Pen-Based User Interface for Geometric Decomposition for Hexahedral Mesh Generation

Jean Hsiang-Chun Lu[1], Inho Song[1], William Roshan Quadros[2], and Kenji Shimada[1]

[1] Carnegie Mellon University, Pittsburgh, PA, hsiangcl@andrew.cmu.edu, {songphd, shimada}@cmu.edu
[2] Sandia National Laboratories[†], Albuquerque, NM, wrquadr@sandia.gov

Summary. In this work, we present a pen-based user interface (UI) that makes manual geometry decomposition easier and helps reduce user time spent on geometry decomposition for hexahedral meshing. This paper presents the first attempt to apply a pen-based UI for geometry preparation for mesh generation. The proposed UI is more natural, intuitive, and easier to use than existing interfaces such as Window-Icon-Mouse-Pointer and command-line interfaces. The pen-based UI contains the following features: (1) drawing freeform strokes, (2) fitting and aligning strokes to CAD geometry automatically, (3) creating cutting surfaces, and (4) performing various tasks including webcutting through gestures. The proposed pen-based UI has been tested on a few models to demonstrate its effectiveness in decomposition, defeaturing and controlling mesh orientation.

Keywords: pen-based interface, decomposition, hexahedral meshing

1 Introduction

Manual geometry decomposition is one of the most time-consuming steps in the mesh generation process. As shown in Fig. 1, geometry decomposition dominates time spent during hex-meshing tasks at Sandia National Lab. In order to perform geometry decomposition in existing preprocessing / mesh-generation tools (e.g. CUBIT, TurboMesh), users have to either select an appropriate command via a Window-Icon-Mouse-Pointer (WIMP) Graphical

[†]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000

User Interface (GUI), or type the command in the command-line interface (CLI). Though these traditional interfaces are powerful, they are cumbersome to a novice user.

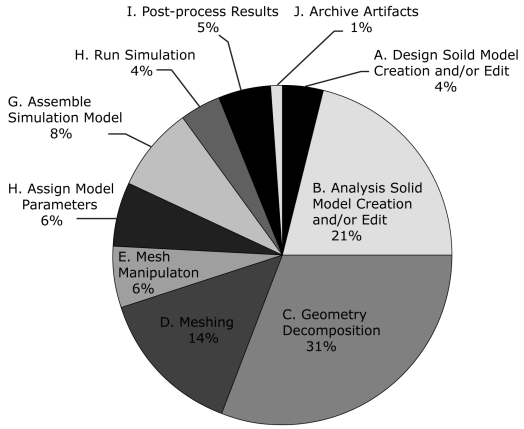


Fig. 1: The user time spent for various tasks in mesh generation process at Sandia National Laboratories [1].

The decomposition of models into meshable geometries has received a great deal of attention due to the limitations of current hexahedral meshing algorithms. The practical approach is to manually decompose the complex geometry into meshable regions. These meshable regions are then imprinted and merged before performing surface and volume meshing. The surfaces of these regions are meshed using existing algorithms such as paving, mapping, and sub-map [2]. Then volume meshing schemes such as sweeping (which requires specifying source and target surfaces) and mapping are used to mesh the sub-domains.

The pen-based UI proposed in this paper improves the interface of decomposition operation in order to make the manual decomposition process more efficient. A recent trend in computer-aided engineering software is moving toward more natural and accessible interfaces, of which pen-based interfaces are a prime example. In the early design stage, pen-based interfaces that use a stylus pen and tablet have already been playing an important role and have been used in freeform styling design. However, deposit its potential, pen-based UI has not been commonly utilized in the engineering analysis. In this paper, the benefits of pen-based interfaces are brought to engineering analysis tasks by developing a new UI for easier and faster manual geometry decomposition for hexahedral meshing as shown in Fig. 2.

This work is the first attempt to apply pen-based interface in mesh generation and is in its early development stage. More pen-based functions will

be implemented in the future. The overview of the proposed UI is as follows. The program first takes freehand input as sequences of screen coordinates. The freehand inputs are then re-sampled and smoothed. The processed input point set are then identified as one of the three types: lines, circles, or splines. The program then fits appropriate 1D geometry entity to each input according to the type. Cutting surfaces are created by sweeping the 1D geometry entity in a given sweeping direction. The solid model is then decomposed using the cutting surfaces.

