EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos
(Guest Editors)

Volume 25 (*2006*), *Number 3*

Easy Mesh Cutting

Zhongping Ji, Ligang Liu†, Zhonggui Chen, and Guojin Wang

Department of Mathematics, Zhejiang University, China
State Key Lab of CAD&CG, Zhejiang University, China

Abstract

We present Easy Mesh Cutting, an intuitive and easy-to-use mesh cutout tool. Users can cut meaningful components from meshes by simply drawing freehand sketches on the mesh. Our system provides instant visual feedback to obtain the cutting results based on an improved region growing algorithm using a feature sensitive metric. The cutting boundary can be automatically optimized or easily edited by users. Extensive experimentation shows that our approach produces good cutting results while requiring little skill or effort from the user and provides a good user experience. Based on the easy mesh cutting framework, we introduce two applications including sketch-based mesh editing and mesh merging for geometry processing.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometric algorithms, languages, and systems

1. Introduction

Triangular meshes are commonly used to define geometric objects in computer graphics applications. Although this representation is useful for rendering, this low-level description is often inadequate for acquiring meaningful information required in some tasks such as object recognition, feature analysis, and shape matching. In searching for good high-level description, psychological studies have shown that human shape perception is partly based on meaningful decomposition.

Mesh segmentation, like image segmentation, is the process of partitioning a mesh into more visually simple and meaningful components. It has recently become of interest and a key ingredient in many mesh manipulation algorithms.

However, human perception is extremely complicated. Both the concept of "meaningful" and the component decomposition are highly subjective and hard to qualify, not to mention computing the meaningful decomposition automatically. Thus it remains a challenge to define useful high-level structures for arbitrary meshes.

We aim to cut a mesh into components that are consistent with user intention and human shape perception. Generally, users cut a meaningful component from its underlying mesh by specifying which parts of the mesh belong to the "foreground" (the part you want to cut out) and the rest to the background. Although it is quite easy for a human to specify foreground and background by saying something like "cut out the ears from the bunny model" to another human, the computer is still a long way from the sort of cognitive object understanding required to do this work unassisted.

1.1. Our approach

We propose *Easy Mesh Cutting*, which is a novel interactive sketch-based system for mesh cutting. The sketch-based user interface is inspired by the emerging sketch-based applications [IMT99, KG05], particularly Lazy Snapping [LSTS04] for image cutout.

As shown in Figure 1(a) and (b), the user simply and quickly draws freehand sketches on the mesh in our system. The freehand strokes roughly mark out the subpart of interest (as foreground) and background (green sketch for foreground and red sketch for background). The subpart of interest is then cut out from the mesh (shown in pink in in Figure 1(a) and (b)) in real time via an efficient region growing segmentation algorithm based on an improved feature sensitive metric (see Section 3). The cutting boundary is almost correct and is smoothed by snakes (see Section 5.1). The user

† Correspondence: ligangliu@zju.edu.cn

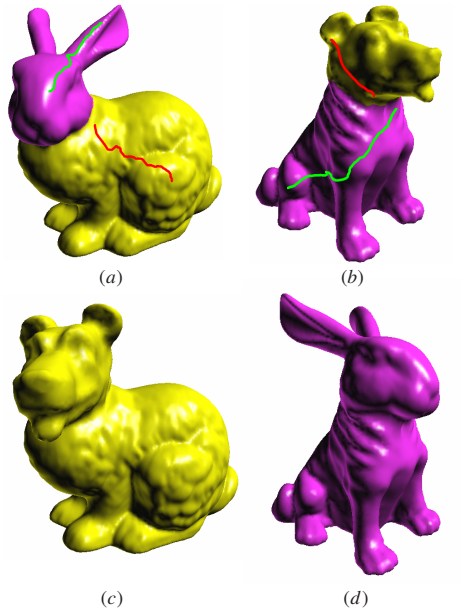


Figure 1: *Easy mesh cutting examples. Users draw two free-hand sketches on meshes to mark out foreground (in green) and background parts (in red). The bunny model (a) and the dog model (b) are cut into head and body parts respectively; (c) pasting dog head onto bunny body; (d) pasting bunny head onto dog body.*

can navigate the mesh to inspect the boundary polygon that is unsatisfactory and then use a brush to override a segment of polygons or click and drag polygon vertices directly.

Our system is efficient and easy to use. Many experiments show that our system extracts perceptual components precisely and efficiently while requiring little skill or effort from the user. Our method improves traditional, fully automatic segmentation by involving human users in the process, yet minimizing user input and providing realtime feedback.

To our knowledge, this is the first time that an intuitive and simple sketch-based interface has been provided for both expert users and untrained users to cut out a mesh surface in such an easy manner.

