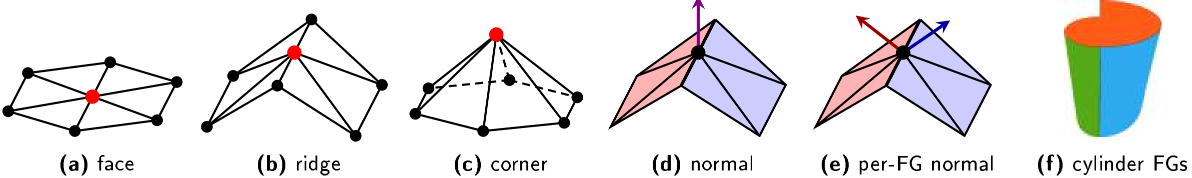


Figure 3: We detect faces (a), ridges (b), and corners (c). Instead of using vertex normals (d), normals for curvature are separated by ridges (e). With fitting, we detect transitions of the same primitive type (f).

With the per-FG normals, we compute per-triangle principal curvatures κ_{min} and κ_{max} using the method of Rusinkiewicz [25], as it is well-suited for GPUs [16]. Like Bénière et al. [1], we use principal curvature and mean curvature $H = \frac{1}{2}(\kappa_{\text{min}} + \kappa_{\text{max}})$ to classify triangles into:

planar if $|H| < \varepsilon_{\text{plane}}/s_{\text{AABB}}$,

spherical if $||\kappa_{\text{max}}| - |\kappa_{\text{min}}|| < H\varepsilon_{\text{sphere}}$,

conic if $|\kappa_{\text{max}}| > H\varepsilon^h_{\text{cone}}$ and $|\kappa_{\text{min}}| < H\varepsilon^l_{\text{cone}}$,

where s_{AABB} is the diagonal length of the axis aligned bounding box (AABB) of the input mesh. In contrast to Bénière et al. [1], we use two thresholds $\varepsilon^h_{\text{cone}}$ and $\varepsilon^l_{\text{cone}}$ for *conic* triangles. The threshold $\varepsilon^h_{\text{cone}}$ specifies the intended portion of κ_{max} to κ_{min} and $\varepsilon^l_{\text{cone}}$ controls how close κ_{min} should be to zero. We distinguish *spherical* and *conic* triangles into inverted and non-inverted depending on $\text{sign}(\kappa_{\text{max}})$.

3.1.2 Region Growing

To form segments we grow regions among triangles of similar curvature. We first present a simple method for highly efficient region growing and then a more robust method based on primitive fitting.

Simple region growing: Initially, we assign each triangle a unique FG associated by its index. Parallel flood filling propagates the index to adjacent triangles. The FG index of a triangle propagates to its edge neighbors, if the adjacent triangles are associated with smaller FG indices. The propagation is restricted to triangles of equal curvature classification to separate regions with different curvature. Eventually, the largest FG index is assigned to all the triangles within a region of common curvature (see Fig. 4).

Region growing based on primitive fitting: The simple region growing method does not detect transitions between differently parametrized shapes, e.g., a smaller cylinder blends into a larger cylinder (see Fig. 3f). Therefore, we provide a region growing method for more complex meshes. Similar to Rendon-Cardona et al. [24], we fit the primitives plane, sphere, cone, and cylinder. In contrast to Rendon-Cardona et al. [24], we choose seed triangles at random and the primitive by curvature type to reduce user interaction. Each triangle is marked as a candidate seed or as processed. We grow regions to adjacent triangles with the closest curvature to the seed. While for a plane the next triangle is the one whose normal is closest to the seed, the propagation is different for non-linear primitives. As the region should grow in the direction of maximal curvature to estimate the shape parameters as early as possible, we choose the adjacent triangle t' that minimizes:

$$\min_{t'} |\kappa^{\text{seed}}_{\text{max}} - \kappa^{t'}_{\text{max}}|(\mathbf{n}_{\text{seed}} \cdot \mathbf{n}_{t'}). \tag{7}$$

Once the region is large enough, we fit the seed's primitive to the region. While planar and spherical curvature types are each associated with one primitive type, conic curvature can indicate a cylinder or cone. For conic curvature, we fit a cylinder if all triangle normals are perpendicular to the region's axis; otherwise, we fit a cone. If primitive fitting succeeds, we mark all the region's triangles as processed. After fitting all regions, we assign each triangle the best-fitting primitive to resolve overlapping regions.

© 2027 U-turn Press LLC, http://www.cad-journal.net

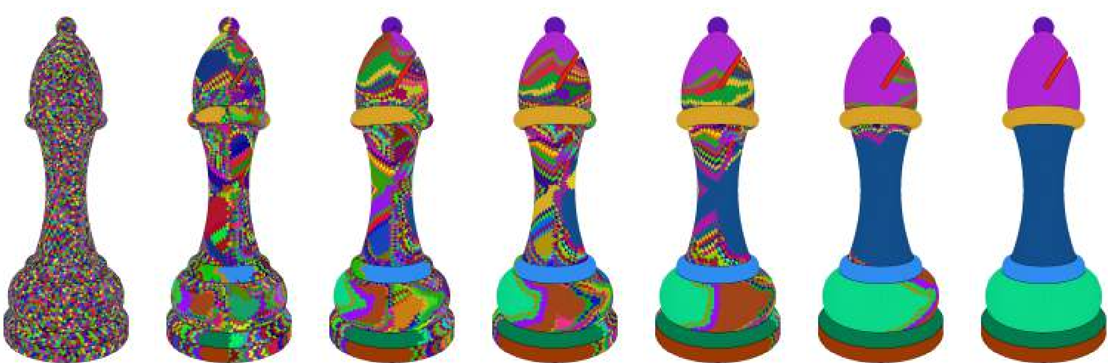


Figure 4: Visualization of the face group id propagation (from left to right) on a chess bishop model.

3.2 Primitive Fitting

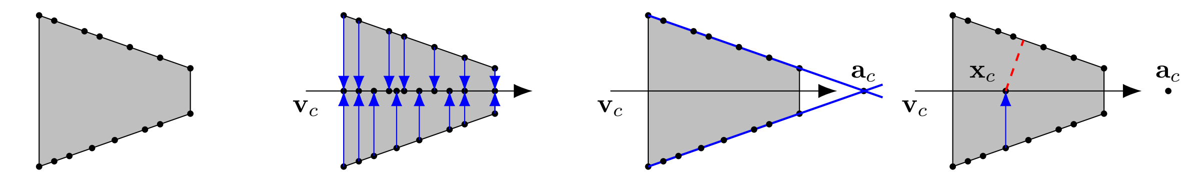
After FG detection and user-guided partitioning, our method determines a set of best fitting primitives for each FG of a partition $\mathsf{PT}_i = \mathsf{FG}_1 \cup \cdots \cup \mathsf{FG}_m$. We first fit plane, sphere, cylinder, and cone primitives and then construct bounded primitives. As the least-squares fitting plane includes the arithmetic average of points [27], we compute the centroid of FG vertices and the normalized sum of normals weighted by triangle area to fit a plane. We accept the resulting plane for the FG, if maximum distance to vertices is below a certain threshold $\varepsilon^d_{\mathsf{plane}}$ and $\frac{1}{N}\sum_{i=1}^{N}\theta_i < \cos(\varepsilon^n_{\mathsf{plane}})$ holds for the triangle's gradients $\theta_i = \max(\mathbf{n}_i \cdot \mathbf{n}, \mathbf{n}_i \cdot (-\mathbf{n}))$. Our system defaults to $\varepsilon^d_{\mathsf{plane}} = 0.01$ and $\varepsilon^n_{\mathsf{plane}} = 10°$. For the non-linear primitives, we use GPU-accelerated Gauss-Newton iterations [22] to find shape parameters $p_1, \ldots, p_n$ that minimize the squared distances evaluating the primitive's signed distance function (SDF) over the FG vertices $\mathbf{v}_i$:

$$\min_{p_1,\ldots,p_n} \sum_i d^2_{\mathsf{prim}}(\mathbf{v}_i, p_1, \ldots, p_n), \text{ where } d_{\mathsf{prim}} \text{ is the SDF of the primitive type.} \tag{8}$$

