Title:
Robust Segmentation of CAD-generated STL files

Authors:
Tathagata Chakraborty, tathagata_chakr@hcl-software.com, HCLTech
Manoj Bhonge, manoj.bhonge@hcl-software.com, HCLTech
Abhijit Kumthekar, abhijit.kumthemar@hcl-software.com, HCLTech
Nitin Umap, nitin.umap@hcl-software.com, HCLTech

Keywords:
STL, Mesh Segmentation

DOI: 10.14733/cadconfP.2025.17-23

Introduction:

Since they were first introduced in 1987, STL files have remained the most popular 3D file format for 3D data exchange. The use of STL files has increased in recent years with the popularity of 3D printing where it is the file format of choice [10, 11]. The STL file format is also very simple, which makes it easy to read from and write to, thus enabling widespread support across all CAD/CAM/CAE products.

STL files are also popular in Computer Aided Manufacturing (CAM) [15] (primarily due to security concerns), where a majority of parts are still manufactured using two and three-axis machining techniques [1], and where the information required for manufacturing can be obtained from STL files. Manufacturing is often outsourced to manufacturing shops, that may not have the same level of security [17]. It is of competitive importance to protect the original parametric part designs [8], and STL files, which encode in them the very minimal amount of 3D geometry required for manufacturing, are often a preferred format for sharing this information, despite several limitations [5].

CAM applications can, however, significantly benefit if the STL file is face-segmented [2]. An accurately face-segmented model allows the CAM programmer to quickly select regions in a part for selective processing [3]. For example, the user may want to generate different toolpaths and operations for different pockets, holes, and other features, which would require the user to select all the triangles comprising these features. However, a particular feature geometry may be defined by thousands of triangles, and selecting these manually can be tedious.

Reverse Engineering and Mesh Segmentation:

There are various reasons why one may want to reverse engineer STL files [7, 12], but this process is very difficult to automate when accurate segmentation is required [9]. Traditional methods for mesh segmenation include region growing, hierarchical clustering, and spectral clustering, each with its own strengths and limitations [16, 2]. Region growing methods, begin with seed components and iteratively expand clusters based on geometric features, but they can be sensitive to seed selection and may struggle with noisy data [13]. Spectral clustering approaches utilize the eigenvalues of the Laplace-Beltrami operator to partition meshes, offering robustness to topological variations. However, they can be computationally intensive [4]. With the advent of deep learning, supervised methods have emerged, such as Graph Convo-

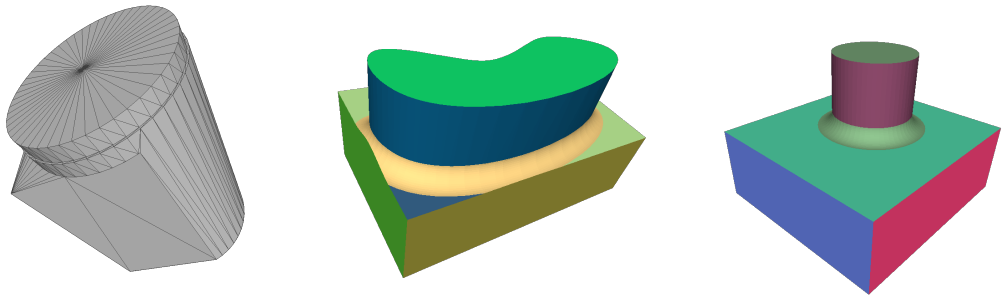

Fig. 1: Left: Typical tessellation of a CAD-generated STL. Middle and Right: Segmentation results of relatively simple STLs.

lutional Neural Networks (GCNNs), which treat meshes as graphs and learn to segment them effectively. These methods can capture complex patterns but require substantial labeled data for training [14].