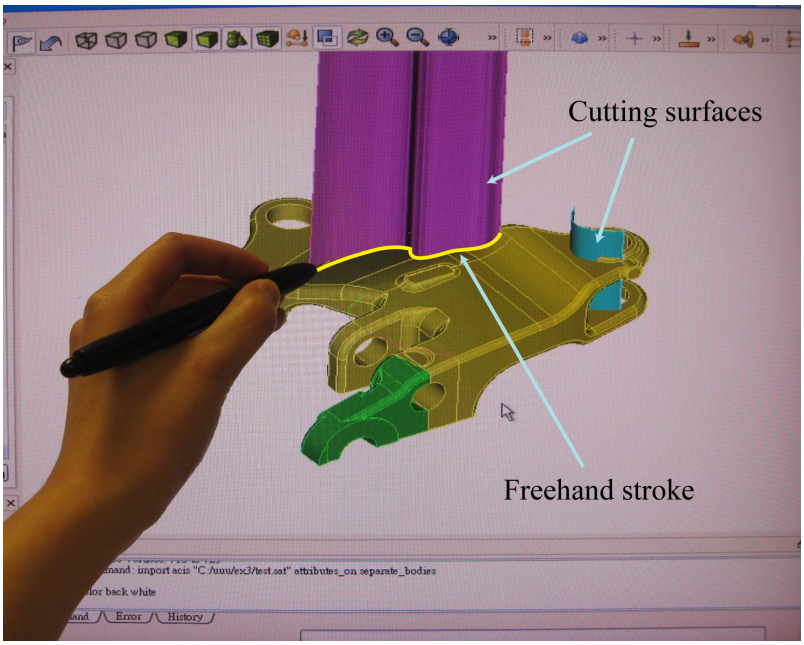


Fig. 2: Decomposing a model with the proposed pen-based UI.

The rest of the paper is organized as follows: Section 2 discusses the previous related works. Section 3 covers the beautification and snapping algorithms to process the freehand inputs. In Section 4, we describe how to create cutting surfaces for webcutting [3]. Section 5 includes results that illustrate decomposition targeting better mesh quality, controlling orientation of mesh and defeaturing for hex meshing.

2 Related Work

2.1 MARK-IT

The approach of Yang et al. to improve user interface (UI) with MARK-IT [4] successfully simplifies the WIMP GUI as shown in Fig. 3 and saves user time by avoiding switching among different panels in a conventional GUI interface. Decomposition commands are executed by selecting entities and drawing a corresponding mark with the mouse. MARK-IT covers over 50 CUBIT operations with different marks. To help users in drawing a correct mark, a label appears by the cursor to display the associated command. When a user needs to input a value, MARK-IT provides a "virtual slider" that can be dragged to adjust the value.

MARK-IT shows the potential of a marking interface that is integrated with a complex meshing system such as CUBIT. MARK-IT avoids unnecessary movement of the mouse by improving access to commands and operations. However, it does not reduce the time spent in the manual decomposition process.

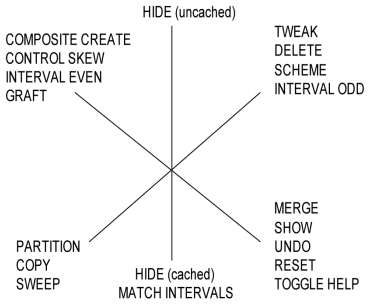


Fig. 3: Commands shown in MARK-IT interface [4].

2.2 ITEM

CUBIT offers the Immersive Topology Environment for Meshing (ITEM) [5], a wizard-like environment that uses the approach used by [6] to determine the possible cutting surfaces. First, all the curves that form a dihedral angle less than an input value (default is 135 degrees) were collected. Second, a graph-based algorithm was used to determine all the closed loops or the open loops that can be closed using a boundary edge. Third, all the surfaces that bound the loops were collected. Fourth, extensions of the collected surfaces were used in defining the cutting surfaces.

The extensions of the surfaces are presented as suggest cutting surfaces to users. Fig. 4 shows some of the cutting surfaces suggested by ITEM. However, those suggested cutting surfaces might not result in a meshable model.

The user has to search through the available options and make meaningful selections if exists. If there are no meaningful cutting surfaces, then the user has to use CLI or GUI to manually create the desired cutting surfaces. Thus, time-consuming manual decomposition operations are still required.

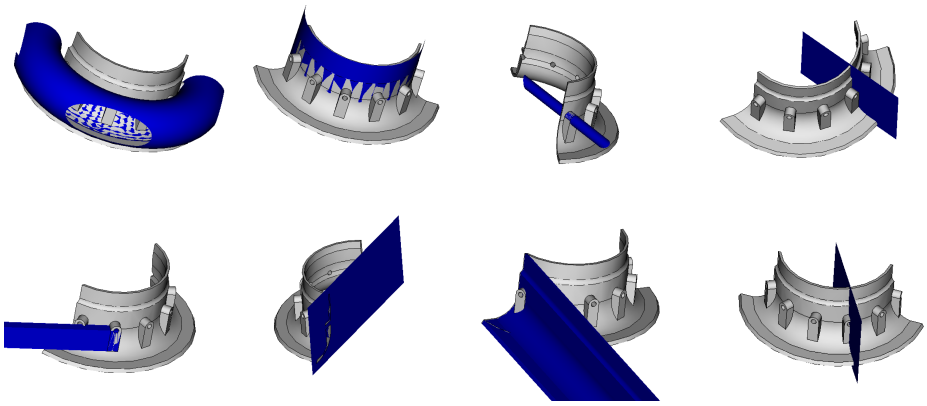


Fig. 4: Suggested cutting locations provided by ITEM.