To optimize the parameter vector $\mathbf{p} = (p_1, \ldots, p_n)^\mathsf{T}$, we assemble the Jacobian matrix $\mathbf{J}(\mathbf{p})$ of d_{prim}. In each iteration i, we solve $\mathbf{J}(\mathbf{p}_i)^\mathsf{T}\mathbf{J}(\mathbf{p}_i)\mathbf{u}_i = -\mathbf{J}(\mathbf{p}_i)^\mathsf{T}\mathbf{d}(\mathbf{p}_i)$, where $\mathbf{d}(\mathbf{p}_i)$ is the vector of distances from vertices to the primitive. We perform updates $\mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{u}_i$ until convergence, i.e., $\|\mathbf{u}\| < 0.001$ and the arithmetic average of distance values falls below a threshold using $\varepsilon^{\mathsf{fit}}_{\mathsf{sphere}} = \varepsilon^{\mathsf{fit}}_{\mathsf{cone}} = 0.002$ and $\varepsilon^{\mathsf{fit}}_{\mathsf{cyl}} = 0.001$ as default values. Similarly to planes, we accept d_{prim} with parameters $\mathbf{p}$ for the FG, if the maximum distance of vertices to the primitives and the maximum deviation of vertex normals to SDF gradients/normals falls below pre-determined thresholds. Our system defaults to $\varepsilon^d_{\mathsf{sphere}} = \varepsilon^d_{\mathsf{cyl}} = \varepsilon^d_{\mathsf{cone}} = 0.01$ and $\varepsilon^n_{\mathsf{sphere}} = \varepsilon^n_{\mathsf{cyl}} = \varepsilon^n_{\mathsf{cone}} = 10°$. If no primitive type fits the FG, we perform the region growing based on primitive fitting within the FG.

As the convergence of Gauss-Newton iterations is sensitive to the initial guess of shape parameters, we initialize $\mathbf{p}$ depending on the geometry of the FG using the GPU. For fitting a sphere, linearization of the least squares term in Eq. (8) provides initial values for the radius and the centroid [10]. For fitting a cylinder, we first estimate its axis direction $\mathbf{v}_c$. The FG triangle normals are interpreted as points on the unit sphere and a plane is fitted to these points using the method of Ram et al. [22], because this fitting step depends only on points. The normal of the fitted plane provides an estimate for $\mathbf{v}_c$. Subsequently, we project the FG vertices onto the fitted plane. Like Li et al. [20], we fit a circle to the projected points. The center and radius of the fitted circle provide estimates for $\mathbf{x}_c$ and the cylinder's radius, respectively. For fitting a cone (see Fig. 5a), the axis direction $\mathbf{v}_c$ is estimated as for a cylinder (see Fig. 5b). Similar to Li et al. [20], we obtain an initial guess for the cone apex $\mathbf{a}_c$ computing the intersection point of tangent planes (see Fig. 5c). While Li et al. [20] focused on point clouds, we use FG triangle normals for the tangent planes. The opening

© 2027 U-turn Press LLC, http://www.cad-journal.net



(a) A cone and FG vertices (black dots). **(b)** The axis direction $\mathbf{v}_c$ is estimated by projecting the FG vertices. **(c)** The intersection of planes provides an estimate for the apex $\mathbf{a}_c$. **(d)** A projected point serves as $\mathbf{x}_c$ to optimize the distance (red line).

Figure 5: Initialization of shape parameters for fitting a cone.

angle φ_{cone} of the cone is estimated as the average angle between FG triangle normals and $\mathbf{v}_c$. As $\mathbf{a}_c$ can be far from the mesh, we use another point $\mathbf{x}_c$ as positional indicator [27]. To initialize $\mathbf{x}_c$ we project an arbitrary point from the fitted plane onto the axis $\mathbf{v}_c$ and compute the orthogonal distance to the estimated cone (see Fig. 5d). The Gauss-Newton iterations refine $\mathbf{x}_c$ and the orthogonal distance. When a fitting cone is found, $\mathbf{a}_c$ is recalculated from the refined shape parameters.

After primitive fitting, we construct bounded cylinders and cones from capping planes [12]. In addition, we check if the fitted planes form a cuboid using a GA and ghost plane detection by Friedrich et al. [12] with unique hash mapping of planes to avoid duplicate evaluation. Sometimes additional primitives are needed for reconstruction, e.g., a fillet on a cuboid. Therefore, we add the planes of a primitive's oriented bounding box [8] to the set of primitives. As an optimization for cylinders/cones, we do not add separating planes but a compound primitive composed of the difference of the plane and the cylinder/cone to reduce the search space.

3.3 CSG Tree Reconstruction

The final step of our method searches for a CSG tree CSG_i^* that combines the primitives fitted for the partition PT_i. To find a CSG tree we evolve a population of 150 random CSG trees using the GA of Fayolle and Pasko [8]. The evolution iteratively optimizes the population, keeping the n_{best} trees and forming new trees by crossover and mutation of trees selected by a tournament sort. We use the same fitness function as Fayolle and Pasko [8] to rate the trees. For fitness evaluation, we uniformly sample the FGs of the partition and project the sampling points onto the primitive surface [11] to obtain S. We implement pre-processing of trees as proposed by Friedrich et al. [11]. As a pre-process, we compute the SDF values and gradients for each single primitive. Especially for compound primitives, e.g., cuboids, this pre-computation relieves the GA from unnecessary tree traversal. If a partition is only associated with one or two primitives, we perform a quick evaluation of the possible combinations, instead of the GA.

During the GA, we evaluate the fitness of each CSG tree by bottom-up traversal starting at the leaves, i.e., the primitives. As the GA is the most expensive step, GPU-acceleration of fitness evaluation is crucial. For each tree node, we compute signed distances and gradients on the GPU and save the results to arrays of length $|S|$. For internal nodes, our traversal uses the arrays of both child nodes to compute new distances and gradients respecting the Boolean operation associated with the node. During traversal the allocated memory needs to accommodate the computed signed distances and gradients. After computation of a child node, the resulting arrays need to be saved until computation of the other child node, in order to compute the parent node. Since CSG forms a binary tree structure, it suffices to allocate one array for every tree level and one additional to keep the values of a sibling node. Therefore, we allocate $\ell + 1$ arrays for signed distances and gradients, where ℓ is the number of hierarchical levels in the CSG tree. The GA terminates early if a maximum number $n_{\max}$ of iterations is performed, no improvement in fitness can be achieved for n_{c} iterations, or the

© 2027 U-turn Press LLC, http://www.cad-journal.net

following fitness condition [11] is satisfied:

$$\frac{g^* - g_{\text{best}}(S)}{g^*} < p_e, \tag{9}$$

where $g^* = 2|S|$ is the target score, g_{best} is the geometry term of the best tree, and p_e is the target error percentage. Our system uses $n_{\max} = 10000$, $n_{\text{c}} = 100$, and $p_e = 5\%$ as default values. After reconstructing each partition, we convert the expression in Eq. (4) to the OpenSCAD format.