In traditional mesh segmentation methods and their applications high-accuracy is not required or expected. However, with CAD-generated STL files, for use in downstream applications like CAM, the expectation is that the faces should be accurately segmented with perfect boundaries. Popular approaches like region growing and clustering work well on meshes obtained from a diverse range of domains but are not very accurate on CAD-generated STL files where is tessellation is highly non-uniform. In this paper we describe a method for robustly segmenting STL files generated from parametric CAD applications. We use an often overlooked property of such files, namely, that here the facets comprising a face are stored contiguously. Leveraging this we describe an algorithm that can incorporate multiple edge identification rules for robust segmentation results. We also briefly describe a visual correction and inspection tool that can help correct any segmentation issues.

Robust Segmentation of CAD-generated STL:

In general, in STL files the triangles are stored in arbitrary order. When STL files are generated from non-parametric CAD applications, the order of triangles in the STL file is unpredictable and depends on the algorithm used. STLs generated from parametric CAD applications, on the other hand, are triangulated face-by-face and have a certain tessellation aesthetic that is readily apparent (see Fig. 1 (left)).

Due to the face-by-face triangulation, the tesselation of each face is stored in a contiguous section but the faces themselves may follow in an arbitrary order. For example, if δ_i represents the triangles comprising the i^{th} face in the CAD file (say, comprising of n faces), then the triangles in the STL will be stored as (δ_1, δ_2, δ_3, $\cdots$, δ_n). One way, therefore, to segment the faces would be to identify the boundaries between a pair of neighboring contiguous sections (that is, between say δ_i and δ_{i+1}), that is to determine in the list of triangles in an STL file where one face ends and the next face starts.

This significantly simplifies the face segmentation problem by constraining the result. For example, a segmentation where the triangles are not contiguous is wrong. Additionally, in the segmentation each region must have a connected outer loop and zero or more connected inner loops. If these loops are not complete, then the segmentation is likely still incomplete. These constraints are inviolable in CAD-generated STLs. In the next section we explain how these constraints can be used to develop a segmentation algorithm.

Segmentation Algorithm Overview:

Our method needs at least one heuristic for identifying edges at face boundaries. These heuristics should

be conservative and should identify a separating edge with a very high probability. Conservative rules lead to better segmentation, whereas, less strict rules result in too many faces. Conservative rules can sometimes club together several smoothly connected faces (typially fillets and chamfers). However, in most cases, the constraints of the structure of the file often prevent such clubbing. When some faces are clubbed together and recognized as a single face, a visual correction and inspection tool can be used to separate the faces easily. Fig. 1 shows two examples of segmentation on simple meshes.

Often just one conservative rule is sufficient to segment most CAD-generated STL files. This rule uses the sharpness of the edge to identify a face boundary with high accuracy. This single rule works well for prismatic CAD models that do not contain fillets or chamfers. That said, even in the presence of fillets and chamfers, where the face boundaries cannot be detected, our algorithm often segments the face correctly, as long as these faces are bounded on each side by faces that were sucessfully detected.

Overall Algorithm:

Assume that we have a function *FindFace* that can detect the end of a face in the list of triangles in the STL file. This function takes a start index of the face and returns a pair of values, the first of which indicates the end index of the last triangle in the face and the second boolean value that indicates whether the search was successful. We will describe this function in the next section later. The overall algorithm *FindAllFaces* simply calls the *FindFace* function repeatedly till all the facets are processed. The function *FindAllFaces* returns a list of indices indicating where each face ends.

Algorithm 1: Overall Algorithm

Input: $body$ `// STL body with connectivity information`
$F = [f_1, f_2, \cdots, f_n]$ `// face edge detection functions`

Function `FindAllFaces(`*body, F*`):`
- $N \leftarrow NumTriangles(body)$
- $idx_c \leftarrow 0$ `// current start index`
- $I \leftarrow []$ `// list of face end indices`
- **while** $idx_c < N - 1$ **do**
  - $(idx_f, valid) \leftarrow FindFace(body, F, idx_c)$
  - **if** $valid$ **then**
    - **if** $failed_{last}$ **then**
      - $I \leftarrow I \cup idx_c - 1$
      - $failed_{last} \leftarrow false$
    - **end**
    - $I \leftarrow I \cup idx_f$
    - $idx_c \leftarrow idx_f + 1$
  - **end**
  - **else**
    - $idx_c \leftarrow idx_c + 1$
    - $failed_{last} \leftarrow true$
  - **end**
- **end**
- **return** I

