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

Authors:
Daniel Ströter, daniel.stroeter@gris.tu-darmstadt.de, Technical University of Darmstadt
Miguel Gonzalez-Nothnagel, Technical University of Darmstadt
Marcus Stegemann, Fraunhofer IGD and Technical University of Darmstadt
Sebastian Besler, Fraunhofer IGD and Technical University of Darmstadt
Johannes S. Mueller-Roemer, Fraunhofer IGD and Technical University of Darmstadt
Markus Friedrich, Munich University of Applied Sciences
Pierre-Alain Fayolle, University of Aizu
André Stork, Fraunhofer IGD and Technical University of Darmstadt

Keywords:
Virtual Prototyping, CSG, Reverse Engineering, Genetic Algorithms, GPU

DOI: 10.14733/cadconfP.2026.32-38

Introduction:
The use of triangle meshes to represent geometric designs is ubiquitous in virtual prototyping. Many triangle meshes originate from discretizing parametric representations defined in computer-aided design (CAD) tools. Using a CAD-native parametric representation, users can easily edit geometric designs. However, if the parametric representation of the geometry is unavailable or lost, users cannot conveniently edit geometric designs, as triangle meshes include little to no parametric or semantic data.

A vast amount of literature addresses the reconstruction of CAD data [1]. While feature-based modeling of boundary representations is common in CAD, constructive solid geometry (CSG) combines parametric primitives with Boolean operations. Due to its low 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 [2]. 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.

Reducing the search space by pre-segmenting the geometry [3] and partitioning the reconstruction [5] provides faster and more robust reconstruction. Therefore, we propose a user-guided 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 connecting two assemblies. After user-guided partitioning, our method performs a per-partition reconstruction. For faster performance, all steps in our pipeline from segmentation to CSG reconstruction are accelerated using the graphics processing unit (GPU). We achieve speedups of up to two orders of magnitude compared to state-of-the-art methods.

Our User-guided CSG Reconstruction:
We present a fast method for user-guided CSG tree recovery from manifold triangle meshes (see Fig. 1). The first step is the pre-segmentation of the input mesh. For a semantic group of adjacent triangles,

© 2026 U-turn Press LLC, http://www.cad-conference.net

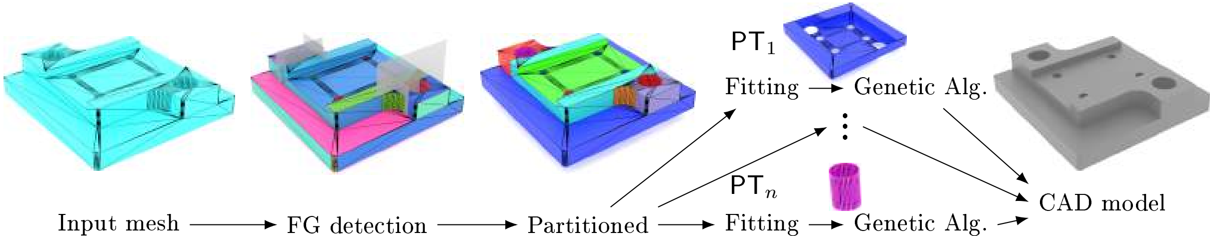

Fig. 1: The input mesh is split into face groups (FG)s (separated by color) using automatic segmentation and cut planes. Then, the user selects FGs to form partitions. For each partition, our method fits primitives and finds a CSG tree. The CSG trees are combined into a CAD model.

the mesh editing community has coined the term face group (FG), which we adopt in our work, because it describes our intent well: user-selectable segments that are associated with parametric shapes. 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 [5]. As partitioning is done 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 genetic algorithm (GA) to find a CSG sub tree for each partition:

$$(\mathsf{CSG}_1^*, \ldots, \mathsf{CSG}_n^*) = (\arg\max_{\mathsf{CSG}_1} \mathcal{E}_1(\mathsf{CSG}_1), \ldots, \arg\max_{\mathsf{CSG}_n} \mathcal{E}_n(\mathsf{CSG}_n)), \tag{2.1}$$

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:

$$\mathsf{CSG}^* = \mathsf{CSG\text{-}expr}(\mathsf{CSG}_1^*, \ldots, \mathsf{CSG}_n^*). \tag{2.2}$$

Segmentation into Face Groups:

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.