4 EVALUATION

We evaluate the efficiency and robustness of our semi-interactive method on a series of experiments. The experiments include meshes of different data sets including the most complex meshes from Du et al. [5] and Friedrich et al. [11], CAD meshes from the ABC data set [19], some meshes from the GrabCAD engineering [15] community, and some meshes from the Thingiverse 3D modeling community [33]. We implemented our method in C++ and CUDA and evaluated its performance on 50 meshes of varying complexity that can be retrieved online [31]. For identification, each mesh has a name consisting of a prefix to indicate the source data set and a numerical suffix. The prefix names INV, FM, ABC, GCAD, and THV map to data from InverseCSG, Friedrich et al., ABC data set, GrabCAD, and Thingiverse, respectively. We evaluate the solver run time of our per-partition reconstruction (see Section 4.1), the accuracy of the reconstruction results (see Section 4.2), and the overhead of interactive partitioning compared to state-of-the-art fully automatic reconstruction (see Section 4.3).

4.1 Run Time Performance

To evaluate the efficiency gains due to user-guided partitioning, we compare solver run times of our method with Liu et al. [21], InverseCSG [5], and the per-partition reconstruction of Friedrich et al. [11] using their reference implementations. We recorded reconstruction run times over 10 repetitions for each mesh on an Ubuntu 25.04 system with an AMD Ryzen 9 9900X (hyperthreading enabled) and an NVIDIA RTX 5080.

Comparing our reconstruction with the method of Liu et al. [21] shows the performance improvement of a fine-grained partitioned reconstruction over a neural network approach with few restrictions on tree structure. Setting the optimization parameters as suggested by Liu et al. [21] provides good reconstruction results for simple meshes training for about 15 minutes per mesh using GPU-accelerated training. However, most of the meshes in our test data set exhibit more complex features. Therefore, we performed optimization using 819600 samples, 30000 epochs, 2^{10} primitives, and marching cubes on a 512^3 grid to obtain more accurate results, whereas many features cannot be accurately reconstructed even by this setup. Due to optimizing a large CSG tree of fixed layout this leads to run times of about 6 hours per mesh, which is substantially slower than our method taking a few seconds per mesh.

Due to user-guided pre-partitioning, our method consistently provides faster median run times compared to InverseCSG [5] (see Fig. 6). On average, our method achieves 185× faster median run times than InverseCSG. While InverseCSG provides quick reconstruction times below 100 seconds for the majority of the meshes, it imposes long reconstruction times of several minutes up to 2.7 hours for 16 meshes. For these meshes, our method provides an average speedup of 465×. Except for THV02, the maximum run time of our method is substantially faster than the minimum run time of InverseCSG for the 16 most complex meshes. Over these meshes, the maximum run time of our method is on average 269× faster than the minimum run time of InverseCSG. The non-determinism of InverseCSG can lead to highly different minimum and maximum run times. On average, the maximum run time of InverseCSG is 3.8× slower than the minimum run time for reconstructing the same mesh. Our method provides a smaller average discrepancy of 3× and significantly

© 2027 U-turn Press LLC, http://www.cad-journal.net

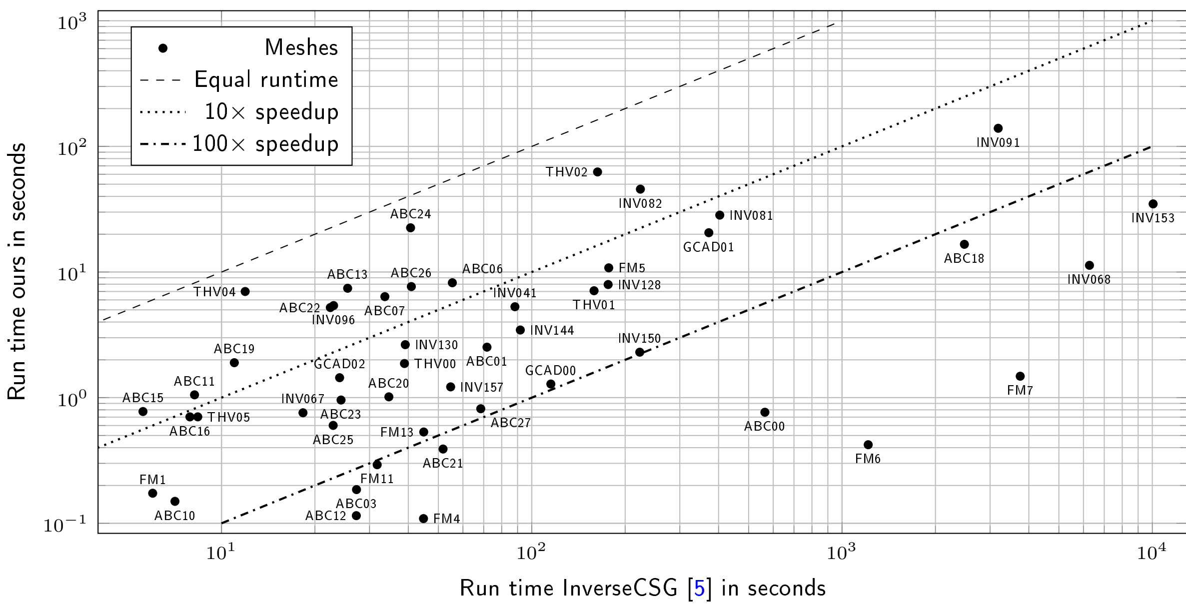
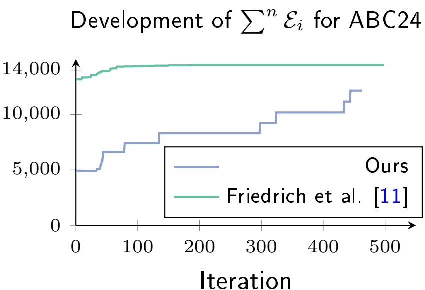


Figure 6: Median run times of our method and InverseCSG [5] (logarithmically scaled axes).

faster maximum run times below 180 seconds, which allows users to optimize their partitioning and run several reconstruction attempts, in order to achieve an accurate reconstruction.

We compare run times of our method with the method of Friedrich et al. [11] focusing on the performance improvement due to GPU-accelerated GA and more fine-grained partitioning. As their method depends on pre-fitted primitives, we provide it with the primitives fitted by our method. We measure run times of Friedrich et al.'s per-partition GA and our per-partition GA, comparing the impact of both partitioning schemes. Compared to Friedrich et al. [11], our method provides faster median run times except for four meshes that are reconstructed in milliseconds (see Fig. 7). On average, our method achieves 301× faster median run times. The faster performance is primarily due to more fine-grained partitioning, because it reduces the combinatorial complexity. As the smaller partitions result in a reduced search space, the GA more steadily improves the fitness function finding better CSG tree candidates more frequently (,e.g., see case ABC24 in the inset). In addition, Friedrich et al. [11] rely on point sampling for partitioning requiring a higher sampling resolution than our method. This also results in larger initial values of the fitness function compared to our method. Despite its per-partition reconstruction, Friedrich et al. [11] require more than 100 seconds for reconstructing half of the evaluated meshes, while our per-partition reconstruction achieves run times of one or two orders of magnitude faster. Hence, the partitioning of Friedrich et al. [11] is too restrictive to exploit the full optimization potential of per-partition reconstruction, whereas our user-guided partitioning enables exploitation of more fine-grained partitioning. Due to the coarse partitioning, Friedrich et al. [11] impose highly varying run times for each reconstruction attempt leading to 5.1× slower maximum run times than minimum run times on average for the same mesh. Therefore, computational costs are difficult to predict, when using the method of Friedrich et al. [11].