1.2. Overview

The remainder of this paper is organized as follows. In Section 2, we briefly review related work. An improved feature sensitive metric is proposed in Section 3. Section 4 describes our easy mesh cutting framework. Section 5 discusses the approaches for optimizing the cutting boundary. Additional experimental results are illustrated in Section 6. We conclude the paper in Section 7 with the summary and future work.

2. Related work

There is a great deal of work on mesh segmentation and its application in the literature. Here we review, only the most relevant work, emphasizing mesh decomposition and sketch-based user interface techniques.

Human perception. Studies have shown that human perceptual system uses meaningful feature decomposition as the primary index into recognition of shapes [Bie87]. Various algorithms and theories have been provided to describe human perception. For example, the minima rule and salience theory have been proposed in human cognitive vision theory [HR84, HS97]. The minima rule states that human perception usually divides a surface into parts along negative minima of the principal curvatures. The part salience theory provides factors of the object size relative to the whole object, the degree to which it protrudes, and the strength of its boundaries to determine the salience of segments. But human perception is more complicated than the minima rule or part salience theories.

Mesh segmentation. Mesh segmentation techniques can be classified into patch-type and part-type. Patch-type segmentation is often used for texture mapping, building charts, simplification and remeshing [LPRM02,GCO06]. Part-type segmentation divides a mesh into meaningful parts without restricting the part topology. Several approaches to automatically segment mesh into meaningful components have been proposed in the past. The minima rule has been used for mesh segmentation recently [KT03]. The watershed-based scheme by Magan and Whitaker [MW99] favors partition boundaries along high curvature regions. A mesh segmentation method using mean shift is proposed in [YLL*05]. Lee et al. [LLS*05] present an intelligent manual scissoring tool for meshes.

Sketch-based user interface. In recent years, sketch-based interfaces have emerged as an approach to enhance user interactions for various design activities ranging from image segmentation to geometric design. They are expected to provide flexible interaction between computers and users that do not hinder creative thinking. Sketch-based interaction has been successfully used in object selection in image [TA01] and graph cut image segmentation [LSTS04]. There has also been substantial interest of developing intuitive sketch-based interactive techniques for 3D applications by sketching curves in the image plane as the direct interaction in 3D space is often a confusing and cumbersome task for non-expert users [IMT99, KG05, NSACO05].

3. Feature sensitive metric

Pottmann et al. introduced the *isophotic metric* [PSH*04], a new metric on surfaces, in which the length of a surface curve is not just dependent on the curve itself, but also on the variation of the surface normals along it. They consider the field of unit normal vectors $\mathbf{n}(\mathbf{x})$ attached to the surface

points $\mathbf{x} \in S$ as a vector-valued image defied on the surface. One can map each surface point $\mathbf{x}$ to a point $\mathbf{x}_f = (\mathbf{x}, w\mathbf{n})$ in R^6 where w denotes a non-negative weight, whose magnitude regulates the amount of feature sensitivity and the scale on which one wants to respect features. In this way, S is associated with a 2-dimensional surface $S_f \subset R^6$. By measuring distances of points and lengths of curves on S_f instead of S a feature sensitive metric on the surface is introduced [PSH*04]. Pottman et al. have used this feature sensitive metric for feature sensitive morphology on surfaces [PSH*04] and for the design of curves on surfaces which are well aligned with the surface features [PLH*05].

The isophotic metric distance between two points $\mathbf{p}$ and $\mathbf{q}$ on a surface is dependent on the path Γ on the surface connecting the two points and is defined via the arc length differential as the following:

$$d_\Gamma(\mathbf{p},\mathbf{q}) = \int_\Gamma ds + w \int_{\Gamma^*} ds^*,$$

where ds is the arc element of Γ, ds^* is the arc element of Γ^* which is the Gaussian image of Γ, and $w > 0$ is the weight of the isophotic components.

As the minima rule is an approximated perceptual criterion in vision theory, we improve the isophotic metric by considering the curvature as:

$$d_\Gamma(\mathbf{p},\mathbf{q}) = \int_\Gamma ds + w \int_{\Gamma^*} ds^* + w^* \int_\Gamma f(k_D) ds,$$

where k_D is the normal-section curvature in the direction tangent to the path Γ, f is a function of curvature, and $w^* > 0$ is the weight of the curvature metric.

As pointed out by the minima rule, all negative minima of the principal curvatures (along their associated lines of curvature) form boundaries between perceptual parts. Therefore, we augment the effect of negative curvature in the improved isophotic metric by setting curvature function as follows:

$$f(k_D) = \begin{cases} k_D, & k_D \geq 0, \\ g(|k_D|), & k_D < 0, \end{cases}$$

where $g(x)$ is an augmentation function that can be set like $g(x) = 3x$, $g(x) = x^2$, or $g(x) = e^x$, etc. From experiments we find the augmentation function $g(x) = e^x - 1$ performs well in finding the concave boundary of the visual part. The principle behind it is that the value of $g(x)$ becomes extremely large when the negative curvature is large, which is in accord to the minima rule. Thus the improved isophotic metric is a feature sensitive metric according to the minima rule.