Curvature classification

We classify triangles based on an angle threshold φ_r for ridge edges and curvature thresholds $\varepsilon_{\text{plane}}$, $\varepsilon^l_{\text{cone}}$, $\varepsilon^h_{\text{cone}}$, and $\varepsilon_{\text{sphere}}$. Our classification first detects geometric faces, ridges and corners in the mesh (see Figs. 2a to 2c). Due to its robustness, we use tensor voting [6] 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 $\mathbf{n}_1, \ldots \mathbf{n}_k$:

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

After singular value decomposition of $\mathbf{A_v}$, we use Jiao's thresholding scheme [6] 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

© 2026 U-turn Press LLC, http://www.cad-conference.net

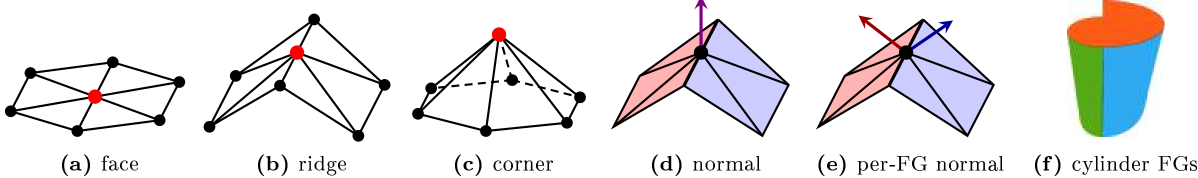

Fig. 2: We detect faces (a), ridges (b), and corners (c). Instead of using vertex normals (d), normals for curvature are separated by ridges (e). With fitting, we detect transitions of the same primitive type (f).

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. 2d), we compute tentative per-FG normals (see Fig. 2e) in another pass over each $\mathbf{v}$'s one-ring:

$$\mathbf{n}_{\mathbf{v}}^{\mathsf{FG}} = \sum_{i=1}^{k} \mathbf{n}_i w_{t_i}, \text{ where } w_{t_i} = \begin{cases} 0 & \text{if } t_i \text{ not part of } \mathsf{FG} \\ \text{area}(t_i) & \text{otherwise.} \end{cases} \tag{2.3}$$

With the per-FG normals, we compute per-triangle principal curvatures $\kappa_{\min}$ and $\kappa_{\max}$ using the method of Rusinkiewicz [7], as it is well-suited for GPUs [8]. Like Bénière et al. [9], we use principal curvature and mean curvature $H = \frac{1}{2}(\kappa_{\min} + \kappa_{\max})$ to classify triangles into:

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

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

conic if $|\kappa_{\max}| > H\varepsilon_{\text{cone}}^{h}$ and $|\kappa_{\min}| < H\varepsilon_{\text{cone}}^{l}$,

where s_{AABB} is the diagonal length of the axis aligned bounding box (AABB) of the input mesh. In contrast to Bénière et al. [9], we use two thresholds $\varepsilon_{\text{cone}}^{h}$ and $\varepsilon_{\text{cone}}^{l}$ for *conic* triangles. The threshold $\varepsilon_{\text{cone}}^{h}$ specifies the intended portion of $\kappa_{\max}$ to $\kappa_{\min}$ and $\varepsilon_{\text{cone}}^{l}$ controls how close $\kappa_{\min}$ should be to zero. We distinguish *spherical* and *conic* triangles into inverted and non-inverted depending on sign($\kappa_{\max}$).

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.

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. 2f). Therefore, we provide a region growing method for more complex meshes. Similar to Rendon-Cardona et al. [10], we fit the primitives plane, sphere, cone, and cylinder. In contrast to Rendon-Cardona et al. [10], 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

Proceedings of CAD'26, Athens, Greece, July 6-8, 2026, 32-38
© 2026 U-turn Press LLC, http://www.cad-conference.net

the closest curvature to the seed. 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:

$$\min_{t'} |\kappa_{\max}^{\text{seed}} - \kappa_{\max}^{t'}|(\mathbf{n}_{\text{seed}} \cdot \mathbf{n}_{t'}). \tag{2.4}$$

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.

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. To fit a plane to a FG, we compute the arithmetic average of vertices and the sum of normals weighted by triangle area. We accept the resulting plane for the FG, if maximum distance of points is below a certain threshold $\varepsilon_{\text{plane}}^{d}$ and $\frac{1}{N}\sum_{i=1}^{N}\theta_i < \cos(\varepsilon_{\text{plane}}^{n})$ holds for the triangle's gradients $\theta_i = \max(\mathbf{n}_i \cdot \mathbf{n}, \mathbf{n}_i \cdot (-\mathbf{n}))$. For the non-linear primitives, we use GPU-accelerated Gauss-Newton iterations [11] to find shape parameters $p_1, \ldots, p_n$ that minimize the squared distances:

$$\min_{p_1,\ldots,p_n} \sum_i d_{\text{prim}}^2(\mathbf{x}_i, p_1, \ldots, p_n), \text{ where } d_{\text{prim}} \text{ is the signed distance function of the primitive type.} \tag{2.5}$$

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. If no primitive type fits the FG, we perform the region growing based on primitive fitting within the FG.