© 2027 U-turn Press LLC, http://www.cad-journal.net

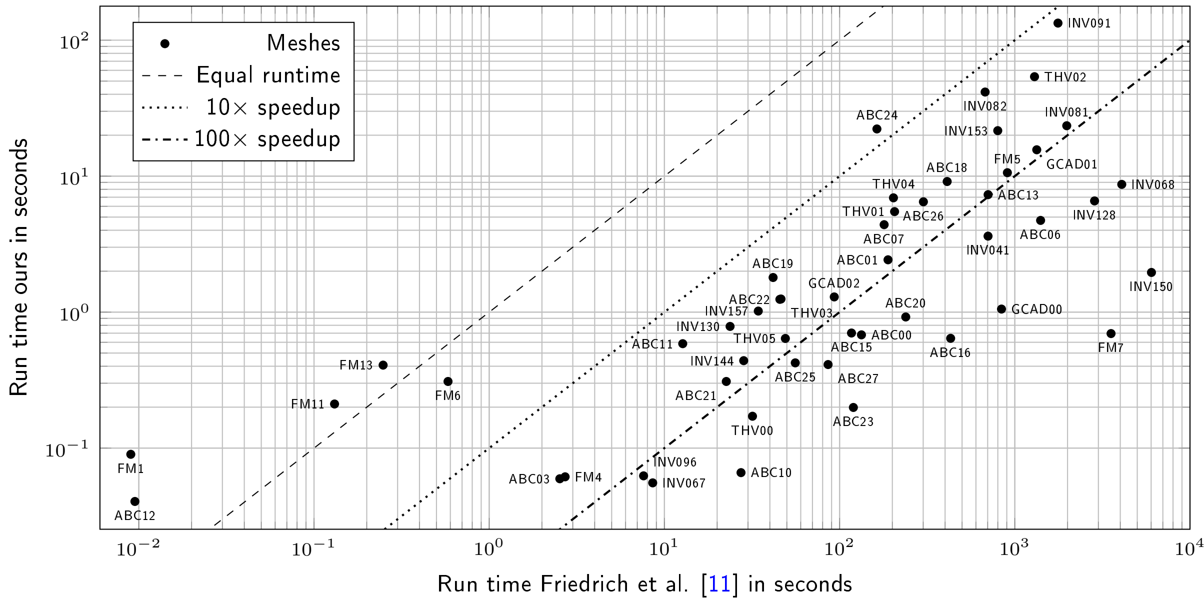


Figure 7: Median run times of our method and the method of Friedrich et al. [11] (logarithmically scaled axes).

The user-guided partitioning provides significant improvements of run time performance enabling the reconstruction of complex meshes in seconds, while other methods require several minutes or even hours. Since reconstructing a CSG tree involves a large search space and is non-deterministic, several reconstruction attempts may be necessary for complex meshes. In such a situation, our user-guided method provides tractable reconstruction and fast performance to quickly perform several reconstruction attempts.

4.2 Reconstruction Accuracy

Fast run time performance is meaningless, if the resulting CSG trees inaccurately reconstruct the geometry. Therefore, we evaluate the accuracy of the reconstruction results. In our quantitative evaluation, accuracy refers to geometric reconstruction accuracy, measured by the relative asymmetric Hausdorff distance $\mathcal{H}_{rel} = \mathcal{H}(\mathcal{T}, \text{CSG})/s_{\text{AABB}}$ from the input mesh $\mathcal{T}$ to the triangulated CSG result [5]. Furthermore, we report the mean distance $\overline{\mathcal{H}_{rel}}$, which reflects the average surface deviation, and the maximum distance $\max \mathcal{H}_{rel}$, which highlights large local errors such as missing primitives or incorrectly combined features. As $\mathcal{H}_{rel}$ is indicative of reconstruction errors, we use the terms mean error and maximum error to refer to $\overline{\mathcal{H}_{rel}}$ and $\max \mathcal{H}_{rel}$, respectively. Other aspects of reconstruction quality such as compactness, semantic editability, and preservation of design intent are discussed qualitatively as they are difficult to quantitatively assess.

Table 1: Sum of mean and max $\mathcal{H}_{\text{rel}}$ over 50 meshes using InverseCSG [5] the method of Friedrich et al. [11] and our method. The lowest error is provided in boldface.

Metric	InverseCSG [5]	Friedrich et al. [11]	Ours
$\sum \overline{\mathcal{H}_{rel}}$	1.474e-2	1.478e-2	**2.32e-3**
$\sum \max \mathcal{H}_{rel}$	2.6e-1	6.6e-1	**3.4e-2**