2.3 Pen-Based Interface

Numerous researches have focused on using pen-based interface to create 3D models. Free-form surface generation or model creation systems such as [7, 8, 9] reconstruct a 3D model or surface from 2D sketch inputs. The template-based system proposed by Yang et al. [10] applies sketch recognition algorithms to match points and curves to the given 2D templates. Kara and Shimada [11] use computer vision techniques to align a template with an input sketch for users to deform a model. By tracing over the sketch, user can simply draw the new shapes on the model for deformation. Kho et al. [12] use strokes to define a free-form skeleton and the region of the model to be deformed. Nealen et al. [13] first detect an object's silhouette, and then let the user create new features or modify the existing one by sketching an approximate shape onto the model. Cheutet et al. [14] allow the user to directly operate on an object by modifying strokes with a stylus pen. The gesture based interface used by Teddy [7] and [15] uses user strokes for operations such as extension, bending, and other modification functions. An optimization-based algorithm is used to determine the depth of 2D sketch vertices in the work presented by Masry et al. [16]. It constructs a 3D wireframe shape from a 2D freehand input for a physical simulation.

Though pen-based interfaces have many contributions in the CAD modeling field, their advantages have not been applied in geometry preprocessing for hex meshing.

3 Drawing, Beautification and Snapping of Freehand Inputs

The proposed UI interprets the user's freehand input as a "hint" in order to guess the intention and suggests the best matching result. It solves the problems of inaccuracy and repeatability of freehand input strokes by beautifying them with re-sampling, smoothing filters and geometric fitting to 1D entities, and then snapping the 1D entities by offsetting and overlapping algorithms. The user draws a freehand input P, then the input is re-sampled and smoothed to $\tilde{P}$. $\tilde{P}$ is represented as a 1D geometric entity $C(t)$ such as a line, a circle, or a spline. In the future implementation, the system will search on the object surface for edges of the same geometry type and calculate the distance between the matching edge and $C(t)$. $C(t)$ will then be snapped as an offset or overlap of the closest edge.

Another feature of the proposed UI is using gestures for easily access commands. Currently, ten gestures can be recognized by the systems. The user can draw gestures to execute the linked commands.

3.1 Drawing on Object Surfaces

The users intentions are either to create 1) a geometric entity, or 2) to specify an operator that works on the geometry entity. In the case of geometric entities, the intensions are to create 1D entity of one degree (e.g. line, polyline), two degrees(e.g. arc, circle, ellipse, parabola), or higher degrees(e.g. cubic splines, Bezier, NURBS). However, currently only lines, arcs, circles, and splines can be identified. The intended operators generally work on the 1D entities or can operate on 2D entities (e.g. cutting surface). The operators that work on 1D entities include alignment, trimming, offsetting, extending, sweeping. The operators that work on 2D entities include webcutting using a cutting surface. These operators are specified via gestures as shown in Fig. 5 (the dots on the gestures indicate the starting points) or key shortcuts. As shown in Fig. 6, the gesture can still be recognized even the directions and sizes are different from the template. Currently, a subset of operators are implemented and linked with the gestures, and rest of the gestures will be linked in the future with other advanced operators.

3.2 Beautification: Resampling, Smoothing and Geometric Fitting

The proposed interface uses three steps to beautify freehand inputs: 1) resampling, 2) smoothing, and 3) geometric fitting. Resampling will uniformly

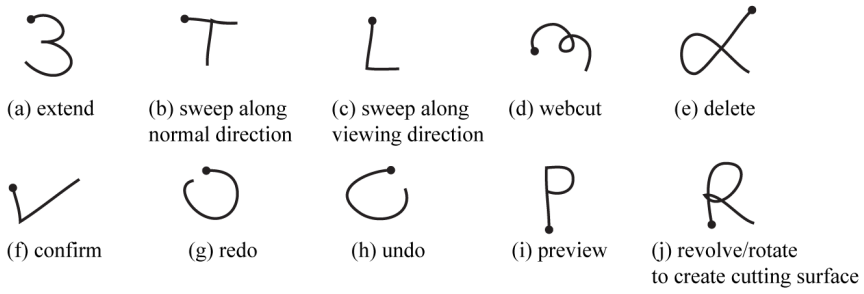


Fig. 5: Gesture set.

Fig. 6: An example of alpha gesture recognition. The program is able to recognize the incurrate alpha gestures correctly though they are in different sizes and drawing directions.

distribute the points carried by the freehand input. Smoothing removes noise while geometric fitting matches the freehand input to an intended shape such as a line, circle, and spline.