In our implementation, we approximate the computation of the improved isophotic metric in a discrete form for triangular meshes. For a vertex $\mathbf{p}$ and its neighbor vertex $\mathbf{q}$ on the mesh, the distance between them are calculated as the following:

$$\bar{d_\Gamma}(\mathbf{p},\mathbf{q}) = |\mathbf{p}-\mathbf{q}| + \omega |n_\mathbf{p} - n_\mathbf{q}| + \omega^* f(k_{\overrightarrow{\mathbf{pq}}}),$$

where $n_\mathbf{p}$ and $n_\mathbf{q}$ are respectively the normals of vertices $\mathbf{p}$ and $\mathbf{q}$, $k_{\overrightarrow{\mathbf{pq}}}$ is the directional curvature along the line direction $\overrightarrow{\mathbf{pq}}$.

The directional curvature at the mesh surface can be computed by various methods. We adopt a simple linear algorithm in which principal curvatures and principal directions are obtained by computing in closed form the eigenvalues and eigenvectors of certain 3×3 symmetric matrices defined by integral formulas, and closely related to the matrix representation of the tensor of curvature [Tau95]. Then the directional curvature can be estimated based on the principal curvatures and principal directions by Euler formula.

4. Easy mesh cutting framework

In this section we describe our novel framework of easy mesh cutting. First we present the sketch-based user interface and later, in successive sections, we describe each step in more details.

4.1. User interfaces

The input data sets are triangular mesh models that we want to cut. Our system allows users to use marker sketches to specify the part of interest on the meshes.

The user loads a mesh model into our system, navigates it freely, and selects an appropriate view position. To specify an interesting part on the mesh, the user marks a few free-hand sketches on the image plane by dragging the mouse cursor while holding a button (left button indicating the foreground and right button for the background). The strokes for foreground and background parts are displayed in green and red respectively (Figure 1(a),(b)). This marking UI is inspired by the similar UI presented in [LSTS04] for image cutout.

The sketches in the image plane are then projected onto the surface of the mesh in 3D world space by computing the point of intersection of rays from the view point through the points on the sketch curve.

These high level sketches specified by users need not be very precise. As shown in Figure 1(a),(b) and Figure 5, most marking strokes are actually far from the component boundaries. After the user finishes marking the sketches, our system starts the cutting process based on the sketch information. The user then inspects the cutting result on screen and decides if more strokes need to be marked. Therefore, it is extremely crucial that our system generates the cutting boundary in a short time. To achieve this goal, our system adopts a novel and fast region growing algorithm to compute the cutting boundary described in the following section.

4.2. Mesh segmentation approach

As our system needs real-time feedback to user's interaction, it is critical that the mesh segment approach should be fast

enough to generate the cutting boundary with very little delay.

It is straightforward to think of using graph cut to do the segmentation task inspired from the work of Lazy Snapping [LSTS04]. Lazy Snapping adopts a graph cut algorithm to generate the foreground object boundary by maximizing the color similarity inside the object. The system is interactive as it is easy to compute the likelihood energy in color space. But the graph cut on 3D mesh performs very badly in computation time since it is time consuming to compute the geodesic distance between every two points on the mesh [KT03].

We apply a region growing based algorithm [WL97, PKA03] for mesh segmentation in our system, which avoid computing the geodesic distance between every two points on the mesh. The main difference between various algorithms which use region growing is in the criteria which determines if an element can be added to an existing cluster. We use the improved isophotic metric as a distance measurement between two adjacent points on the mesh. As we have seen that the isophotic metric simplifies the definition of local neighborhoods for shape detection. It also simplifies the implementation of region growing algorithms. Region growing based on this feature sensitive metric can easily be stopped at features on the mesh.

The region growing algorithm starts with different seed vertices from marker sketches simultaneously and grows several sub-meshes according to the improved isophotic metric incrementally.

Every mesh vertex on (or near) the input sketches is labelled as "F" (foreground part) or "B" (background part) and all other vertices are labelled "U" (unknown). For each unknown vertex v, we define a triplet (v,m,d) where d is the improved isophotic distance between v and the nearest marked vertex $v^* \in N$ and m is the marking label of v^*. A queue Q is defined as a set of triplets and is initialized as empty. The vertex that has the minimum value d in Q is selected and labelled by its marker m. Then it is removed from Q but its neighboring vertices with label "U" are updated and added to Q. This process continues until all the vertices are labelled as "F" or "B".