© 2027 U-turn Press LLC, http://www.cad-journal.net

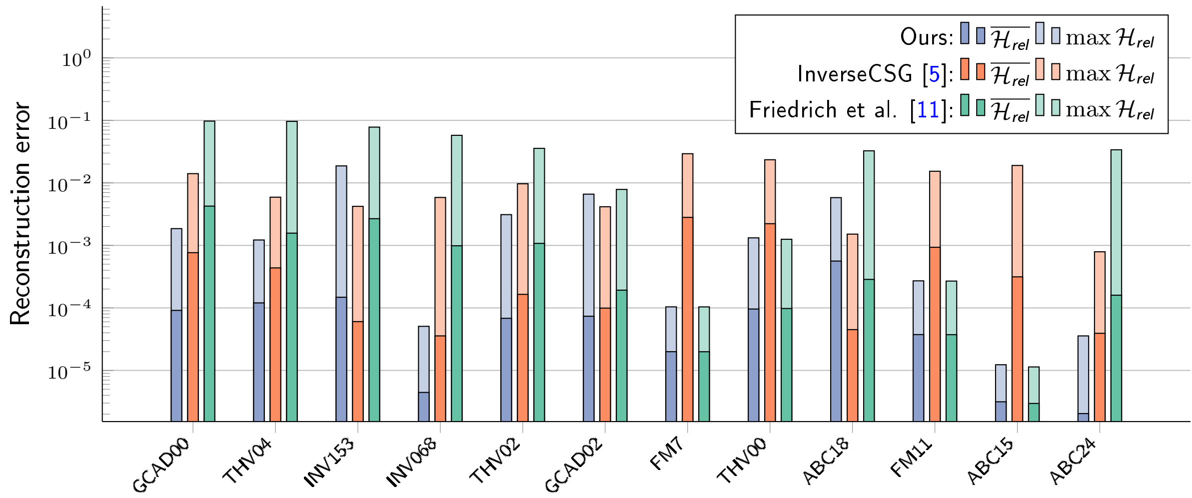


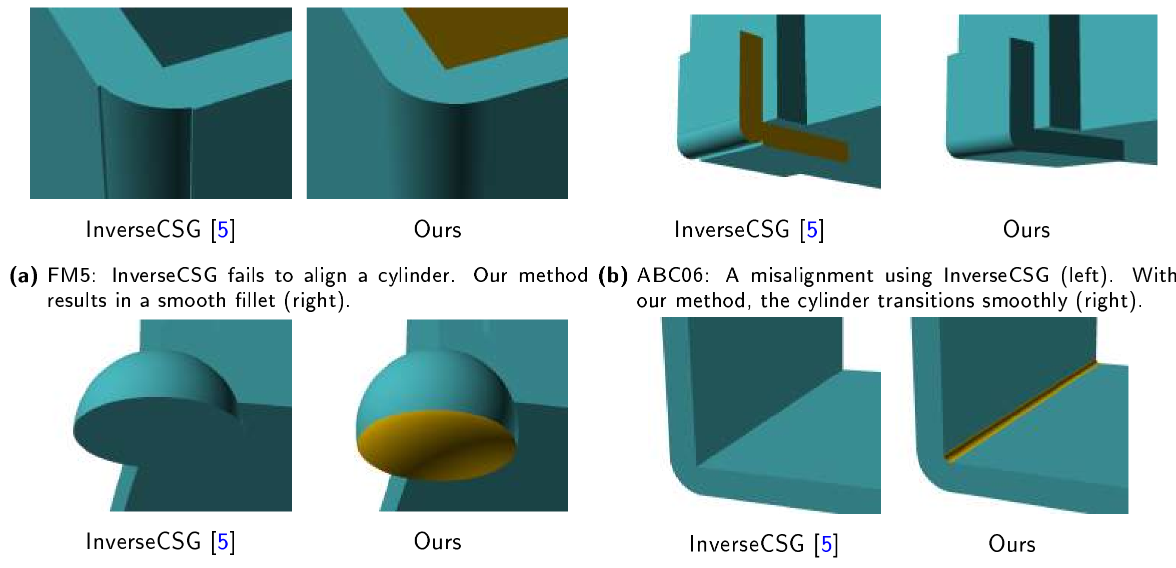
Figure 8: Mean and Max reconstruction error for the 12 most complex reconstruction cases.

As Liu et al. [21] optimize a large CSG tree of fixed layout requiring long reconstruction times on complex meshes, we do not provide a full comparison over the 50 meshes, but show on examples that our method more accurately reconstructs fine features. Over the 50 meshes in our test set, we compare the reconstruction results of our method with the results of InverseCSG [5] and the method of Friedrich et al. [11]. Table 1 provides the sums of $\overline{\mathcal{H}_{rel}}$ and $\max \mathcal{H}_{rel}$ over the 50 meshes. Our method achieves the lowest error indicating more accurate reconstruction results. While the sum of $\overline{\mathcal{H}_{rel}}$ is similar for InverseCSG and Friedrich et al., the sum of $\max \mathcal{H}_{rel}$ is significantly larger for Friedrich et al. indicating missing primitives in resulting CSG trees. Figure 8 plots the reconstruction error for the 12 most complex meshes. Due to the fine-grained pre-partitioning, our method tends to provide lower reconstruction errors for the complex reconstruction cases. The method of Friedrich et al. sometimes misses a primitive, leading to large $\max \mathcal{H}_{rel}$, while in some cases their intersection graph method admits a fine-grained partitioning, leading to results similar to our method. InverseCSG rarely misses a primitive leading to low $\max \mathcal{H}_{rel}$. However, its parameterization of the primitives is sometimes inaccurate, increasing $\overline{\mathcal{H}_{rel}}$ due to more deviations from the input surface. We discuss the comparison with each method in detail and visualize different reconstruction cases.

On average over the meshes, our user-guided per-partition reconstruction achieves 35× lower $\overline{\mathcal{H}_{rel}}$ and 127× lower $\max \mathcal{H}_{rel}$ than InverseCSG. Therefore, our user-guided partitioning enables more accurate CSG reconstruction. Figure 9 showcases some inaccuracies in the results of InverseCSG that our method reconstructs more accurately. As InverseCSG sometimes parametrizes the primitives inaccurately, misalignment can occur at primitive transitions (see Figs. 9a and 9b). In contrast, our per-FG primitive fitting isolates regions of common curvature enabling more accurate primitive parametrization and smoother transitions. In addition, InverseCSG may not detect small curvature changes, while our fitting method enables reconstruction of nuanced features (see Fig. 9c). In rare cases, InverseCSG fails to detect fine-grained features such as small fillets, while our method finds it (see Fig. 9d). As a result, the pre-segmentation into FGs for primitive fitting achieves more accurate parametrization, because fitting to an isolated surface of common curvature provides more accuracy.

Compared to Friedrich et al. [11], our user-guided partitioning leads to 14× lower $\overline{\mathcal{H}_{rel}}$ and 108× lower $\max \mathcal{H}_{rel}$. Since we provide both methods with the same fitted primitives, the more fine-grained partitioning of our user-guided method substantially reduces the search space, leading to more accurate reconstruction results. Due to the larger search space, Friedrich et al. may not find a suitable combination of primitives to reconstruct

© 2027 U-turn Press LLC, http://www.cad-journal.net



(a) FM5: InverseCSG fails to align a cylinder. Our method results in a smooth fillet (right).

(b) ABC06: A misalignment using InverseCSG (left). With our method, the cylinder transitions smoothly (right).

(c) FM11: InverseCSG reconstructs the hemisphere with a flat bottom (left), while our method reproduces the spherical, hollowed out bottom (right).

(d) ABC27: In some cases, InverseCSG misses a fillet (left), while our method finds it (right).

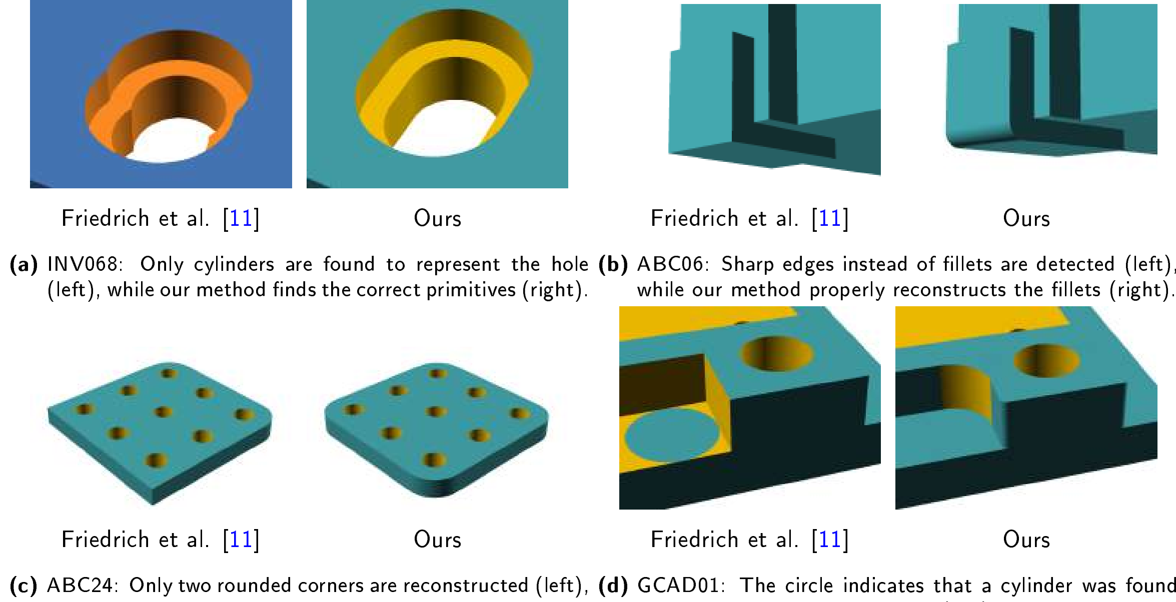
Figure 9: Several comparisons of reconstruction results using InverseCSG [5] and our method.

geometric features (see Fig. 10a). Especially, the reconstruction of fillets can fail in the larger search space (see Figs. 10b and 10c), because the partitioning of Friedrich et al. is restricted to prime implicants and fillets require correct combination of a compound primitive. In addition, browsing in a larger search space can lead the GA to terminate in a suboptimal CSG tree that uses only a few of the input primitives and does not reconstruct important geometric features (see Fig. 10d). As a result, reducing the search space is crucial for using a GA for CSG reconstruction and our method provides a user-guided, effective way to obtain a fine-grained partitioning.