A freehand input is noisy due to the hand control of a user as well as sampling errors caused by stylus and tablet. The number of samples in a given input stroke depends on the pen speed (i.e. point density is inversely proportional to the users pen speed) and varies among devices. The system needs to re-sample the input to obtain data points uniformly spaced along the stroke before Laplace smoothing [17] can be applied to remove noise from the inputs. The noisy freehand input is replaced with a re-sampled and smoothed stroke, shown in Fig.7(a) and (b), respectively. The third step is to fit the

(a) Before smoothing. (b) After smoothing.

Fig. 7: Smoothing the freehand input.

smoothed inputs to common predefined shapes, such as line segments (Fig. 8(a)), circles (Fig. 8(b)), and splines (Fig. 8(c)), if they match. We define a set of tolerances to determine how a stroke should fit to each predefined shape. This tolerance is defined as the ratio of the length between the endpoints of the stroke, L_{length} and the actual curve length of the stroke L_o. If the tolerance is greater than 0.99, then the stroke is fit to a line segment. If the tolerance is less than 0.06, then the stroke is fit to a closed circle. We assume that when the user would like the freehand input be fit to a circle with radius r, the length of the input L_o is close to $2\pi r$ and L_{length} is within 0 to $r/3$ as shown in Fig. 9. Otherwise, the stroke is returned as a spline as shown in Fig. 8(c).

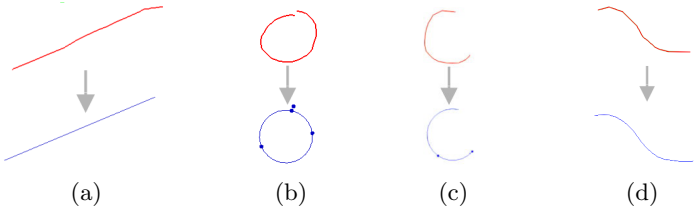


Fig. 8: Freehand inputs are fitted to (a) a line segment; (b) a circle; (c) an arc; (d) a spline.

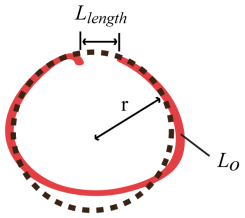


Fig. 9: Snap a freehand input to a circle. Thick solid line represents a freehand input, which will be fitted to a circle shown with a dash line.

3.3 Identifying Candidate Edges and Snapping

In this section, we describe how to snap 1D input entities to fit the object edges of the same type. It is important to identify the 1D entities types before snapping to reduce the search space. In order to snap the 1D entities correctly to the closet edges on the object surfaces, we first identify the 1D entities as different types: lines, circles and splines, then the system can search on the

object surfaces for candidate edges which have the same type. Next the system compares the average distances between different candidate edges and the 1D entity. If the average is within the tolerance T_L, the edge will be selected as a candidate. The algorithm for identification is shown in Fig. 10.

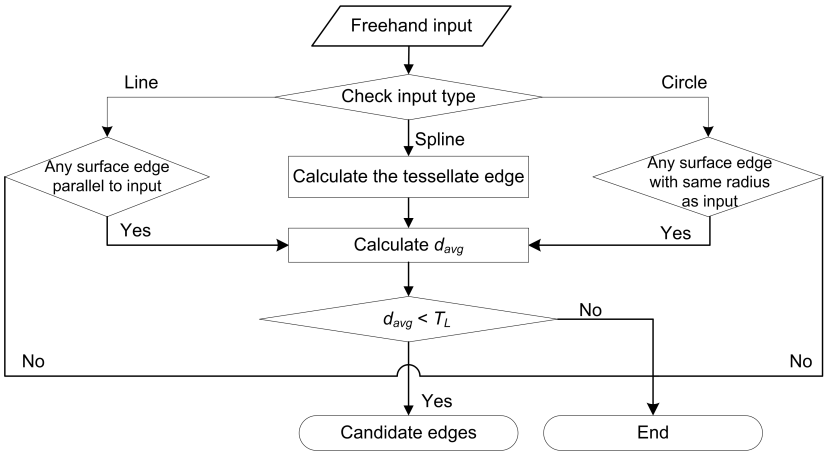


Fig. 10: The algorithm to identify candidate edges.

To snap a 1D entities to edges on object surfaces, we apply the method proposed by Song and Shimada [18] as shown in Fig. 11. After the system identify the candidate edges of the same type as the 1D entity, it calculates the average distance d_{avg} shown in Fig. 12(a), between the 1D entities and those edges. The d_{avg} is then compared with the overlapping tolerance T_o and parallel tolerance T_p. When d_{avg} is less than the overlap tolerance T_o, the 1D entity will be trimmed to its closest edge length. When d_{avg} is within T_o and T_p, the 1D entity will be offset d_{avg} from the edge displayed in Fig. 12(b) as the result shown in Fig. 12(c). The offsetting result to the circle edge is shown in Fig. 13. If d_{avg} is greater than T_p, the 1D entities will not be offset to the edge.