We summarize the steps of our approach as following:

Input: Triangular mesh M and foreground and background sketches on the mesh.

Output: Foreground part and background part of the mesh.

Step 1. Label each vertex on foreground and background sketches as "F" and "B" respectively. Set $Q = \Phi$. Denote N the set of labelled "F" or "B".

Step 2. For each vertex in N, add the triplet of its nearest (by improved isophotic distance) vertex in N^c into Q.

Step 3. Sort Q by value d.

Step 4. Find minimum d_{min} of (v,m,d) in Q, label v as m, set $Q = Q - \{(v_{min}, m_{min}, d_{min})\}$ and set $N = N \cup \{v\}$.

Step 5. Repeat Step 2 to Step 4 until all the vertices are labelled as "F" or "B".

Note that it is straightforward to extend the above algorithm to segmentation according to more than two marker sketch specifications.

4.3. Preprocessing

In order to make the overall cutting process faster for very large meshes, we perform the cutting process over the simplified meshes. In our system, the simplification [GH97] is done before the user begins to navigate it and place sketches. After the user specifies the hints to some foreground and background regions on the model, our system separates background and foreground regions using region growing segmentation approach on the simplified mesh. Results from this are used to achieve the final segmentation on the original model using the hierachical correspondence between the simplified mesh and the original mesh.

5. Cutting boundary optimization

As the vertices and edges of cutting boundary are constrained to lie on the mesh vertices and edges, the above approach might generate jaggy boundaries between the components. We provide both automatic and manual mechanisms to optimize the cutting lines to avoid jaggy boundaries.

5.1. Boundary refinement by snake

The snake was originally developed to find features in images [KWT88] in which curves moving in a domain are called *snakes* because they move like snakes.

In our system, the cutting boundaries are considered as snakes on the mesh and can be refined by snake movements. Early attempts for moving snakes on 3D meshes suffered from the parameterizaiton artifacts [LL02]. For the sake of real-time snake movements, we introduce a modified method to move snakes directly and quickly on 3D mesh based on the parameterization free active contour models presented in [BK04].

We represent a snake by a polygon S on a 3D triangular mesh and the snake vertices are called *snaxels* (snake elements), see Figure 2(a). To guarantee that the snake is actually embedded on the underlying mesh, we enforce two consistency constraints on the snaxels: the snaxels have to be on mesh vertices (called vertex snaxels) or lie on mesh edges (called edge snaxels) and the segments of the snake have to lie in the interior of triangles.

The movement of the snaxel v_i is governed by the energy which is defined as:

$$E_{snake}(v_i) = E_{int}(v_i) * E_{ext}(v_i).$$

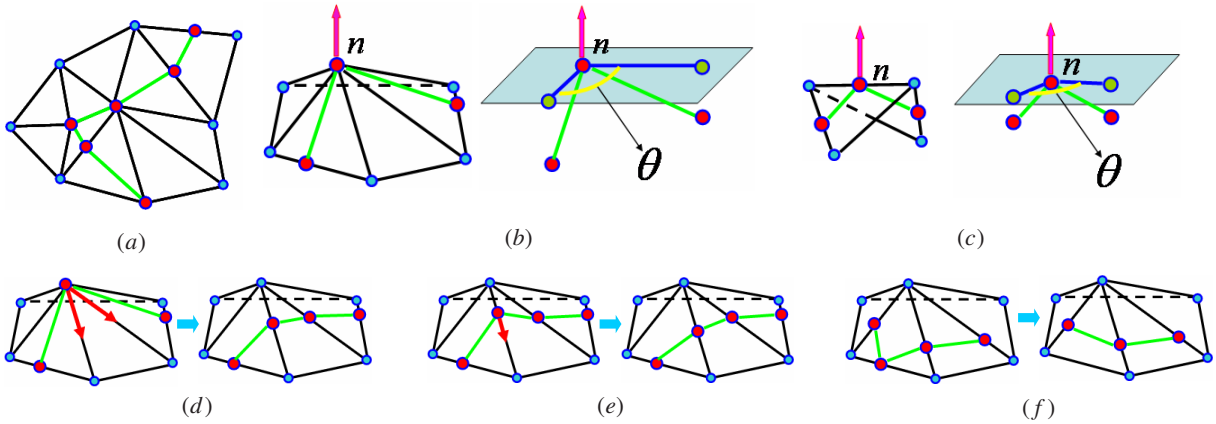


Figure 2: *Snakes on 3D mesh. (a) Snake (with red vertices and green edges) on 3D mesh (with blue vertices and black edges); (b) internal energy computation at a vertex snaxel; (c) internal energy computation at an edge snaxel; (d) one vertex snaxel is split into two edge snaxels; (e) one vertex snaxel slides along its underlying edge; (f) one redundant snaxel is removed.*