To evaluate the reconstruction accuracy with a learning-based method without strict constraints on tree structure, we compare reconstruction results of our method with Liu et al. [21]. We used a grid resolution of 512^3 significantly more fine grained than the geometric features to avoid inaccuracies due to the final marching cubes mesh conversion. We sample each model with 819600 samples to avoid inaccuracies due to low sampling rates. We perform 30000 epochs for stabilizing the loss to avoid inaccuracies due to premature termination. As Liu et al. optimize on a large initial CSG tree, their method imposes long reconstruction times for complex meshes. Therefore, we showcase four reconstructions in Fig. 11. Despite the high-resolution sampling and long optimization, the method of Liu et al. struggles to reconstruct fine features. Especially features that are significantly smaller than the overall mesh are inaccurately reconstructed or completely missing (see Fig. 11a). Our method exhibits more accurate reconstruction of fine features. In addition, the method of Liu et al. can result in inaccuracies when many features of different scale are spatially dense (see Figs. 11b and 11c). In extreme cases, this can lead to many missing features (see Fig. 11d). Our method more robustly reconstructs features of varying scale.

Our method successfully reconstructs many geometric features, but can fail to reconstruct highly complex features. Figure 12 presents reconstruction results for two rounded features that meet in a sharp transition. Friedrich et al. [11] fail to reconstruct both of the rounded features producing only sharp ridges. InverseCSG [5] finds both rounded features but inaccurate parametrization prevents reconstruction of the correct sharp

© 2027 U-turn Press LLC, http://www.cad-journal.net



(a) INV068: Only cylinders are found to represent the hole (left), while our method finds the correct primitives (right).

(b) ABC06: Sharp edges instead of fillets are detected (left), while our method properly reconstructs the fillets (right).

(c) ABC24: Only two rounded corners are reconstructed (left), while our method reconstructs the four rounded corners (right).

(d) GCAD01: The circle indicates that a cylinder was found but not used in the CSG tree (left), while our method reconstructs all the rounded features (right).

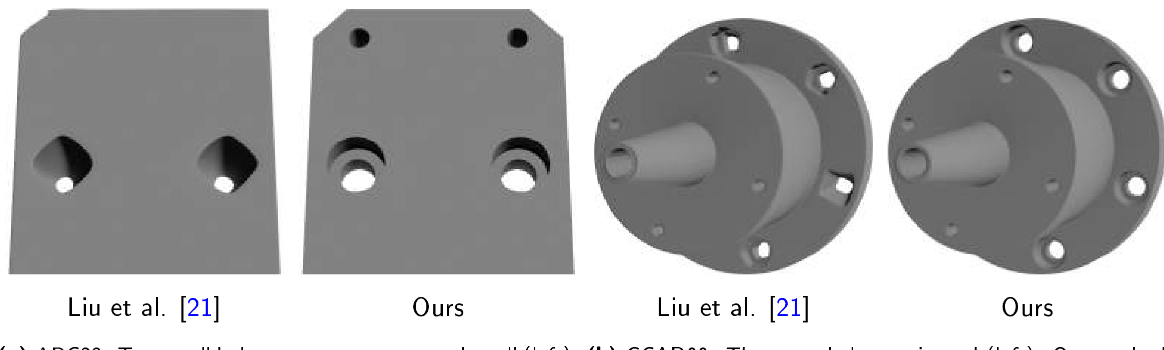
Figure 10: Several comparisons of reconstruction results of Friedrich et al. [11] and our method.

transition, producing disjoint curved features. Our method can correctly parametrize the curved features and approximately reconstruct their transition, but fails to restrict the primitives to the region of transition and primitives erroneously blend into the adjacent features producing a false reconstruction result.

4.3 Interactive Partitioning Time

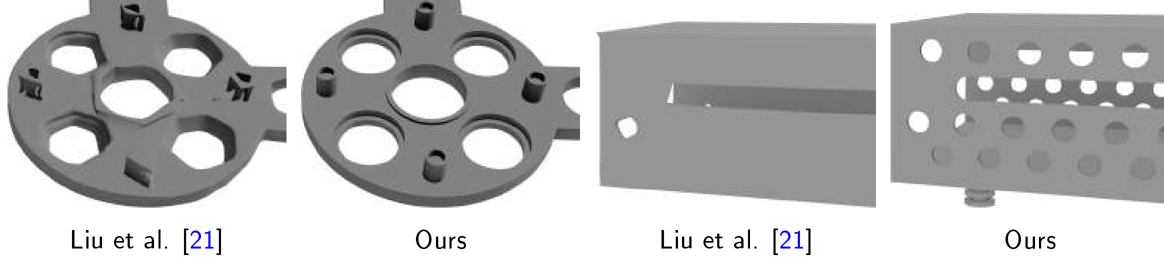
As our method involves user interaction, we demonstrate on the FM07 and INV153 meshes that user-guided partitioning is beneficial for complex meshes (see Fig. 13). For the FM07, we reconstruct all partitions in 1.5 seconds, while InverseCSG requires 62 minutes and Friedrich et al. [11] require 59 minutes. After automated segmentation, we specify cut planes to separate FGs at different assemblies in 1.5 minutes and perform user-guided partitioning in 4 minutes. The 5.5 minutes for user-interaction are significantly lower than the reconstruction times of InverseCSG and Friedrich et al. [11]. In terms of reconstruction error, our method provides a 140$\times$ lower error than InverseCSG, while the method of Friedrich et al. [11] provides the same result as our method. For the INV153, we reconstruct most features correctly in 35 seconds, while InverseCSG requires 167 minutes to provide a result of 2.4$\times$ lower reconstruction error compared to our method. The method of Friedrich et al. [11] requires 13 minutes for the INV153 mesh but misses many features leading to an 18$\times$ larger mean error compared to our method. Using our method, we specify cut planes in 4 minutes to split segments at different assemblies and then perform user-guided partitioning in 14 minutes. This results in 18 minutes of user interaction which is significantly lower than the reconstruction time of InverseCSG. Friedrich et al. provide a result in less than the interaction time using our pre-fitted primitives, but the result is of low accuracy and imposes significantly more reconstruction errors (see Fig. 8).

© 2027 U-turn Press LLC, http://www.cad-journal.net



(a) ABC23: Two small holes are not reconstructed at all (left), while our method reconstructs all holes accurately (right).

(b) GCAD00: The outer holes are jagged (left). Our method accurately parametrizes the holes (right).



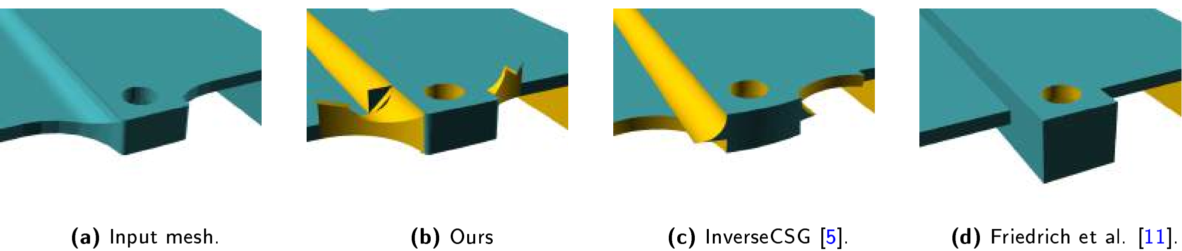
(c) FM07: The holes are jagged and the pins are distorted (left). Our method provides a more accurate result (right).

(d) INV150: Many holes are missing (left), while our method reconstructs more features (right).

Figure 11: Several comparisons of reconstruction results of Liu et al. [21] and our method.