4 Creating Cutting Surfaces and Decomposition

The user can sweep a beautified and snapped 1D entity to create a cutting surface. The system first picks a surface of the given model by the midpoint of the entity then projects the stroke on the surface. The entity will be swept to create a cutting surface in one of the following directions: 1) the object's surface normal direction; or 2) the current viewing direction. More sweeping directions will be implemented in the future.

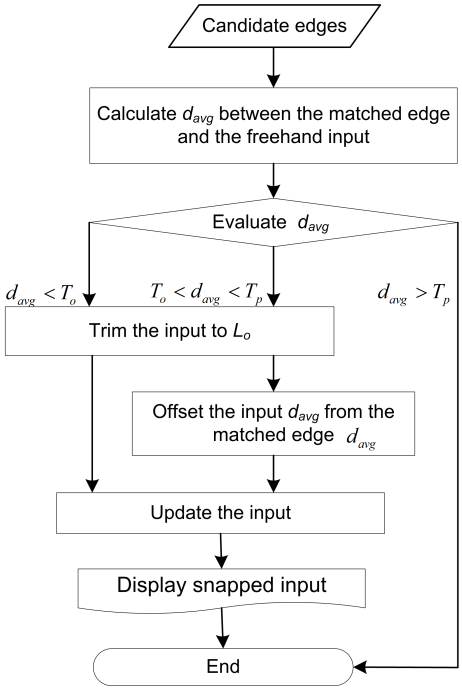
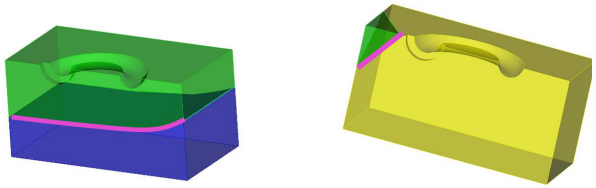


Fig. 11: The snapping algorithm if candidate edges exist.

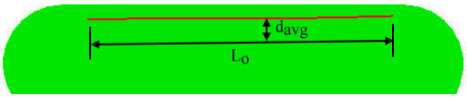
Next, the system determines the target volume with the midpoint of the stroke. Then the stroke is swept along the sweeping direction until it reaches the bounding box size of the target volume. Fig. 14(a) and (b) show the cutting surface and the decomposition along the surface normal direction and the viewing direction, respectively.



(a) Sweeping along normal direction

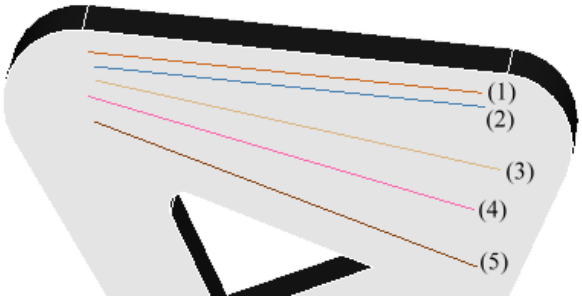
(b) Sweeping along current viewing direction

Fig. 14: Creating cutting surfaces for webcutting.



(a) The original input stroke

(b) The snapped stroke



(c) The distances from strokes (1) and (2) to the upper edge are smaller than T_P so they are offset to the upper edge. Stroke (3) is not offset to any edges. The distance between (4) and the lower edge is greater than T_P thus (5) is not offsett to the lower edge. (6) is offsett to the lower edge.

Fig. 12: Offset snapping illustration.

Fig. 13: Concentric circle snapping.

5 Results and Discussions

The first example demonstrates the geometry decomposition using freeform strokes to obtain better mesh quality. The second example demonstrates the use of the proposed UI on defeaturing to obtain a meshable model. In the last example we show the potential of using the proposed interface to control the orientation and the size of the mesh.

5.1 Decomposition with pen-based interface for hexahedral meshing

The hexahedral mesh generation is a hard problem even for simpler geometries and generally requires geometry decomposition as a pre-process. For example, hex meshing algorithms such as sweeping and mapping fail to mesh the model shown in Fig. 16(a) since the model contains two holes in orthogonal direction. The model can be meshed via a sweeping algorithm if it can be decomposed so that the two holes are in two different sub-domains. As shown in Fig. 15, the ITEM does not provide any cutting surfaces in ideal region between the two holes as there are no geometric features in this ideal region. Using the CLI user will be able to webcut the volume using a planar surface; however, the mesh quality will not be good because of the bad angles. The cutting surface in the ideal region should be orthogonal to the sweeping source and target surfaces (see Fig. 16(b)). Using the pen-based interface the user can draw a stroke in the desired direction orthogonal to the source and the target surface in the ideal region as shown in Fig. 16(b). Fig. 16(c) shows the meshed model and the mesh quality is shown in Fig. 16(d). The decomposition via the proposed interface has given a min Scaled Jacobian of 0.7255. While cutting with a planer surface using the command line the min scaled Jacobian was 0.6513.