The internal energy at snaxel v_i is defined as

$$E_{int}(v_i) = \theta,$$

where $\theta \in [0,\pi]$ is the angle between the projection lines of its adjacent snake edges over the tangent plane at v_i, see Figure 2(b) and (c). The external energy at snaxel v_i is defined as

$$E_{ext}(v_i) = |k_{\min}(v_i)| + |k_{\max}(v_i)|,$$

where $k_{\min}(v_i)$ and $k_{\max}(v_i)$ are the two principal curvature at v_i. It can be seen that increasing the internal energy of the snake makes it smooth and the purpose of the external energy is to attract the snake to sharp features on the mesh.

There are three operations, including snaxel splitting, snaxel sliding, and snaxel removal, on snaxels during the evolution of snakes. When a snaxel runs into a mesh vertex, it is split into several new snaxels that are put on the outgoing edges, see Figure 2(d). The snaxels move along their supporting edges to increase its energy, see Figure 2(e). No two consecutive snake segments should lie in the same triangle. Snaxels adjacent to two such segments can iteratively be removed, see Figure 2(f).

We adopt a greedy algorithm to maximize the snake energy by operating the above three operations in an iterative way. The refinement iteration is fast enough for real-time refining at cutting boundaries in our system. Figure 3 shows an example to illustrate the effect of refining the cutting boundary using our snake technique where the original cutting boundary lines in blue are jaggy whereas the refined boundary lines in green are smoother after snake optimization.

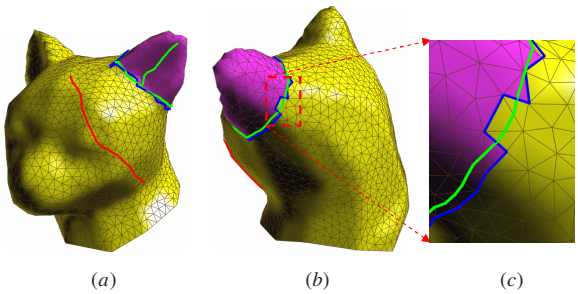


Figure 3: *Illustration of snake refinement. The original cutting boundary lines are shown in blue and the refined boundary lines after snake optimization are shown in green. (a),(b) are different views for the cutting boundary; (c) is the zoom-in view of part of (b).*

5.2. Boundary editing by users

Although the step of region growing with a feature sensitive metric preserves the cutting boundary as accurately as possible and the step of snake optimization refines the cutting boundary as smooth as possible, there still exist unwanted edges or errors in the cutting boundary between the component, especially around ambiguous and non-feature boundaries. Therefore, we provide two simple polygon editing tools for the user to modify the cutting boundary as in [LSTS04].

First, the user can directly drag the vertex over the mesh to adjust the shape of the cutting boundary. Users can also add or remove vertices by simple mouse clicks. Second, the user can draw freehand strokes to replace a segment of the cutting boundary. The vertices on the boundary are recomputed by

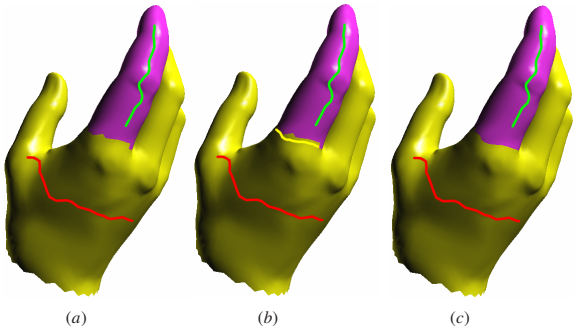


Figure 4: *Overriding brush editing of cutting boundary. (a) The cutting boundary is jaggy; (b) the user draws a short stroke in yellow near the cutting boundary; (c) the modified boundary.*

fitting the shape of user's input stroke. This tool can modify multiple vertices at one time and is more efficient. Note that the user can also adjust viewing angles to get the best editing position for a particular object during the boundary editing.

In Figure 4(a), the cutting boundary is jaggy. The user draws a short stroke in yellow near the boundary, see Figure 4(b). The modified boundary by fitting the shape of the stroke is shown in Figure 4(c).

6. Experimental results

We show some of examples illustrating the applicability and flexibility of our Easy Mesh Cutting approach in this section. All the examples presented in this paper were made on a 3GHz Pentium IV computer with 1G memory. The model is first uniformly scaled into one with a unit bounding box in our system.

The weight parameters ω and ω^* used in the improved isophotic metric measure the importance of normal variation and curvature variation respectively. It is worthwhile to point out that the two parameters are important for computing the isophotic distance which will affact the mesh cutting results. The segmentation is acutally computed according to the minima rule if ω^* is set to be large enough. We set $\omega = 5$ and $\omega^* = 5$ in our system, which have performed well for all the examples shown in our paper.