5 CONCLUSIONS

We have presented a user-guided method for CSG tree recovery that provides fast performance and accurate results. In conjunction with parallelization, the reduced combinatorial complexity leads to a speedup of up to two orders of magnitude. Our face group segmentation into segments of similar curvature has proven to be a suitable basis for accurate primitive fitting. We have achieved more accurate parametrization of primitive shapes due to fitting non-overlapping segments. In addition, our face group segmentation has shown to be effective for convenient user-guided partitioning, enabling fine-grained partitioning of the mesh. Fine-grained partitioning of the reconstruction has demonstrated substantial reduction of the search space. The reduced search space of our per-partition reconstruction facilitates finding a suitable CSG tree. In combination with more accurate primitive fitting, this results in up to one order of magnitude lower mean errors. As our method



(a) Input mesh. **(b)** Ours **(c)** InverseCSG [5]. **(d)** Friedrich et al. [11].

Figure 12: Comparison of how the different methods reconstructed a particularly complex part of the mesh THV02.

© 2027 U-turn Press LLC, http://www.cad-journal.net

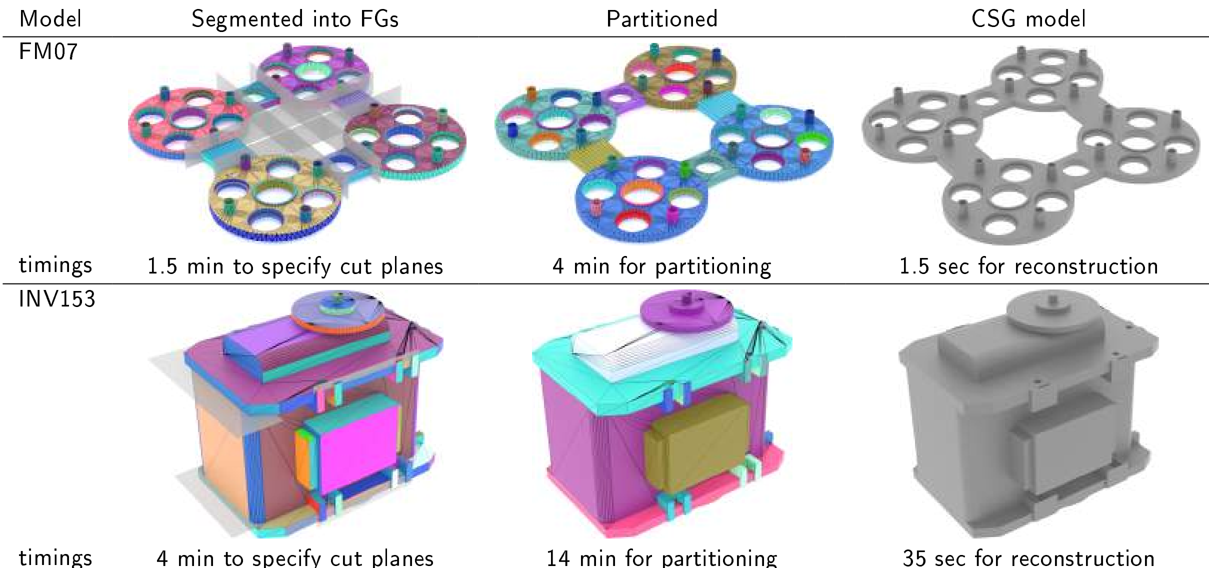


Figure 13: Reconstruction of FM07 (top) and INV153 (bottom) meshes. Each model is segmented into FGs (left) and partitioned by the user (center). Our method then reconstructs a CSG model (right).

provides substantially faster solver performance and improved accuracy, it is worth investing a few minutes for the user-guided partitioning. Our CSG reconstruction system enables the reconstruction of complex geometries, for which fully automatic methods impose suboptimal accuracy and slow performance.

5.1 Limitations

As our method relies on user interaction, the quality of the partitioning depends on the user's knowledge of CSG modeling. Thus, our system is targeted at users with basic knowledge of CSG. Another limitation of our method is that it is not resistant to noise and cannot be applied to raw point clouds. Like previous methods, our method is non-deterministic and cannot guaranty finding a suitable CSG tree.

5.2 Future Work

An interesting avenue could be to automatically join face groups into partitions, as they provide a good basis for primitive fitting. As per-partition reconstruction is not limited to genetic algorithms, it would be interesting to explore the potential of partitioning for alternative CSG reconstruction methods. In addition, our method could be combined with parametric morphing [18] and mesh editing systems [32] for faster virtual prototyping cycles. To this end, our method could obtain a parametric representation to enable design optimization.

Daniel Ströter, https://orcid.org/0000-0002-2672-7377
Marcus Stegemann, https://orcid.org/0009-0001-4118-5318
Sebastian Besler, https://orcid.org/0000-0002-6347-2481
Johannes S. Mueller-Roemer, https://orcid.org/0000-0002-0712-0457
Markus Friedrich, https://orcid.org/0000-0001-5719-3198
Pierre-Alain Fayolle, https://orcid.org/0000-0003-4723-6208
André Stork, https://orcid.org/0000-0001-7538-7674

© 2027 U-turn Press LLC, http://www.cad-journal.net

REFERENCES

[1] Bénière, R.; Subsol, G.; Gesquière, G.; Le Breton, F.; Puech, W.: A comprehensive process of reverse engineering from 3D meshes to CAD models. Computer-Aided Design, 45(11), 1382–1393, 2013. http://doi.org/10.1016/j.cad.2013.06.004.

[2] Buchele, S.F.; Crawford, R.H.: Three-dimensional halfspace constructive solid geometry tree construction from implicit boundary representations. In Proceedings of the eighth ACM symposium on Solid modeling and applications, SM03, 135–144. ACM, 2003. http://doi.org/10.1145/781606.781629.

[3] Chen, J.; Shen, Z.; Zhao, M.; Jia, X.; Yan, D.M.; Wang, W.: FR-CSG: Fast and reliable modeling for constructive solid geometry. IEEE Transactions on Visualization and Computer Graphics, 31(9), 5869–5883, 2025. http://doi.org/10.1109/TVCG.2024.3481278.

[4] Chen, Z.; Tagliasacchi, A.; Zhang, H.: BSP-Net: Generating compact meshes via binary space partitioning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[5] Du, T.; Inala, J.P.; Pu, Y.; Spielberg, A.; Schulz, A.; Rus, D.; Solar-Lezama, A.; Matusik, W.: InverseCSG: automatic conversion of 3D models to CSG trees. ACM Transactions on Graphics, 37(6), 1–16, 2018. http://doi.org/10.1145/3272127.3275006.

[6] Durupt, A.; Remy, S.; Ducellier, G.: KBRE: A knowledge based reverse engineering for mechanical components. Computer-Aided Design and Applications, 7(2), 279–289, 2010. http://doi.org/10.3722/cadaps.2010.279-289.

[7] Fayolle, P.A.; Friedrich, M.: A survey of methods for converting unstructured data to CSG models. Computer-Aided Design, 168, 103655, 2024. http://doi.org/10.1016/j.cad.2023.103655.

[8] Fayolle, P.A.; Pasko, A.: An evolutionary approach to the extraction of object construction trees from 3D point clouds. Computer-Aided Design, 74, 1–17, 2016. http://doi.org/10.1016/j.cad.2016.01.001.

[9] Feser, J.; Dillig, I.; Solar-Lezama, A.: Metric program synthesis for inverse CSG. Tech. rep., Massachusetts Institute of Technology, 2022.