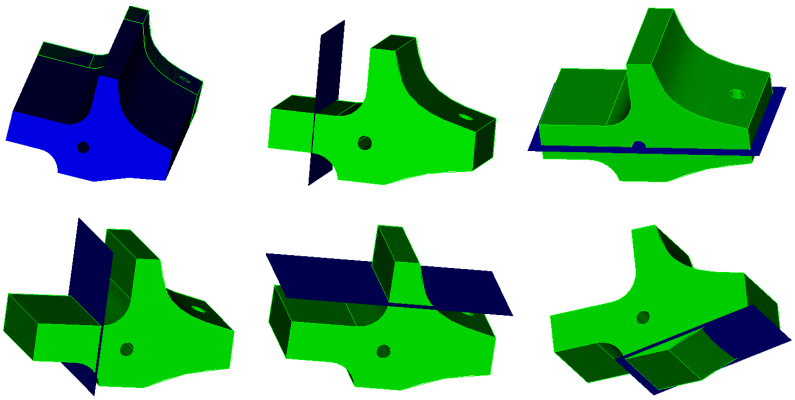
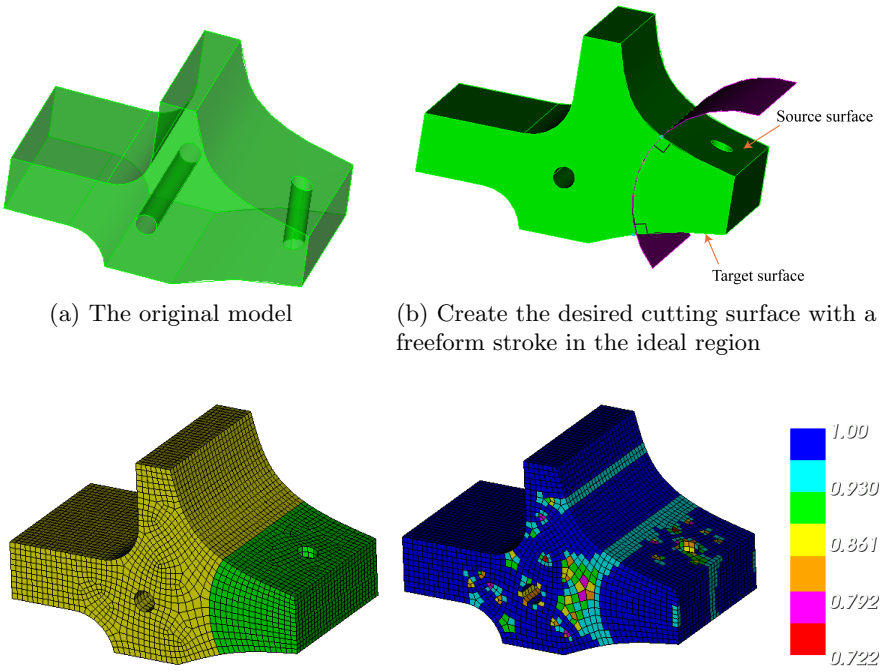


Fig. 15: None of the suggested cutting surfaces by ITEM locate in the ideal region.

5.2 Defeaturing via Pen-Based Interface

In the last couple of decades the CAD technology has evolved significantly and has enabled the users in creating detailed CAD models; however, this has brought new challenges to the mesh generation phase. In order to obtain a



(a) The original model

(b) Create the desired cutting surface with a freeform stroke in the ideal region

(c) The decomposition result via proposed pen-based interface

(d) The hex mesh quality result uses the Scaled Jacobian matrix

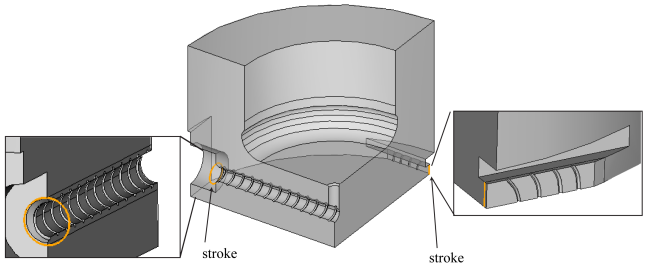
Fig. 16: Decomposition with freefrom stroke via pen-based interface.

conformal high-quality hex mesh with a small number of mesh element, irrelevant detailed features should be removed. Though much focus on automatic defeaturing has been given [19], user interactions are still required in many cases, especially during hexahedral mesh generation.