Figure 1 shows two cutting examples produced in our system. The bunny head (a) and dog body (b) are cut out by simply drawing two sketches on the models respectively. The sketches for specifying foreground and background parts are shown in green and red respectively. The cutting foreground parts are shown in pink.

Figure 5 illustrates some examples produced in our easy mesh cutting system. In Figure 5(a), the user can cut a protrusive cube into different components by specifying different sketches according to his intension. One green stroke and two red strokes are specified in the middle example of Figure 5(a), while only one green stroke and one red stroke are specified in the rightmost example of Figure 5(a).

Our approach works well for meshes with various sizes of features. In Figure 5(b), the ear part and the eye part on a head model are easily cut out by drawing two sketches respectively. We use couples of overriding strokes to editing the cutting boundaries to get better results.

Figure 5(c) shows an example of cutting a high genus component from a model. The user uses two sketches to cutout the tail part with a hole inside from the feline model. Our approach can cutout high-genus components well, which benefits from the region growing algorithm.

Our system can also work well for foreground parts on background surface with much noise. In Figure 5(d), we quickly cut out the relief dragon from a noisy background surface by drawing several strokes. Note that the boundary of the relief dragon is perceptually unclear but our approach has obtained a satisfactory result.

It is worthwhile to point out that it is rather difficult to cut out the subparts shown in Figure 5(b,c,d) using intelligent scissors [FKS*04, LLS*05] as the cutting boundaries in these cases can not be computed by intersecting a slicing plane with the underlying meshes. Furthermore, the cutting results by our system are view independent as we use the geometric properties of the mesh in our approach.

From our experience on using the easy mesh cutting system, users can simply select an interesting part in the model by intuitively and simply drawing freehand sketches on the model. Thanks to ease of manipulation proposed sketch-based interface is suitable both for experienced users and unskilled users (e.g. children) who wish to create new models in the style of masters.

Table 1 lists the running time of some of the mesh cutting examples shown in this paper. The meshes in these examples are not simplified for computing the running time. It takes less than 0.2 second to cut a mesh with over 10k vertices. For large mesh the system first simplifies it into a simplified mesh model with number of vertices smaller than 10k in a short time. As we can see, our approach achieves a good combination of speed, cutting quality, and good user experience.

Model	Vertex Number	Running Time (s)
Hand (Fig. 4)	4,991	0.078
Dog (Fig. 1)	10,000	0.172
Horse (Fig. 8)	19,851	0.297
Bunny (Fig. 1)	34,835	0.594
Triceratops (Fig. 8)	40,706	0.656