After primitive fitting, we construct bounded cylinders and cones from capping planes [12]. In addition, we check if the fitted planes can be combined to a cuboid using a GA and ghost plane detection by Friedrich et al. [12] with unique hash mapping of planes to avoid duplicate cuboid evaluation. Sometimes additional primitives are needed to reconstruct the geometry, e.g., a fillet on a cuboid. Therefore, we add the planes of a primitive's oriented bounding box [3] 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. This reduces the search space for a GA.

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 [3]. 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 to rate the trees. To evaluate the fitness function, we uniformly sample the triangles of the partition and project the sampling points onto the primitive surface for robust evaluation [5]. We implement pre-processing of trees and early termination as proposed by Friedrich et al. [5]. 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.

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. If a node is only associated with one or two primitives, we perform a quick evaluation of the possible combinations, instead

© 2026 U-turn Press LLC, http://www.cad-conference.net

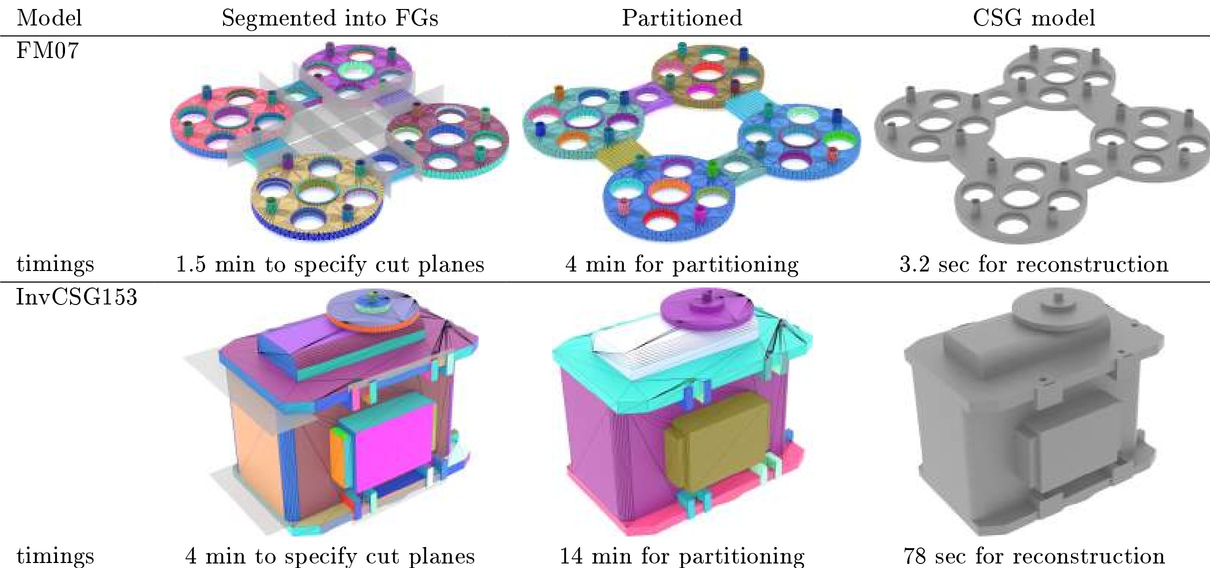

Fig. 3: Reconstruction of FM07 (top) and InvCSG153 (bottom) meshes. Each model is segmented into FGs (left) and partitioned by the user (center). Our method then reconstructs a CSG model (right).

of executing a GA. 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. We allocate the upper bound of arrays in GPU memory required for distances and gradients, which is the maximum tree level $\ell + 1$ due to the binary tree structure. The reconstruction for the partition terminates when the best tree achieves a sufficient fitness or a maximum number of iterations is performed. After reconstructing each partition, we convert the expression in Eq. (2.2) to the OpenSCAD format.