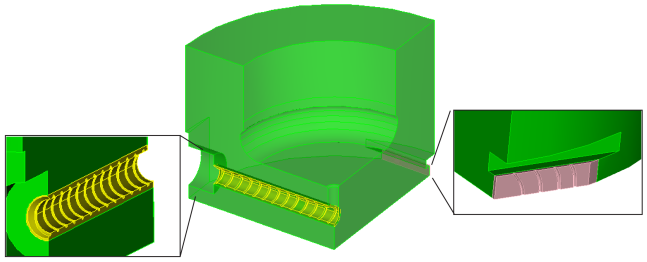
In this example, we demonstrate an application of the proposed pen-based UI in the area of defeaturing. Fig. 17(a) shows a typical industrial model containing two small features: threads and slots. Fig. 17(a) also shows the two input strokes marked on the model surface to indicate the regions that need to be defeatured. Fig. 17(b) shows that the small threads and slots are decomposed from the main solid and in Fig. 17(c) those features are removed. Fig. 17(c) and Fig. 17(d) show the meshable sub-domains and the final mesh obtained by combining the proposed UI with other existing tools in CUBIT.

5.3 Controlling mesh orientation and size via freeform stroke

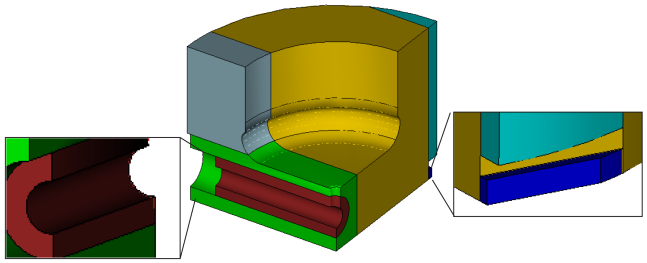
In this example, we show the potential of our pen-based interface in controlling mesh orientation and size. Currently, there are no general automatic hex meshing algorithms that can utilize tensor fields [20] based on physical



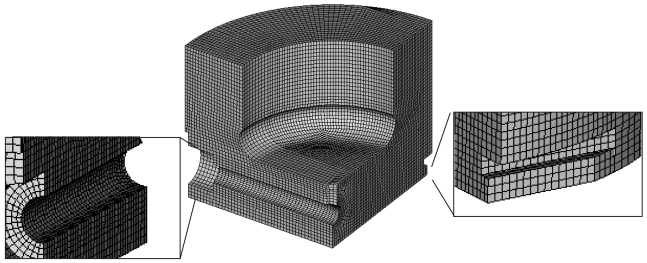
(a) The original model: unmeshable due to small features, the slot and threads



(b) The defeatured model



(c) The decomposed and defeatured model

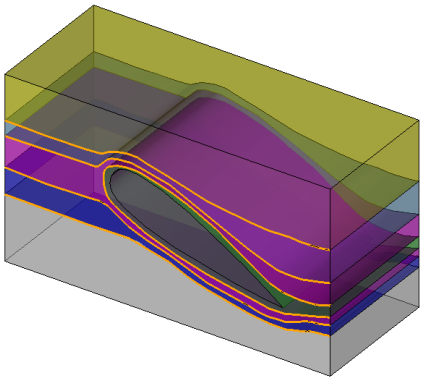


(d) All hex mesh output

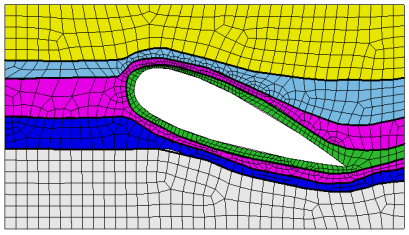
Fig. 17: Defeaturing to achieve a meshable model.

parameters such as temperature distribution, stress contours, or streamlines in flow problems. With the proposed UI, the user can draw freeform strokes along the contours based on domain knowledge. Fig. 18(a) shows a typical streamline pattern in the flow around an airfoil. To generate a hex mesh that aligns with the flow pattern, the user can draw strokes along typical streamlines. The model is then decomposed along the strokes as shown in Fig. 18(b). Fig. 18(c) shows that mesh is oriented along the streamlines and the mesh size at the boundary layer at the bottom of the airfoil can be controlled by maintaining a required spacing between the freeform strokes. A smaller spacing produces finer elements at boundary layers and a larger spacing produces coarser elements.

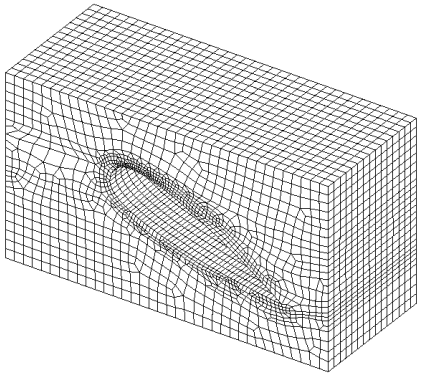
(a) A flow pattern around an airfoil [21]



(b) Drawing strokes along the streamlines to decompose the model. The strokes are marked as orange



(c) The oriented mesh along the streamlines



(d) The oriented hex mesh along the streamlines

Fig. 18: Controlling mesh orientation.