Table 1: *Running time for different examples shown in the paper.*

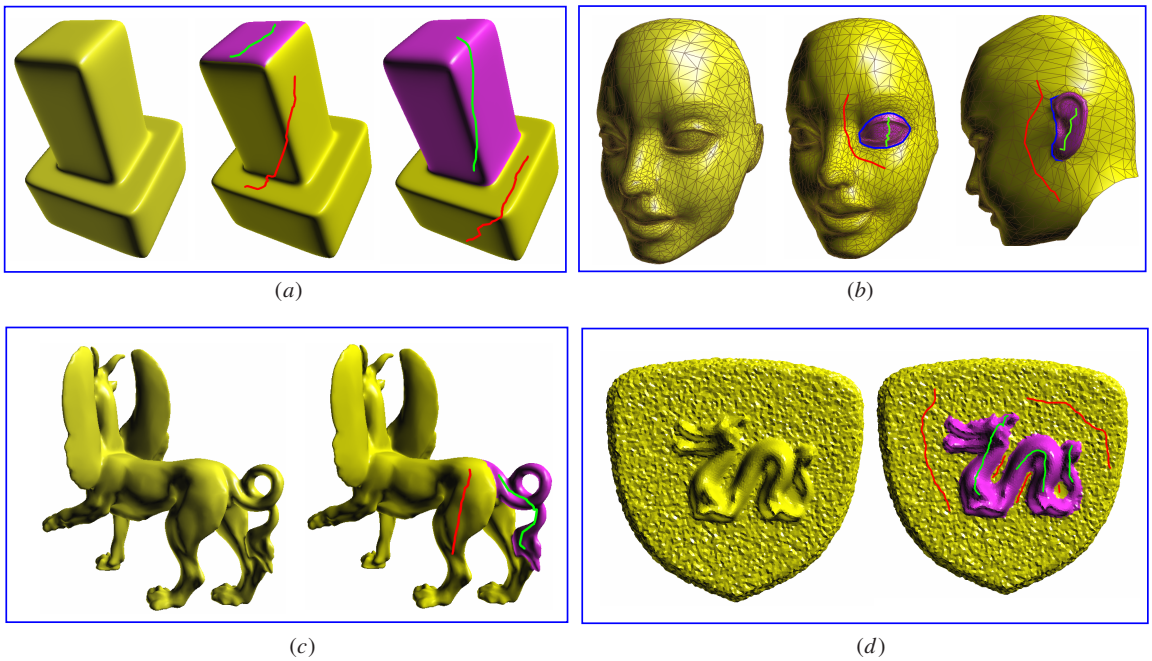


Figure 5: *More mesh cutting examples produced in our easy mesh cutting system. (a) cutting a protrusive cube: user selects one face or the whole cube by different sketches; (b) cutting eye and ear components from a head model; (c) cutting a high genus tail part from feline model; (d) cutting the relief dragon model from a noisy background.*

6.1. Applications

We illustrate a couple of applications based on our easy mesh cutting framework for geometry processing.

Easy Mesh Cutting provides an intuitive interface for component-based mesh editing by drawing sketches in the image plane. Figure 6 shows a simple example of our system in action. The user begins by drawing a foreground sketch along the leg and a background sketch along the body on the feline model, shown in Figure 6(a). The leg is then selected and shown in pink. The user then draws a target curve in light blue indicating the desired deformation (Figure 6(b)). From the foreground sketch and target curve, the system automatically generates the deformation of the leg. We adopt a similar approach based on detail preserving deformation technique [NSACO05] to compute the vertex coordinates of the selected component by adding soft constraints on vertices on the component boundary and vertices on the sketch. A more complex editing example is shown in Figure 7. The deformed dinosaur model shown in Figure 7(b) is edited from the original model shown in Figure 7(a) by 6 individual deformations.

Merging meshes to assemble a new object is another important application of our framework. We use our framework to cut different parts of models from different models and then merge them into one new model [FKS*04]. The partial meshes are merged at their open (mesh) boundaries using the

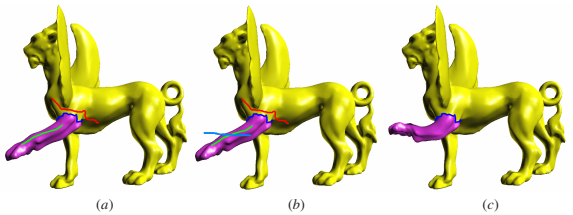
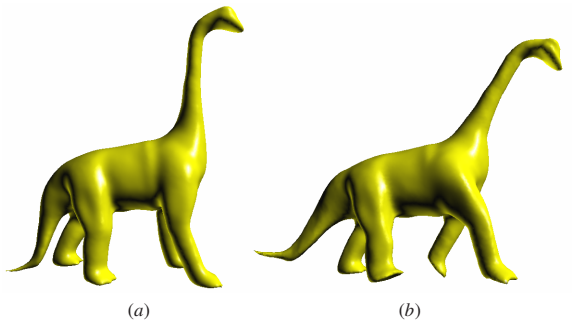


Figure 6: *A simple sketch-based mesh editing. (a) The user selects the leg of feline by drawing a green foreground sketch and red background sketch; (b) The user draws a target sketch along the leg; (c) The foreground sketch and the target sketch induce a deformation of the leg.*

Poisson approach [YZX*04]. In Figure 1, the bunny head is cut from bunny model (Figure 1(a)) and the dog body is cut from dog model (Figure 1(b)) and then they are merged into a new model shown in Figure 1(c). The bunny body and the dog head can also be merged into another new model shown in Figure 1(d). A more complicated mesh merging example is shown in Figure 8. Here we cut the head from a triceratops model, the body from a horse model, the wings from lucy model, and the tail from a dinosaur model, respectively using the easy mesh cutting tool in our system. Then these



(*a*) (*b*)

Figure 7: *The dinosaur model in (b) is edited from (a) by 6 individual deformations.*

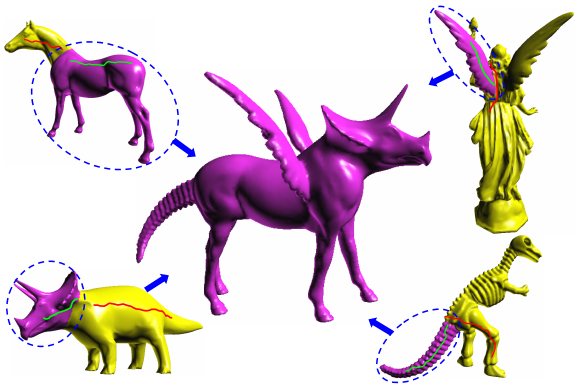


Figure 8: *A complicated mesh merging example.*

cut components are merged into an unknown mythical creature.