This algorithm is shown below. Some implementation details have been omitted for legibility. It is also assumed that the connectivity of the triangles has been computed and stored in a half-edge data structure

since this is required by the *FindFace* function. The algorithm must additionally include data-structures that map a facet to a segmented face and query edges already identified as boundary edges. Properly caching this information can lead to significant improvements in the performance of the algorithm.

The basic idea is to start with the current index idx_c set to 0 and find the end index of the first face idx_f. If the *FindFace* function is successful then we add the end index idx_f to the list I, update the start index idx_c to idx_f+1. If on the other hand the *FindFace* function fails, then we only increment the current index idx_c by 1 and call the *FindFace* function again. When all triangles have been processed, we return the list I from the function, that now contains the end indices of the identified faces.

Face Segmentation Algorithm:

The *FindFace* function uses a half-edge data structure encoding the connectivity of the triangles in the STL [6]. The *FindFace* function is shown below.

Algorithm 2: FindFace Algorithm

Input: *body* `// STL body with connectivity information`
$F = [f_1, f_2, \cdots, f_n]$ `// face edge detection functions`

Function `FindFace`(*body*, F, idx_s)**:**
- $N \leftarrow NumTriangles(body)$
- $idx_c \leftarrow idx_s$ `// current start index`
- $LC_l \leftarrow []$ `// list of face boundary coedge lists`
- **while** $idx_c < N - 1$ **do**
  - $MergeCoedgeLists(LC_l)$
  - **if** $size(LC_l) > 0 \wedge AreLoopsComplete(LC_l)$ **then**
    - **return** $(idx_c - 1, true)$
  - **end**
  - $f \leftarrow GetFacet(body, idx_c)$
  - $C_l \leftarrow FindHardEdgesInFacet(f, F)$
  - $LC_l \leftarrow LC_l \cup C_l$
  - $idx_c \leftarrow idx_c + 1$
- **end**
- $MergeCoedgeLists(LC_l)$
- **if** $size(LC_l) > 0 \wedge AreLoopsComplete(LC_l)$ **then**
  - **return** $(idx_c - 1, true)$
- **end**
- **return** $(idx_s, false)$

The *FindFace* function maintains a list of coedge lists LC_l. The coedges stored in these lists are those identified by at least one of the edge detection functions F or those marked as face boundary from previously detected faces. The function starts from the input index idx_s and gets the triangle corresponding to that index. The *FindHardEdgesInFacet* function tries to identify any face boundary edges in this facet using the functions F and from previously identified face boundaries. If any such edges are detected then their corresponding coedges are appended to the list of list CL_l. If possible the coedge is appended to a coedge list to which it is connected. Otherwise a new list is created and the coedges added to the new list.

Before analyzing the next facet, a function called *MergeCoedgeLists* is called that tries to identify and merge the separate lists in CL_l if they can be connected. Additionally we check if all the coedge

lists in the CL_l list form loops. If all the coedge lists form loops after merging, then we have successfully identified a face and return the last index that was analyzed. The implementations of the functions *MergeCoedgeLists*, *FindHardEdgesInFacet* and *AreLoopsComplete* have been left out.

The *FindFace* algorithm as shown in Alg. 2 can be slow when it fails to detect a face as, for example, when it fails to detect a face boundary edge and thus fails to close a coedge loop. In such cases it will try checking all triangles till the end of the file but will still fail in every case. To improve performance the *FindHardEdgesInFacet* function can return the index of the facet neighboring a hard edge when a hard edge is found. This index can be used to limit the value of the index till which the algorithm checks for a valid face.