6 Conclusions

This paper proposes a new application of pen-based UI in the area of geometric processing for mesh generation. The proposed UI makes the manual geometry decomposition for hexahedral meshing very intuitive, easier, and faster. The proposed approach beautifies and snaps the freehand input strokes (drawn directly on the CAD model) and uses gestures to create cutting surfaces to decompose the CAD model into meshable sub-domains. The pen-based UI can also be used in controlling the size and orientation of the hex mesh and in defeaturing irrelevant small features. Work is underway to add more features to the UI and to apply it to related areas in pre-processing stage.

Acknowledgement. We would like to thank Dr. Soji Yamakawa, Mr. Ved Vyas and Mr. Erick Johnson for their valuable advises. We would also like to thank Cubit users, in particular Kristin Dion and Ricardo Garcia, for their valuable feedback.

References

1. M. Hardwick in *DART System Analysis Presented to Simulation Sciences Seminar*, 2005.
2. S. J. Owen, "A survey of unstructured mesh generation technology," in *Proceeding of the 7th International Meshing Roundtable, Sandia National Laboratories*, pp. 239–267, 1998.
3. Sandia National Laboratories, "Cubit 12.1 on-line user's manual: Web cutting," in $http://cubit.sandia.gov/help-version12.1/cubithelp.htm$.
4. N. Yang, A. Forsberg, J. Shepherd, R. Garcia, and K. Merkley, "Mark-it: A marking user interface for cutting decomposition time," in *Proceeding of the 13th International Meshing Roundtable, Sandia National Laboratories*, pp. 231–242, 2004.
5. S. J. Owen, B. W. Clark1, D. J. Melander, M. Brewer, J. F. Shepherd, K. Merkley, C. Ernst, and R. Morris, "An immersive topology environment for meshing," in *Proceeding of the 16th International Meshing Roundtable, Sandia National Laboratories*, pp. 553–578, 2007.
6. Y. Lu, R. Gadh, and T. Tautges, "Volume decomposition and feature recognition for hexahedral mesh generation," in *Proceeding of the 8th International Meshing Roundtable, Sandia National Laboratories*, pp. 269–280, 1999.
7. T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A sketching interface for 3D freeform design," in *Proceeding of the 26th annual conference on Computer graphics and interactive*, pp. 409–416, 1999.
8. S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan, "A suggestive interface for image guided 3D sketching," in *Proceeding of the SIGCHI conference on Human factors in computing systems*, pp. 591–598, 2004.
9. S. Bae, R. Balakrishnan, and K. Singh, "Ilovesketch: As-natural-as-possible sketching system for creating 3D curve models," in *Proceeding of the 21st annual ACM symposium on User interface software and technology*, pp. 151–160, 2008.

10. C. Yang, D. Sharon, and M. van de Panne, "Sketch-based modeling of parameterized objects," in *Proceeding of the SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, p. 89, 2005.
11. L. Kara and K. Shimada, "Construction and modification of 3D geometry using a sketch-based interface," in *Proceeding of the EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, pp. 59–66, 2006.
12. Y. Kho and M. Garland, "Sketching mesh deformations," in *Proceeding of the Symposium on Interactive 3D Graphics and Games*, pp. 147–154, 2005.
13. A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or, "A sketch-based interface for detail-preserving mesh editing," in *Proceeding of the ACM SIGGRAPH 2005.*
14. V. Cheutet, C. Catalano, J. Pernot, B. Falcidieno, F. Giannini, and J. Leon, "3D sketching for aesthetic design using fully free-form deformation features," *Computer Graphics*, vol. 29, no. 6, pp. 916–930, 2005.
15. L. Eggli, B. Brüderlin., and G. Elber, "Sketching as a solid modeling tool," in *Proceeding of the 3th ACM symposium on Solid modeling and applications*, pp. 313–322, 1995.
16. M. Masry, D. Kang, and H. Lipson, "A freehand sketching interface for progressive construction of 3D objects," *Computers and Graphics*, vol. 29, no. 4, pp. 563–575, 2005.
17. G. Taubin, "Curve and surface smoothing without shrinkage," in *Computer Vision*, pp. 852–857, 1995.
18. I. Song and K. Shimada, "Sketch-based computer-aided design tool for configuration design of automobile instrument panel," *Computer Aided Design and Application*, vol. 6, no. 5, pp. 585–594, 2009.
19. W. R. Quadros and S. J. Owen, "Defeaturing CAD models using a geometry-based size field and facet-based reduction operators," in *Proceeding of the 18th International Meshing Roundtable, Sandia National Laboratories*, pp. 301–318, 2009.
20. V. Vyas and K. Shimada, "Tensor-guided hex-dominant mesh generation with targeted all-hex regions," in *Proceeding of the 18th International Meshing Roundtable, Sandia National Laboratories*, pp. 377–396, 2009.
21. Massachusetts Institute of Technology, "Marine hydrodynamics," in $http://web.mit.edu/13.021/13021003/Lifting20surfaces/lectureC.htm$.