The accompanying video shows how our system work on mesh cutting, mesh component editing, and mesh components merging.

7. Conclusions

We present a novel method for cutting out meaningful components from meshes using simple sketches that are drawn in the image plane. The freehand strokes roughly mark out parts of interest and the background. Our system then segments the regions of interest in real time. The cutting boundary can be optimized automatically. The system also provides flexible tools for users to edit the boundary. Our easy mesh cutting approach is beneficial for interactive graphics applications.

The presented approach still has much room for improvements and extensions. The shape of sketches should be investigated to guide the perceptual decomposition combining with the skeleton of the mesh. It is also worthwhile to improve the graph cut algorithm and use it in the sketch-based mesh cutting system. We believe that this extension is feasible but not straightforward.

Acknowledgement. We would like to thank Mr. Lei Zhang for his help in video production. This work is supported by Zhejiang Provincial Natural Science Foundation of China (No. Y105159), National Natural Science Foundation of China (No. 60503067, 60333010), and the National Grand Fundamental Research 973 Program of China (No. 2002CB312101).

References

[Bie87] BIEDERMAN I.: Recognition-by components: A theory of human image understanding. *Psychological Review 94*, 2 (1987), 115–147.

[BK04] BISCHOFF S., KOBBELT L.: Parameterization-free active contour models. *The Visual Computer 20* (2004), 217–228.

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. In *Proc. of SIGGRAPH* (2004), pp. 652–663.

[GCO06] GAL R., COHEN-OR D.: Salient geometric features for partial shape matching and similarity. *ACM Trans. Graph. 25*, 1 (2006), 130–150.

[GH97] GARLAND M., HECKBERT P.: Surface simplification using quadric error bounds. In *Proc. of Siggraph* (1997), pp. 209–216.

[HR84] HOFFMAN D., RICHARDS W.: Parts of recognition. *Cognition 18* (1984), 65–96.

[HS97] HOFFMAN D., SIGNH M.: Salience of visual parts. *Cognition 63* (1997), 29–78.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3D freeform design. In *Proc. of Siggraph* (1999), pp. 409–416.

[KG05] KHO Y., GARLAND M.: Sketching mesh deformations. In *Proc. of the ACM Symposium on Interactive 3D Graphics* (2005), pp. 147–154.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. In *Proc. of SIGGRAPH* (2003), pp. 954–961.

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: Active contour models. *International Journal of Computer Vision 1*, 4 (1988), 321–331.

[LL02] LEE Y., LEE S.: Geometric snakes for triangular meshes. In *Proc. of Eurographics* (2002), pp. 229–238.

[LLS*05] LEE Y., LEE S., SHAMIR A., COHEN-OR D., SEIDEL H.-P.: Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design 22*, 5 (2005), 444–465.

[LPRM02] LEVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. In *Proc. of Siggraph* (2002), pp. 362–371.

[LSTS04] LI Y., SUN J., TANG C.-K., SHUM H.-Y.: Lazy snapping. In *Proc. of SIGGRAPH* (2004), pp. 303–308.

[MW99] MANGAN A., WHITAKER R.: Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 308–321.

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. In *Proc. of Siggraph* (2005), pp. 1142–1147.

[PKA03] PAGE D., KOSCHAN A., ABIDI M.: Perception-based 3D triangle mesh segmentation using fast marching watersheds. In *Proc. of IEEE Computer Vision and Pattern Recognition(Volume II)* (2003), pp. 27–32.

[PLH*05] POTTMANN H., LEOPOLDSEDER S., HOFER M., STEINER T., WANG W.: Industrial geometry: recent advances and applications in CAD. *Computer-Aided Design 37* (2005), 751–766.

[PSH*04] POTTMANN H., STEINER T., HOFER M., HAIDER C., A.HANBURY: The isophotic metric and its application to feature sensitive morphology on surfaces. In *Proc. of ECCV (Part IV)* (2004), pp. 560–572.

[TA01] TAN K., AHUJA N.: Selecting objects with free-hand sketches. In *Proc. of CVPR* (2001).

[Tau95] TAUBIN G.: Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. of ICCV* (1995), pp. 902–907.

[WL97] WU K., LEVINE M.: 3D part segmentation using simulated electrical charge distributions. *IEEE transactions on pattern analysis and machine intelligence 19*, 11 (1997), 1223–1235.

[YLL*05] YAMAUCHIY H., LEE S., LEE Y., OHTAKE Y., BELYAEVY A., SEIDEL H.-P.: Feature sensitive mesh segmentation with mean shift. In *Proc. of Shape Modeling International* (2005), pp. 236–243.

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. In *Proc. of Siggraph* (2004), pp. 644–651.