Visual Inspection and Correction:

With an appropriate set of face boundary detection functions F, many faces in a typical STL file can be successfully detected. However in some cases two or more faces which are smoothly connected and occur next to each other in the STL can get clubbed (this typically happens in the case of fillet or chamfer chains). Manual user intervention is then required to correct this. Again, the contiguous nature of the face facets can be leveraged to correct such cases by providing visual feedback to the user.

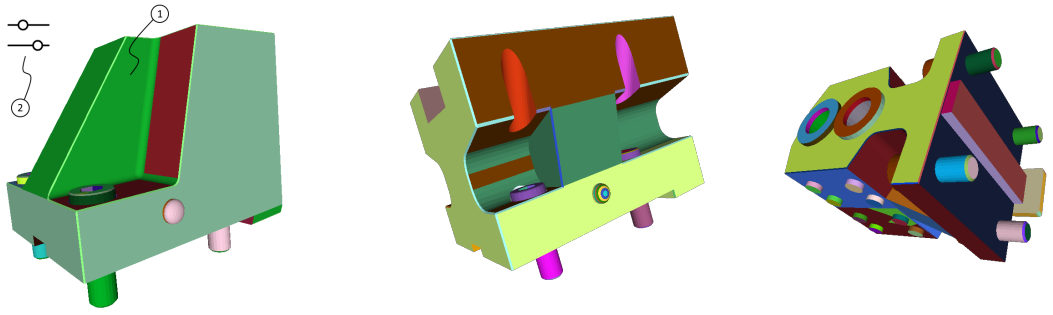

Fig. 2: Illustrating a segmentation result where fillets have been incorrectly grouped and two more segmentation results on medium complexity models.

The user first opens the STL file in a visualization application where the faces of the segmented STL file are color coded randomly. The user next identifies by visual inspection any cases where two or more faces have been inaccurately merged together (refer to Fig. 2). In such cases the user selects a random triangle in the merged face (1 in Fig. 2) and uses two sliders (2 in Fig. 2) to extend the selection to the actual face. Once such selection is done the user clicks a button to mark this as a face. Alternately, in order to further speed up the process of selection, softer versions of the face boundary detection functions maybe used to extend the user selection to the nearest face boundaries.

Limitations and Conclusions:

Our method does not work with STL files generated from non-parametric CAD applications, for example, 3D sculpting applications, and reverse engineering applications that generate STL files from point clouds. In engineering domains though, the STL files are typically generated from parametric CAD applications, and here the facets for each parametric face in the model are always contiguous in the STL file. Our method is designed for only such STL files.

Our method may also sometimes club neighboring faces when they are smoothly connected (for example, in the case of fillet chains, where the fillets are next to each other in the STL file, and it is very difficult to identify the face edges). This may then require manual correction in regions where a

perfect segmentation results is expected. That said, the visual inspection and correction tool described here can significantly reduce the tediousness involved in correcting these segmentation issues.

Acknowledgement:
This research was conducted as a part of the research and development efforts of the CAMWorks team in HCLTech. We are thankful to the CAMWorks team, and especially to Baburaj Iyer and Vivek Govekar for allowing us the time to undertake this research.

Tathagata Chakraborty, https://orcid.org/0000-0002-2752-2533
Manoj Bhonge, https://orcid.org/0000-0001-7638-4600
Abhijit Kumthekar, https://orcid.org/0000-0001-9362-7141
Nitin Umap, https://orcid.org/0000-0002-9063-1230

References:

[1] Abdullah, H.; Ramli, R.; Wahab, D. A.: Tool path length optimisation of contour parallel milling based on modified ant colony optimisation, The International Journal of Advanced Manufacturing Technology, 92, 2017, 1263-76. https://doi.org/10.1007/s00170-017-0193-5

[2] Agathos, A.; Pratikakis, I.; Perantonis, S.; Sapidis, N.; Azariadis, P.: 3D mesh segmentation methodologies for CAD applications, Computer-Aided Design and Applications, 4(6), 2007, 827-41. https://doi.org/10.1080/16864360.2007.10738515