Results:

We implemented our method in C++ and CUDA and evaluated its performance on 50 meshes of varying complexity. Our evaluation compares the performance of our method with InverseCSG [4] and the per-partition reconstruction of Friedrich et al. [5]. We recorded reconstruction run times over 10 repetitions for each mesh on an Ubuntu 24.04 system with an AMD Ryzen 7 3700 (hyperthreading enabled) and an NVIDIA GTX 1650. Our method consistently provides the fastest median run times. On average, our method achieves 137× faster median run times than InverseCSG and 175× faster median run times than Friedrich et al. [5]. To evaluate the accuracy of the reconstruction, we compute the relative reconstruction error as the Hausdorff distance between the input mesh and the triangulation of the reconstructed CSG tree divided by the diagonal of the AABB of the mesh [4]. Our per-partition reconstruction achieves a 34× smaller mean error compared to InverseCSG and 14× smaller mean error compared to Friedrich et al.'s [5] method. The smaller errors of our method are primarily due to correctly detecting more primitives and more accurate primitive parametrization.

As our method involves user interaction, we demonstrate on the FM07 and InvCSG153 meshes that user-guided partitioning is beneficial for complex meshes (see Fig. 3). For the FM07, we reconstruct all partitions in 3.2 seconds, while InverseCSG requires 94 minutes and Friedrich et al. [5] require 125 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

© 2026 U-turn Press LLC, http://www.cad-conference.net

are significantly lower than the reconstruction times of InverseCSG and Friedrich et al. [5]. In terms of reconstruction error, our method provides a 140× lower error than InverseCSG, while the method of Friedrich et al. [5] provides the same result as our method. For the InvCSG153, we reconstruct most features correctly in 78 seconds, while InverseCSG requires 172 minutes to provide a result of 2.4× lower reconstruction error compared to our method. The method of Friedrich et al. [5] requires 32 minutes for the InvCSG153 mesh but misses many features leading to an 18× 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 times of InverseCSG and Friedrich et al. [5].

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.

Conclusions:

We have presented a user-guided method for CSG tree recovery that provides fast run time performance and accurate results. In conjunction with proper parallelization, the reduced combinatorial complexity leads to a speedup of up to two orders of magnitude. The reduced search space of our per-partition reconstruction facilitates finding a correct CSG tree, which leads to an order of magnitude lower mean error. As our method provides substantial improvements in terms of run times and accuracy, it is worthwhile to invest a few minutes for the interactive partitioning step. A limitation of our method is that it is not resistant to noise and cannot be applied to scanned point clouds. In future research, our method could be extended to automatic partitioning to reduce user labor.

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

References:

[1] Fayolle, P.-A.; Friedrich, M.: A Survey of Methods for Converting Unstructured Data to CSG Models, Computer-Aided Design, 2024, 168, https://doi.org/10.1016/j.cad.2023.103655

[2] Xu, X.; Peng, W.; Cheng, C.; Willis, K.; Ritchie, D.: Inferring CAD Modeling Sequences Using Zone Graphs, Proc. CVPR, 2021, 6062-6070, https://doi.org/10.1109/cvpr46437.2021.00600

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

[4] Du, T.; Inala, J.; 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), 2018, https://doi.org/10.1145/3272127.3275006

[5] Friedrich, M.; Fayolle, P.-A., Gabor, T.; Linnhoff-Popien, C.: Optimizing evolutionary CSG tree extraction, Proc. GECCO'19, 2019, 1183-1191, https://doi.org/10.1145/3321707.3321771

© 2026 U-turn Press LLC, http://www.cad-conference.net

[6] Jiao, X.: Volume and Feature Preservation in Surface Mesh Optimization, Proc. IMR, 2006, 359-373, https://doi.org/10.1007/978-3-540-34958-7_21

[7] Rusinkiewicz, S.: Estimating curvatures and their derivatives on triangle meshes, Proc. 3DPVT'04, 2004, 486-493, https://doi.org/10.1109/TDPVT.2004.1335277

[8] Griffin, W.; Wang, Y.; Berrios, D.; Olano, M.; GPU curvature estimation on deformable meshes, Proc. I3D'11, 2011, 159-166, https://doi.org/10.1145/1944745.1944772

[9] 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), 2013, 1382-1393, https://doi.org/10.1016/j.cad.2013.06.004

[10] Rendon-Cardona, C.; Correa, J.; Acosta, D.; Ruiz-Salguero, O.: Analytic Form Fitting in Poor Triangular Meshes, Algorithms, 14(11), 2021, https://doi.org/10.3390/a14110304

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

[12] Friedrich, M.; Illium, S.; Fayolle, P.-A.; Linnhoff-Popien, C.: A Hybrid Approach for Segmenting and Fitting Solid Primitives to 3D Point Clouds, Proc. VISIGRAPP, 2020, https://doi.org/10.5220/0008870600380048

© 2026 U-turn Press LLC, http://www.cad-conference.net