[10] Forbes, A.B.: Least-squares best-fit geometric elements. Tech. rep., National Physical Laboratory, Teddington, United Kingdome, 1991. http://eprintspublications.npl.co.uk/id/eprint/5050.

[11] Friedrich, M.; Fayolle, P.A.; Gabor, T.; Linnhoff-Popien, C.: Optimizing evolutionary CSG tree extraction. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'19, 1183–1191. ACM, 2019. http://doi.org/10.1145/3321707.3321771.

[12] Friedrich, M.; Illium, S.; Fayolle, P.A.; Linnhoff-Popien, C.: A hybrid approach for segmenting and fitting solid primitives to 3D point clouds. In Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 38–48. SCITEPRESS, 2020. http://doi.org/10.5220/0008870600380048.

[13] Friedrich, M.; Illium, S.; Fayolle, P.A.; Linnhoff-Popien, C.: CSG Tree Extraction from 3D Point Clouds and Meshes Using a Hybrid Approach, 53–79. Springer International Publishing, 2022. http://doi.org/10.1007/978-3-030-94893-1_3.

[14] Genova, K.; Cole, F.; Vlasic, D.; Sarna, A.; Freeman, W.T.; Funkhouser, T.: Learning shape templates with structured implicit functions. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[15] GrabCAD: GrabCAD community library, 2025. https://grabcad.com/library.

[16] Griffin, W.; Wang, Y.; Berrios, D.; Olano, M.: GPU curvature estimation on deformable meshes. In Symposium on Interactive 3D Graphics and Games, I3D' 11, 159–166. ACM, 2011. http://doi.org/10.1145/1944745.1944772.

© 2027 U-turn Press LLC, http://www.cad-journal.net

[17] Jiao, X.: Volume and Feature Preservation in Surface Mesh Optimization, 359–373. Springer Berlin Heidelberg, 2006. http://doi.org/10.1007/978-3-540-34958-7_21.

[18] Kabalan, A.; Casenave, F.; Bordeu, F.; Ehrlacher, V.; Ern, A.: Elasticity-based morphing technique and application to reduced-order modeling. Applied Mathematical Modelling, 141, 115929, 2025. http://doi.org/10.1016/j.apm.2025.115929.

[19] Koch, S.; Matveev, A.; Jiang, Z.; Williams, F.; Artemov, A.; Burnaev, E.; Alexa, M.; Zorin, D.; Panozzo, D.: ABC: A big CAD model dataset for geometric deep learning. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019. http://doi.org/10.1109/CVPR.2019.00983.

[20] Li, L.; Sung, M.; Dubrovina, A.; Yi, L.; Guibas, L.J.: Supervised fitting of geometric primitives to 3D point clouds. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2647–2655. IEEE, 2019. http://doi.org/10.1109/cvpr.2019.00276.

[21] Liu, H.T.D.; Agrawala, M.; Yuksel, C.; Omernick, T.; Misra, V.; Corazza, S.; Mcguire, M.; Zordan, V.: A unified differentiable boolean operator with fuzzy logic. In ACM SIGGRAPH 2024 Conference Papers, SIGGRAPH '24. ACM, 2024. http://doi.org/10.1145/3641519.3657484.

[22] Panyam Mohan Ram, M.; Kurfess, T.R.; Tucker, T.M.: Least-squares fitting of analytic primitives on a GPU. Journal of Manufacturing Systems, 27(3), 130–135, 2008. http://doi.org/10.1016/j.jmsy.2008.07.004.

[23] Ren, D.; Zheng, J.; Cai, J.; Li, J.; Jiang, H.; Cai, Z.; Zhang, J.; Pan, L.; Zhang, M.; Zhao, H.; Yi, S.: CSG-Stump: A learning friendly CSG-like representation for interpretable shape parsing. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 12478–12487, 2021.

[24] Rendon-Cardona, C.; Correa, J.; Acosta, D.A.; Ruiz-Salguero, O.: Analytic form fitting in poor triangular meshes. Algorithms, 14(11), 304, 2021. http://doi.org/10.3390/a14110304.

[25] Rusinkiewicz, S.: Estimating curvatures and their derivatives on triangle meshes. In Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004., 486–493. IEEE, 2004. http://doi.org/10.1109/tdpvt.2004.1335277.

[26] Schnabel, R.; Wahl, R.; Klein, R.: Efficient ransac for point-cloud shape detection. Computer Graphics Forum, 26(2), 214–226, 2007. http://doi.org/10.1111/j.1467-8659.2007.01016.x.

[27] Shakarji, C.: Least-squares fitting algorithms of the nist algorithm testing system. Journal of Research of the National Institute of Standards and Technology, 103(6), 633, 1998. http://doi.org/10.6028/jres.103.043.

[28] Shapiro, V.; Vossler, D.L.: Construction and optimization of CSG representations. Computer-Aided Design, 23(1), 4–20, 1991. http://doi.org/10.1016/0010-4485(91)90077-a.

[29] Sharma, G.; Goyal, R.; Liu, D.; Kalogerakis, E.; Maji, S.: Csgnet: Neural shape parser for constructive solid geometry. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[30] Solar-Lezama, A.; Tancau, L.; Bodik, R.; Seshia, S.; Saraswat, V.: Combinatorial sketching for finite programs. In Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, ASPLOS06. ACM, 2006. http://doi.org/10.1145/1168857.1168907.

[31] Ströter, D.: Evaluation data for fast CSG tree reconstruction from triangle meshes via user-guided partitioning, 2026. http://doi.org/10.6084/m9.figshare.31891300.v1.

[32] Ströter, D.; Krispel, U.; Mueller-Roemer, J.; Fellner, D.: TEdit: A distributed tetrahedral mesh editor with immediate simulation feedback. In Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, 271–277. SCITEPRESS, 2021. http://doi.org/10.5220/0010544402710277.

[33] Thingiverse: Thingiverse - digital designs for physical objects, 2025. https://www.thingiverse.com/.

© 2027 U-turn Press LLC, http://www.cad-journal.net

[34] Wu, Q.; Xu, K.; Wang, J.: Constructing 3D CSG models from 3D raw point clouds. Computer Graphics Forum, 37(5), 221–232, 2018. http://doi.org/10.1111/cgf.13504.

[35] Xu, X.; Peng, W.; Cheng, C.Y.; Willis, K.D.; Ritchie, D.: Inferring cad modeling sequences using zone graphs. In 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 6058–6066. IEEE, 2021. http://doi.org/10.1109/cvpr46437.2021.00600.

[36] Yan, S.; Yang, Z.; Ma, C.; Huang, H.; Vouga, E.; Huang, Q.: Hpnet: Deep primitive segmentation using hybrid representations. In 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2733–2742. IEEE, 2021. http://doi.org/10.1109/iccv48922.2021.00275.

[37] Yu, F.; Chen, Q.; Tanveer, M.; Mahdavi Amiri, A.; Zhang, H.: D^2CSG: Unsupervised learning of compact CSG trees with dual complements and dropouts. In Advances in Neural Information Processing Systems, vol. 36, 22807–22819. Curran Associates, Inc., 2023.

[38] Yu, F.; Chen, Z.; Li, M.; Sanghi, A.; Shayani, H.; Mahdavi-Amiri, A.; Zhang, H.: CAPRI-Net: Learning compact CAD shapes with adaptive primitive assembly. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 11768–11778, 2022.

[39] Zhang, Z.; Zhao, M.; Shen, Z.; Wang, Y.; Jia, X.; Yan, D.M.: Interactive reverse engineering of CAD models. Computer Aided Geometric Design, 111, 102339, 2024. http://doi.org/10.1016/j.cagd.2024.102339.

© 2027 U-turn Press LLC, http://www.cad-journal.net