[3] Al-wswasi, M.; Ivanov, A.: A novel and smart interactive feature recognition system for rotational parts using a STEP file, The International Journal of Advanced Manufacturing Technology, 104, 2019, 261-84. https://doi.org/10.1007/s00170-019-03849-1

[4] Bao, X.; Tong, W.; Chen, F.: A Spectral Segmentation Method for Large Meshes, Communications in Mathematics and Statistics, 2023 Sep, 11(3), 583-607. https://doi.org/10.1007/s40304-021-00265-4

[5] Bonnard, R.; Hascoët, J. Y.; Mognol, P.: Data model for additive manufacturing digital thread: State of the art and perspectives, International Journal of Computer Integrated Manufacturing, 32(12), 2019, 1170-91. https://doi.org/10.1080/0951192x.2019.1690681

[6] Brönnimann, H.: Designing and implementing a general purpose halfedge data structure, In Algorithm Engineering: 5th International Workshop, WAE 2001 Århus, Denmark, August 28-31, 2001 Proceedings, 5, 2001, 51-66. https://doi.org/10.1007/3-540-44688-5_5

[7] Buonamici, F.; Carfagni, M.; Furferi, R.; Governi, L.; Lapini, A.; Volpe, Y.: Reverse engineering of mechanical parts: A template-based approach, Journal of computational design and engineering, 5(2), 2018, 145-59. https://doi.org/10.1016/j.jcde.2017.11.009

[8] Chaduvula, S. C.; Dachowicz, A.; Atallah, M. J.; Panchal, J. H.: Security in cyber-enabled design and manufacturing: A survey, Journal of Computing and Information Science in Engineering, 18(4), 2018. https://doi.org/10.1115/1.4040341

[9] Hao, J.; Fang, L.; Williams, R. E.: An efficient curvature-based partitioning of large-scale STL models, Rapid Prototyping Journal, 17(2), 2011, 116-27. https://doi.org/10.1108/13552541111113862

[10] Iancu, C.; Iancu, D.; Stăncioiu, A.: From CAD model to 3D print via "STL" file format, Fiability & Durability/Fiabilitate si Durabilitate, (1), 2010.

[11] Kumar, V; Dutta D.: An assessment of data formats for layered manufacturing, Advances in Engineering Software, 28(3), 1997, 151-64. https://doi.org/10.1016/s0965-9978(96)00050-6

[12] Kumar, A.; Jain, P. K.; Pathak, P. M.: Industrial application of point cloud/STL

data for reverse engineering, DAAAM International Scientific Book, 2012, 445-62. https://doi.org/10.2507/daaam.scibook.2012.38

[13] Liang, C.; Yin, J.; Wu, J.; Wang, J.; Wei, M.; Guo, Y.: A survey of 3D mesh segmentation based on clustering analysis, Journal of Computer-Aided Design & Computer Graphics, 2020 Apr 20, 32(4), 680-92. https://doi.org/10.3724/SP.J.1089.2020.17976

[14] Perek, E, K: Supervised mesh segmentation for 3d objects with graph convolutional neural networks, Master's thesis, Middle East Technical University (Turkey).

[15] Qu, X.; Stucker, B.: Raster milling tool-path generation from STL files. Rapid Prototyping Journal, 12(1), 2006, 4-11. https://doi.org/10.1108/13552540610637219

[16] Shamir, A.: A survey on mesh segmentation techniques, InComputer graphics forum, Oxford, UK: Blackwell Publishing Ltd., 27(6), 2008, 1539-1556. https://doi.org/10.1111/j.1467-8659.2007.01103.x

[17] Tuptuk, N.; Hailes, S.: Security of smart manufacturing systems, Journal of manufacturing systems, 47, 2018, 93-106. https://doi.org/10.1016/j.jmsy.2018.04.007